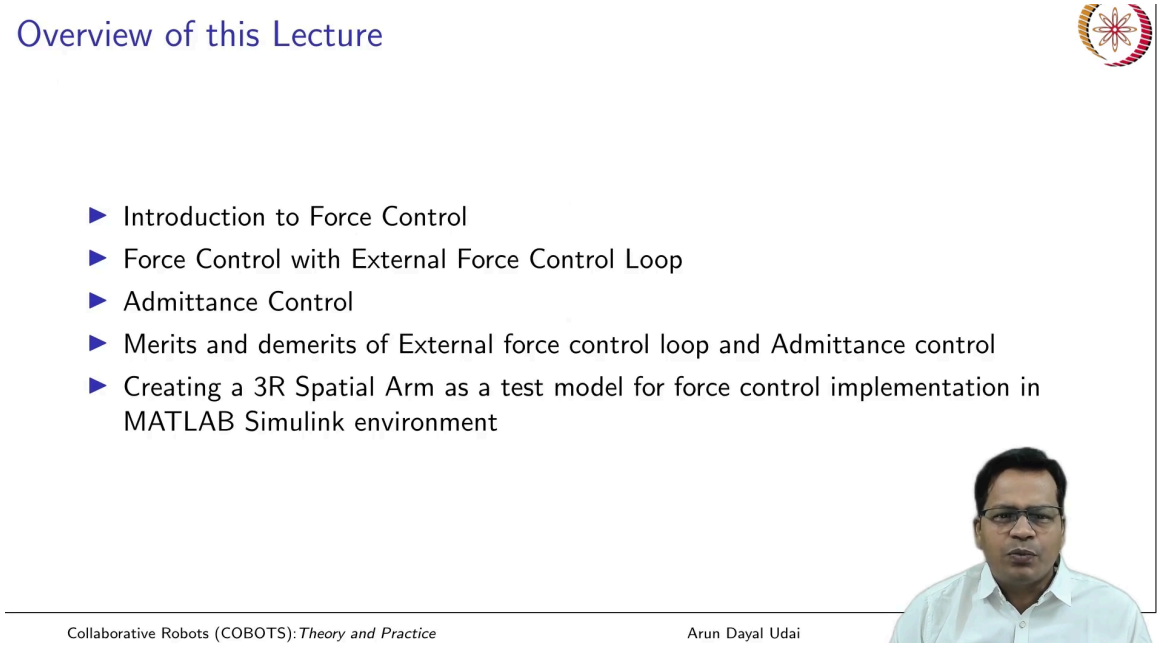
NPTEL Online Certification Courses
COLLABORATIVE ROBOTS (COBOTS): THEORY AND PRACTICE
Dr Arun Dayal Udai
Department of Mechanical Engineering
Indian Institute of Technology (ISM) Dhanbad
Week: 08
Lecture: 33

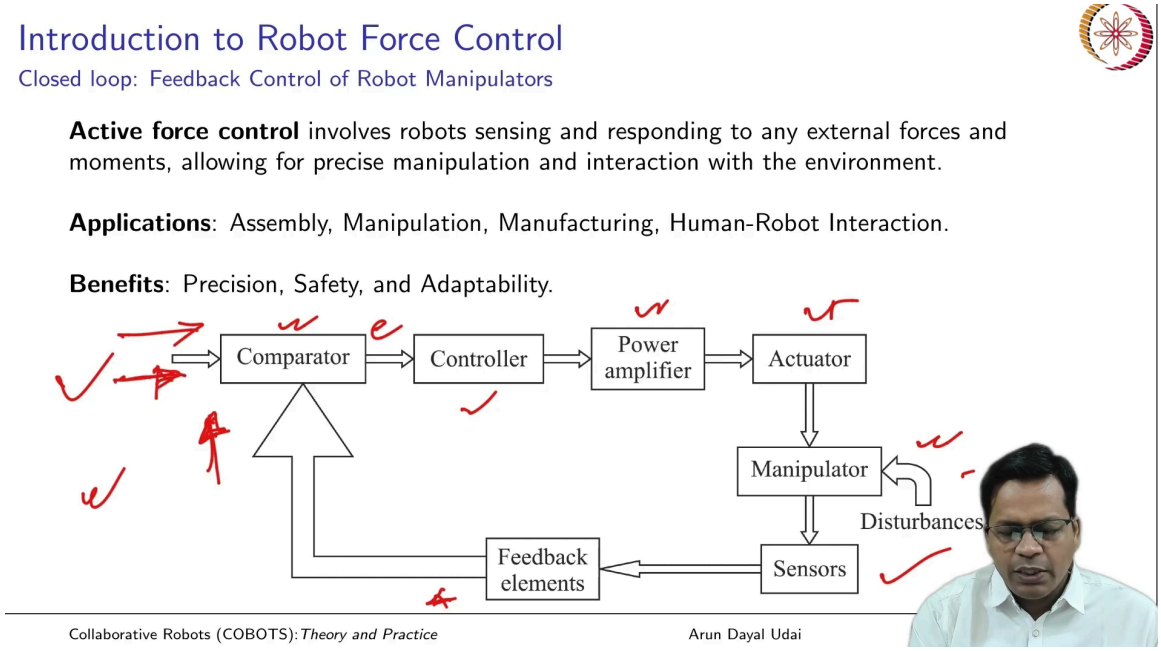
External Force Control Loop and Admittance Control

Welcome to the 8th week of the course Collaborative Robots: Theory and Practice. This week, we will discuss on Force Control of Serial Chain Robots. Using force control, we enable the robot system to handle compliant robotic tasks. It uses the principle of Robot Dynamics and Control. With the background that we have developed in the previous weeks, we are now ready to study Force Control. So, let us start with the first lecture of this week, which is on external Force Control Loops and Admittance Control.



So, in this lecture, I will be discussing on Force Control. I will just introduce you to that. We will see Force Control with External Force Control Loop. We will see what admittance control is. We will discuss the merits and demerits of the external force

control loop and admittance control. Later, I'll create a 3R Spatial Arm as a test model for force control implementation in the MATLAB Simulink environment. So, let us begin.



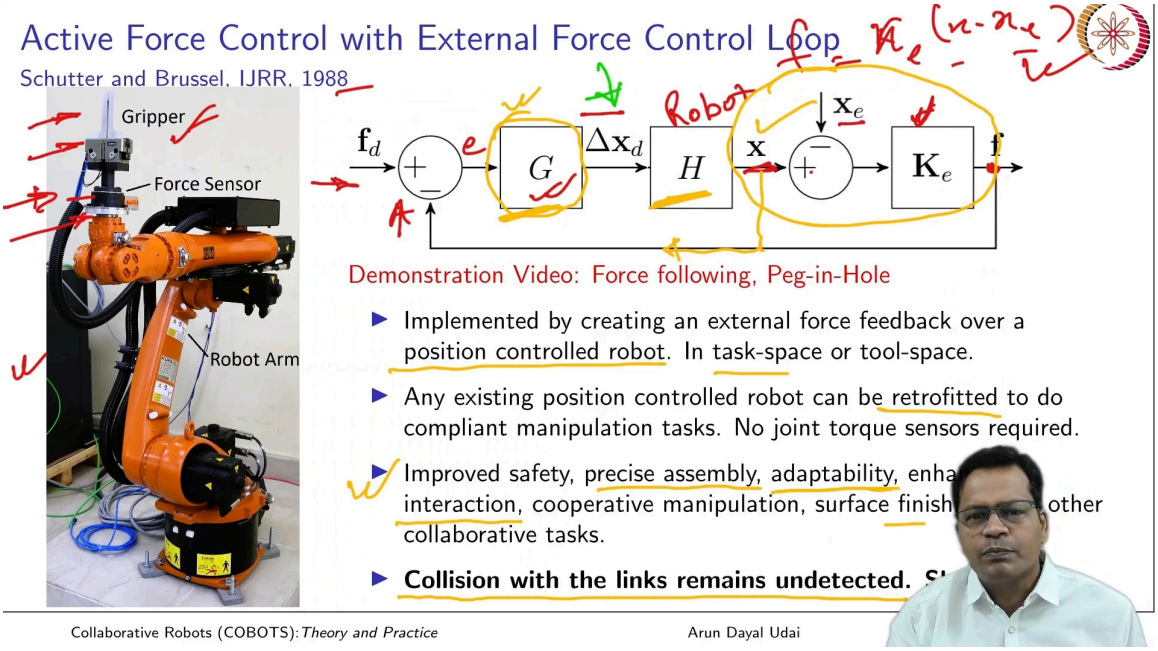
So, what is force control, as you have already seen in my earlier week, which is on week one. I have shown you Various applications of force control, that is, through assembly operation, manipulation, manufacturing and robot-human interaction. Those were the applications of Cobot. So, how is that done? Those were all done using robot force control. Force control involves robot sensing and responding to any external forces and moments, which allows for precise manipulation and interaction with the environment.

Now, what is active? What is passive? Passive is the one that I have used with the springs at the end effector or some spring-like structure at the end effector, which can also do some compliance at the end effector beyond the robot flange. That is a spring-like structure fitted between the gripper and the robot flange. So, there is that compliance structure that allows some sort of disturbances or any kind of external force jerks to be taken up by the robotic system. But yes rest of the robot remains stiff. So, that was passive control.

In active control, we take care of live forces from the end effector or sometimes from the joints, also force, and torque and include that in its control loop to do compliant

manipulation. In that case, the whole of the robot structure becomes compliant. So, we'll discuss on active force control in this module only.

The benefits are precision, safety and adaptability. In general, any control structure would look like this. You see, you have some desired position or force commands over here. You have a comparator that keeps on checking the desired value and the actual value that comes from the sensors. Sensors with feedback elements give you that. So, it compares that calculates the error and sends it to the controller, and finally, it goes to the power amplifier and to the actuator and actuators. Actuators are fitted to the manipulator, and finally manipulator gives you the motion. Any disturbances or uncertainties are taken up by the manipulator or by the end effector of the manipulator, and it does precise manipulation okay. So, this is the general structure of any control algorithm. In the case of force control, we restrict the commands to be position and forces only. So this will be the structure of that. So that is what is the robot force control is.



Now, let us discuss what is active force control is with an external force control loop. This was long back proposed by Shutter and Brussel. It was proposed in the IJRR journal, International Journal of Robotics Research, in 1988 it proposed using a Force sensor. So, this is the typical structure of such a system. You have a robot arm. Here, it is shown as a KUKA KR5 Arc robot with a force sensor over here, and you have the gripper. It may be

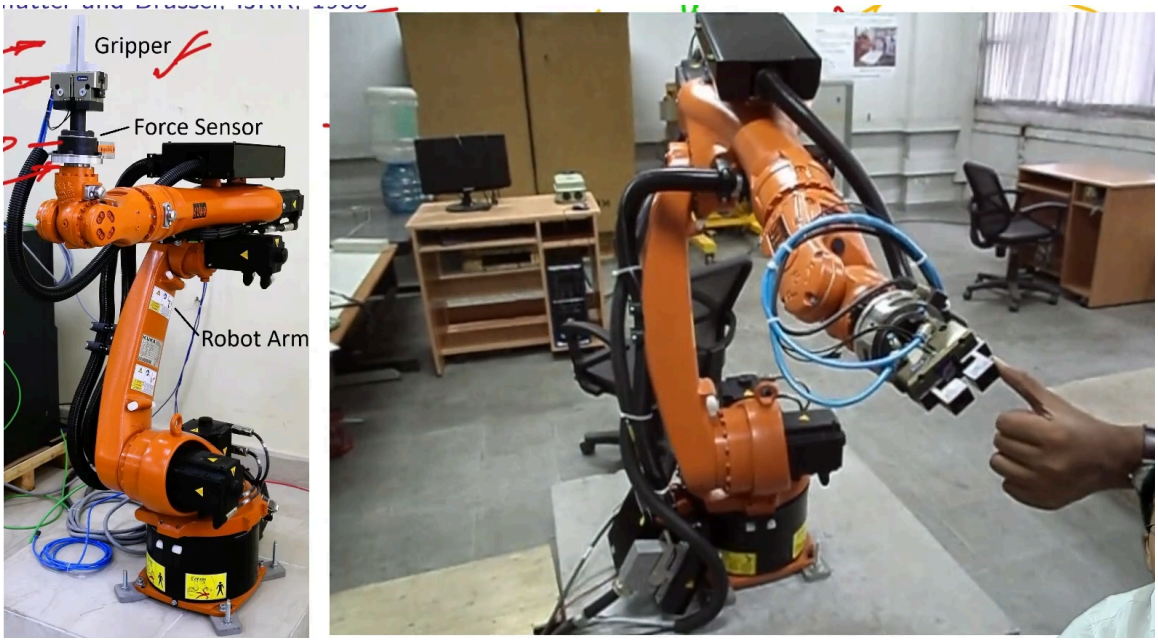
a gripper, it may be a grinding polishing, or any other tool. Over here, it is a gripper. So that is the tool, to be more specific, and you have four sensors in between the robot flange, which is here and the end effector tool. So, this is the fourth sensor. So, it checks the live forces. While it does any job, it checks the live forces, and those forces are now included in the external loop apart from the position control loop, which is already there in such robots which does precise positioning. You bind an external force loop to control the forces at the end effector. So, you see, this is the control structure for that. So, you have the desired force that should be there at the end effector, and it continuously monitors the force through the sensors. Send you feedback here. Calculate the errors sent to the controller here. This is the transfer function for the controller. The controller sets the reference trajectory in terms of the position of the robot arm. So, this is a standard position-controlled robot that takes in the position trajectory here and gives you the end effector position. So, this is a comparator. Basically, this is in the case of a live robot. You can directly get the four sensors which are fitted which are through the four sensors fitted at the end effector, but in case of simulation, you may be required to calculate it using the environment position and the robot position to calculate the error and stiffness of the wall or the obstruction is put here so that ke into x minus xc that is robot position minus environment position. Multiplied by ke gives you the force. So, this is the general equation for interaction, wall interaction or any interaction, and it gives you the force feedback.

So, this is only in case of whole of this is only in case of simulation, but in the real case robot directly gives you the position, and you directly get the force feedback to this loop. So, this is the general structure of such active force control using an external force control loop. So, this is implemented by creating an external force feedback over a position-controlled robot in task space or in tool space. Tool space: The tool has its own frame. In that frame, it may be achieved, or in the robot's Cartesian space, it can be achieved.

Any existing position-controlled robot can be retrofitted. This is the beauty of such compliant manipulation algorithms: you can just have four sensors fitted on the existing robots in the assembly line or in any application, and you can convert them to do such

compliant manipulation tasks. It gives you improved safety, precise assembly tasks can be done, and adaptability to any uncertainty that may be there through robot vision or through robot accuracy. So, it can take up those uncertainties. Enhanced interaction can be more interactive, cooperative manipulation, surface finishing or any other collaborative manipulation tasks.

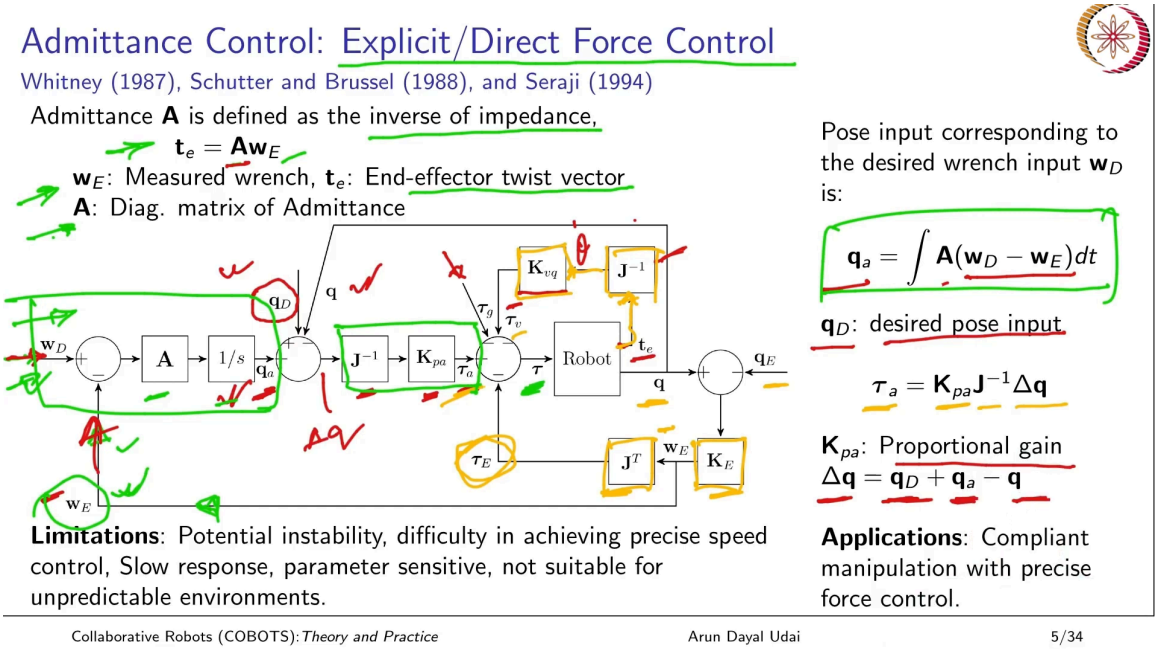
The only problem here is that the collision with the links remains undetected. The link collision cannot be detected because there are no sensors available that are in the rest of the robot. So, whatever happens, it happens beyond the force sensor, and those are sensed by the robot, and that can be considered in your closed loop. Apart from that, they are slower in response. That means that because this robot has an external loop and the forces, the error is considered by the robot's controller using this controller signal. So, this controller essentially remains outside the robot controller, okay. You have position control, which is inbuilt into the robot, but this controller force controller remains outside the robot. So, there is a continuous communication that is happening here, and that communication and doing this closed loop has some latency there. So that is the reason such control is very much slow. It is not very much in real-time.



So, let us just see a small video on that. So, this is what I am doing. I have just fitted the

end effector. This force-torque sensor is at the end, and it is checking for forces from all three directions.

I am able to move the wheel. I have demonstrated in 10 courses in all three Cartesian directions. So, anything greater than 10 meters will start making the robot move in one direction, corresponding to the direction. That is what is happening here. I am able to drag this robot anywhere I want. So, this is an external force control loop.



So now, Admittance Control. This again is an Explicit or Direct Force Control where you directly command the forces or, in general, the wrench. Wrench is a 6 cross 1 matrix which essentially contains the moments and the forces at the end effector. So, that is there. So, in general, we talk about moments and forces, but normally we use forces, okay. So, admittance is defined as the inverse of impedance.

So, the twist is equal to the admittance times of the wrench, that is, the measured wrench at the end effector. So, you have measured the wrench given by w_E. te is the end effector twist. Twist again is the 6 cross 1 vector of angular velocity and displacements. A is the diagonal matrix of admittance.

So, when you see twist is equal to A times of w_E, integrating the twist should give you the positions because those were the velocities.

$$t_e = Aw_E$$

So, integrating that gives you the position. So, the admittance times of the desired wrench minus the actual wrench. So again, this is a closed loop that takes in the rest desire and the feedback input that is the wrench's actual measured range takes the difference to calculate the wrench error. Wrench error into multiplied by the admittance gives you the twist vector, and integrating that gives me the position trajectory.

$$q_a = \int A(w_D - w_E)dt$$

So, this position trajectory, after integrating here, is sent to the robot controller as a reference trajectory. This (qD) is the second input. The first input is the wrench input. This could be the second input, which also takes in the position trajectory here, and this (q) is the actual position. So, the final desired pose input. The final desired pose input is q_D.

q_D minus $q_{a,}$ that is, the actual position and the reference trajectory input. Delta q that comes here. This is Delta q. What is Delta q?

$$\Delta q = q_D + q_a - q$$

Basically, it is the difference between the q desired that is the trajectory that is the desired trajectory. q_a, this is the reference trajectory, which is set by the admittance controller here, and q is the actual feedback.

So, if you take the difference, you get a change in pose, which should happen due to the admittance controller and the trajectory inputs. This is sent to the Jacobian inverse. Finally, it gets converted to the joint space multiplied by proportional gain, which here gives you the torque trajectory. So, tau a is the torque that goes to the robot controller once again, and again over here, you have tau g. This essentially is the gravitational compensation torque, which makes this robot free from the gravitational pull, and the robot becomes floating in space, as we have already discussed in the statics module that

you have seen. So, Tau g essentially makes the robot floating in the space. So, that controller becomes free of gravity. So that is there.

Now, you have this is the twist which is sensed by the sensed or calculated by the robot's forward kinematics. So, that is the end effector twist. Twist multiplied by Jacobian inverse gives me the theta dot times, and theta dot that is the joint rates multiplied by the viscous friction, which is there at the joint, should give me the torque due to the viscosity or due to the damping, which is there at the joints.

So, this is what is realised only in the case of simulation. So, this loop is there for simulation only. In the case of a real robot, you don't need to calculate the viscous or damping torque that is automatically there within the robot. So, tau v is there in the case of simulation.

Similarly, it is calculating the torque due to the forces that are felt at the end effector. So, robot position minus environment position multiplied by the stiffness of the environment gives you the wrench due to the environment interaction. So, this wrench multiplied by the Jacobian transpose gives you the torque due to any environmental interaction.

This again is a direct case. In the case of a robot, a physical robot, you need not calculate. So, all the yellow blocks this one this one. So, this is only there in the case of simulation. In the case of a real robot, you need not worry about it.

So, this is how this admittance controller would work. So, all the elements I hope it is clear to you.

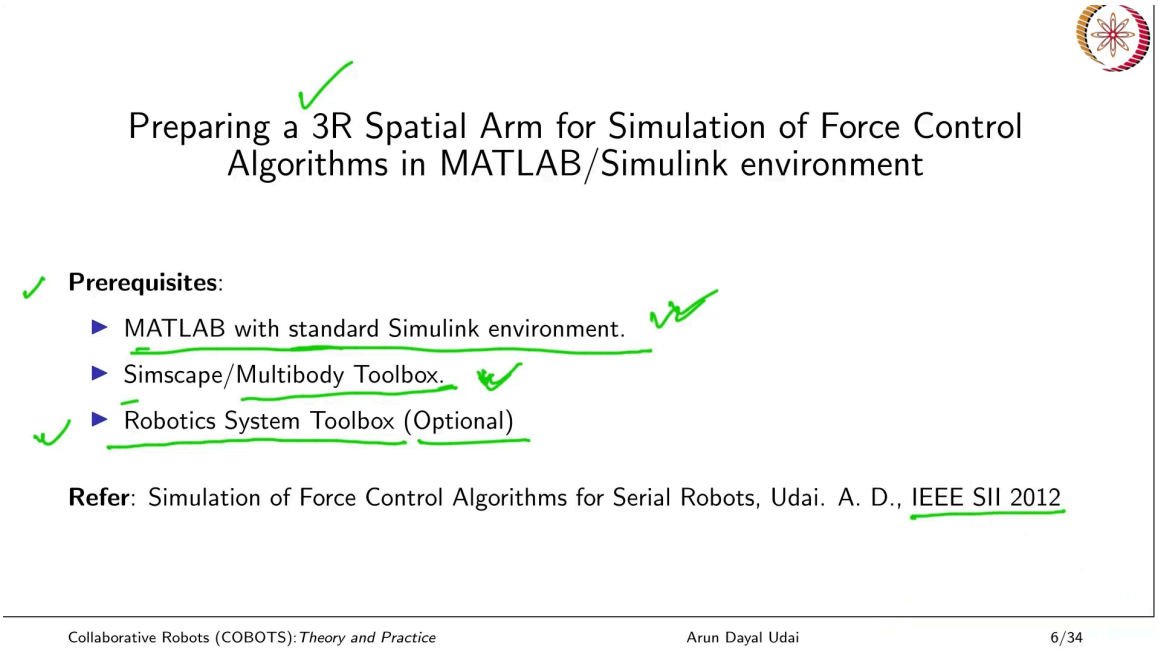
$$\tau_a = k_{pa} J^{-1} \Delta q$$

So, this is tau a. Tau a is sitting here. Tau a is kpa, which is the proportional gain times of delta q Jacobian inverse. Delta q Jacobian inverse it is here. So, this equation sits here, and this one is from here to here. w_D minus w_E an integrated.

$$q_a = \int A(w_D - w_E)dt$$

So, it is here. So, this is all. It is a dual loop again. So, you have $w_{E,}$ which is set as feedback, which is sensed through the external force torque sensor. So again, you are depending on an external force torque sensor. However, the robot is torque-controlled. So, in this case robot is also compliant because it is sensing the end-effector force torque using this, and also, the joints are compliant because of torque torque-driven robot.

Torque is exactly whatever is required. So, the robot becomes compliant with any external force that comes on the link this. There is a problem: you also need an external force torque sensor. The beauty here is that it is a direct force control technique. So, you can directly and precisely put the force commands, and it exactly achieves the precise force at the end effector. It is very beneficial for applications that require precise force control. So, compliant manipulation with precise force control.

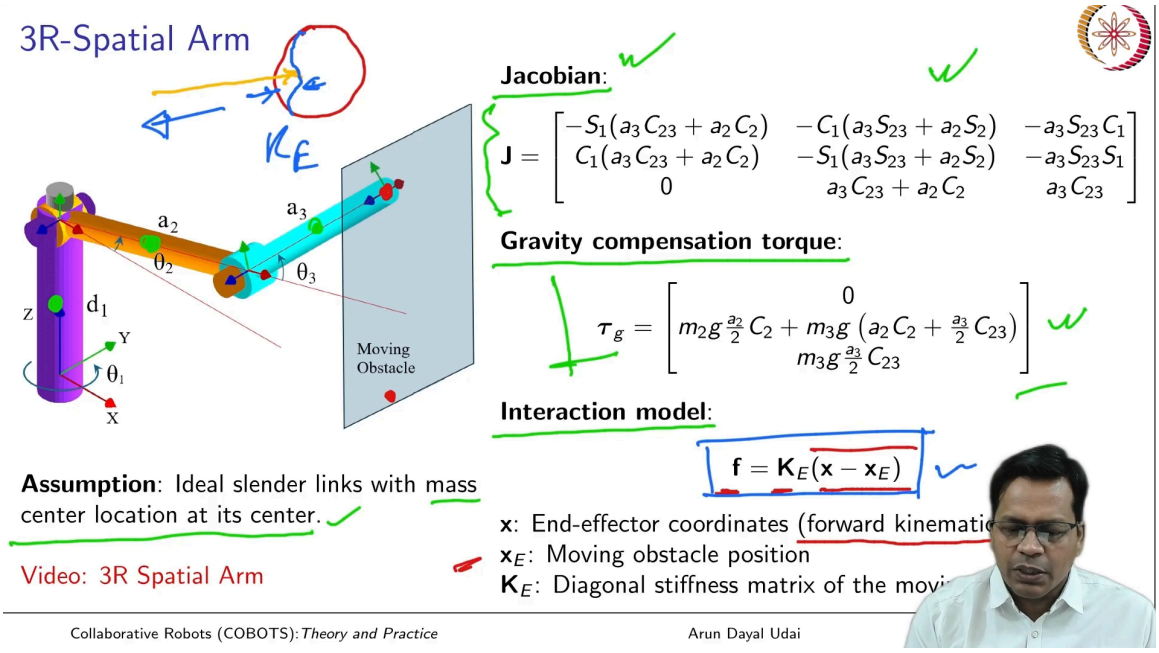


Let us start preparing the 3R Spatial Arm for the Simulation of Force Control Algorithms in the MATLAB Simulink environment. Why I have chosen the 3R Spatial Arm here is because most industrial robots or cobots also have a 3R Spatial Arm right up to their wrist, and the Spatial Arm can attain any position in Cartesian space in x, y, z location. So that is the reason I have chosen this, and to make things very, very simple, I have chosen just a three-degree-of-freedom system which can move along x, move along y and

move along z, that is in all three directions. So I can control my forces In all three directions. I am not talking about the moments here to make things very, very simple.

The prerequisites here would be MATLAB with any standard Simulink environment. Any version of MATLAB with a standard Simulink environment. I would require a Simscape Multibody Toolbox to create my robot and do my closed-loop operations. Additionally, you can have this is an optional Robot System Toolbox, which can take care of gravity torque, or it can calculate the robot Jacobian and all. So that becomes very trivial in the case of a robot system toolbox, but that can also be done programmatically when you are using a multi-body toolbox.

So, you may refer to this paper that I have cited here, which is the Simulation of Force Control Algorithms for Serial Robots by me (Arun Dayal Udai), which is an IEEE SII 2012 conference publication that you can go through.



So, let us have some background on the 3R-Spatial Arm that you already know. I will directly show you this video, the kind of motion that it can do. I am moving joint 1 here. Okay, this is joint 1 motion.

So, the first axis is perpendicular to the ground and it is along the Z-axis. The XY plane is the ground plane here. Joint 1 is done. This is joint 2, which moves links 2 and 3

simultaneously. This plane of motion is perpendicular to the plane of the ground, which is the surface on which this robot is mounted, and the third joint is also like that. This moves link 3 only along with it, and this is also in a plane that is perpendicular to the ground. Got it? So, this is how it is going to move.

So, I assume the links are all slender. It is an ideal slender link with the mass centre located at its centre. All the centres of gravity are precisely located at their centres. So, the Jacobian can be calculated. You already know how to calculate the Jacobian. In the same way, you can arrive at this result using the techniques that were discussed in the robot Jacobian and velocity kinematics.

$$\mathbf{J} = \begin{bmatrix} -S_1(a_3C_{23} + a_2C_2) & -C_1(a_3S_{23} + a_2S_2) & -a_3S_{23}C_1 \\ C_1(a_3C_{23} + a_2C_2) & -S_1(a_3S_{23} + a_2S_2) & -a_3S_{23}S_1 \\ 0 & a_3C_{23} + a_2C_2 & a_3C_{23} \end{bmatrix}$$

Gravity compensation can be calculated as tau g like this. That was also discussed in the robot statics module.

$$\tau_g = \begin{bmatrix} 0 \\ m_2g\frac{a_2}{2}C_2 + m_3g\left(a_2C_2 + \frac{a_3}{2}C_{23}\right) \\ m_3g\frac{a_3}{2}C_{23} \end{bmatrix}$$

So, these two I have directly put here. I'll be using these to calculate the gravity compensation torque and the robot Jacobian directly when I do these calculations in MATLAB code.

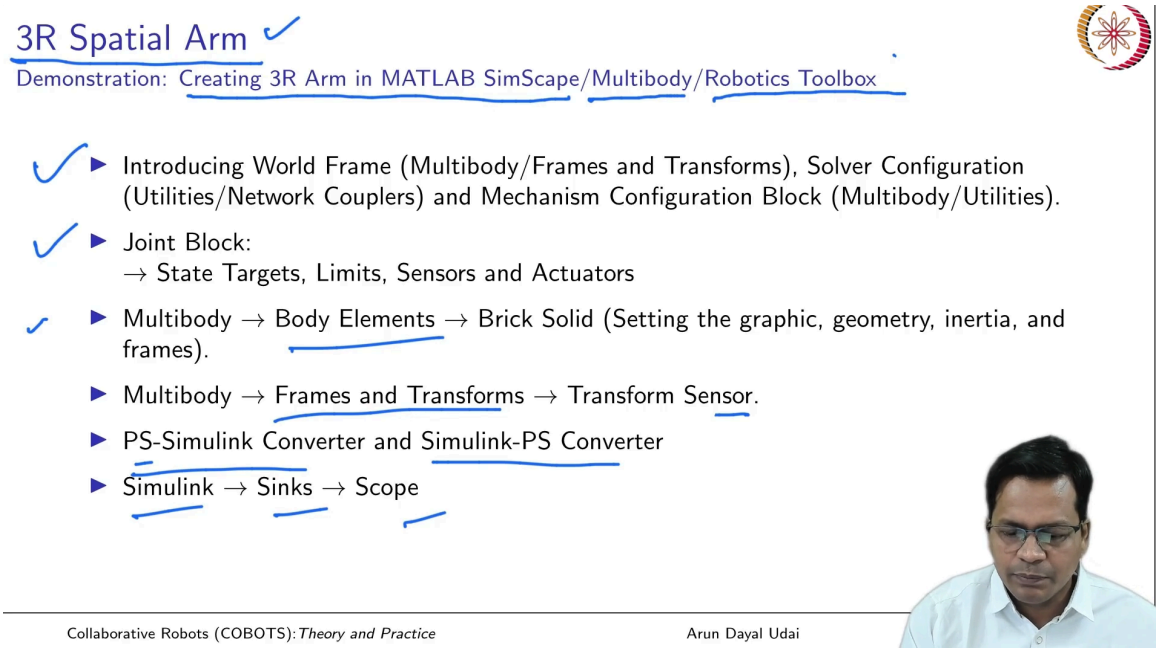
The interaction model I have taken is like this:

$$f = K_E(x - x_E)$$

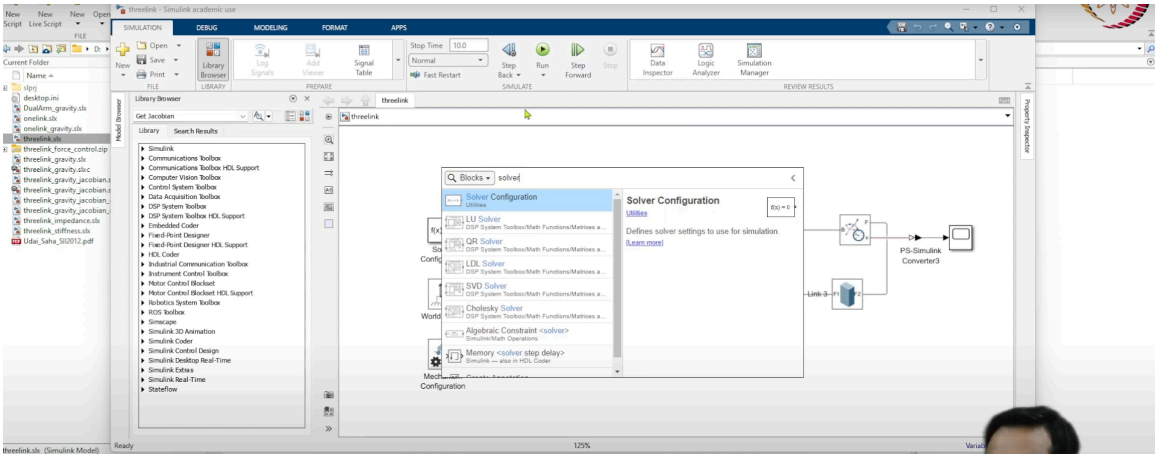
The robot's end-effector position is here, and this moving obstacle or obstacle is somewhere here. So, whatever is the robot's movement into the obstacle? So, whatever penetration happens into this obstacle is calculated using x minus xE. x is the end effector coordinates that are calculated using robot forward kinematics, and xE is the position of this moving arc obstacle. So, the difference between them should give me the amount of

penetration of the robot's end effector that has happened into this obstacle. So, that is actually the compression that is happening to this obstacle. That compression multiplied by the stiffness of the obstacle gives me the forces. So, if this is an obstacle due to any robot movement into that, you get some deformation here. So, this deformation into the stiffness gives me the force that is generated, and this force is normally along the direction that is perpendicular to the direction of this compression. So, whatever the deformation is, it is perpendicular to that plane.

So, that is how it is assumed. I have not assumed any frictional forces that may also be generated due to the movement of the robot along the plane of this obstacle, which I have ignored, but yes, normal forces are generated due to the compression of the obstacle. So, this is what the interaction model is. So, I hope you have understood it.

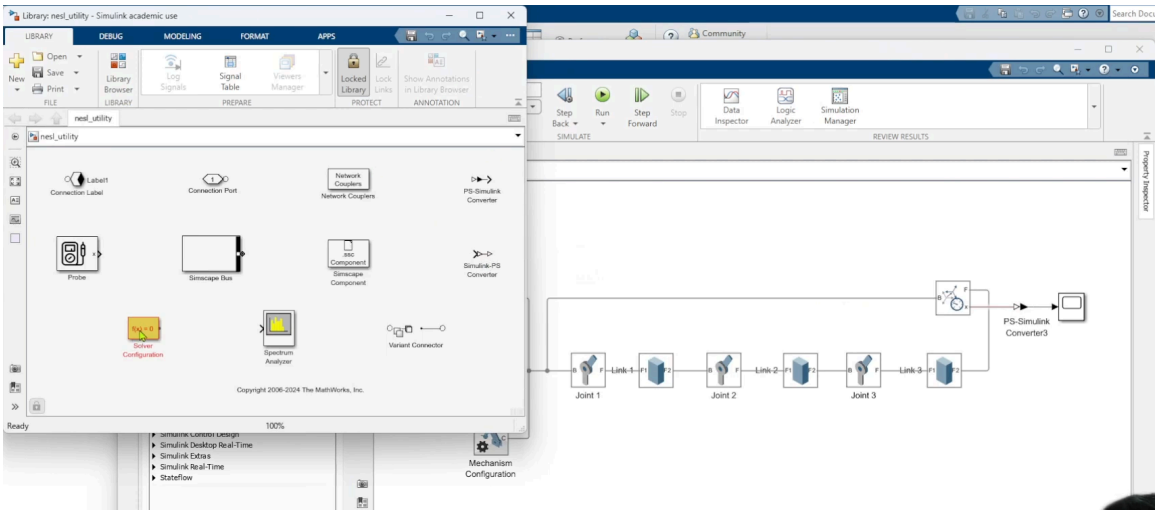


Now, I'll create a 3R Spatial Arm. Let us see, the demonstration of creating the 3R arm in MATLAB Simscape with the Multibody toolbox, also using the Robotics toolbox, which is optional.

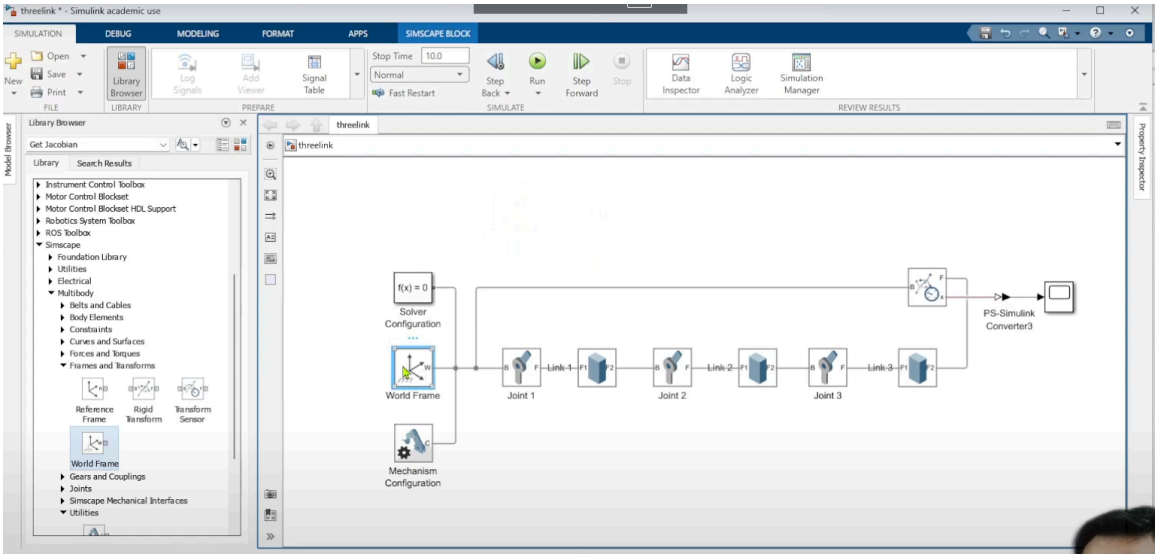


Let us switch to MATLAB now. So, this is my MATLAB environment. I can get into Simulink directly by pressing it here, or by double-clicking the model that I have generated. So, I'll do that now. So, this is my MATLAB Simulink environment. So, you have a library browser where all the tools are there that we have discussed earlier in one of the modules.

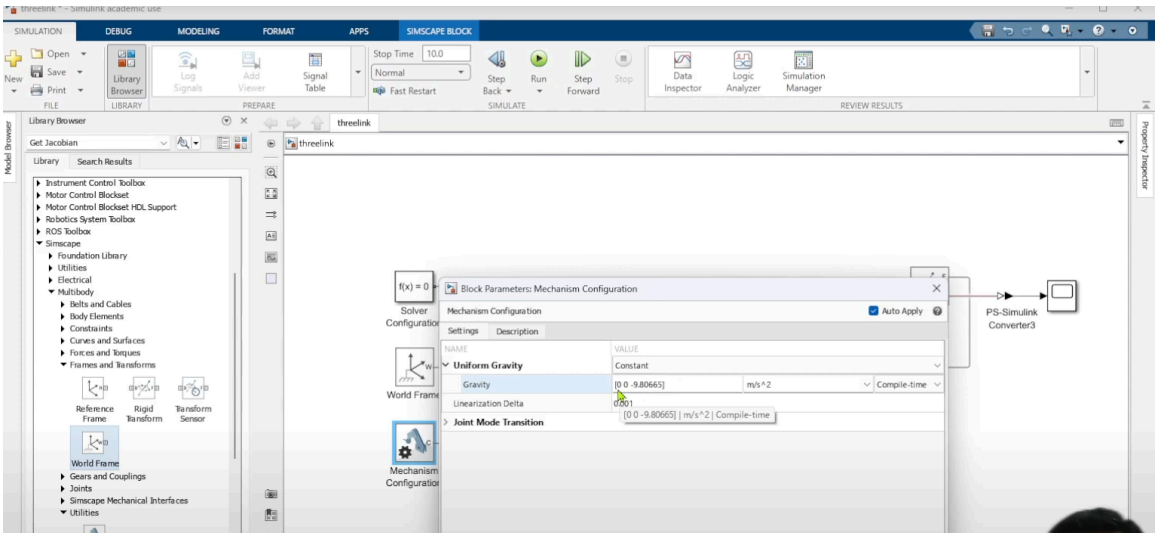
So, this is the 3R Arm that I have built earlier to save some time. So, I will directly touch upon the various blocks that I have used here. I have used the solver configuration. You can insert this directly here. Double plating on the space type solver configuration.



So, you can directly get that in the utilities here. You can take that from the utilities here. From utilities, you can directly drag and drop. There, you get the robot solver configuration. So, this is the block that is there in the utilities.

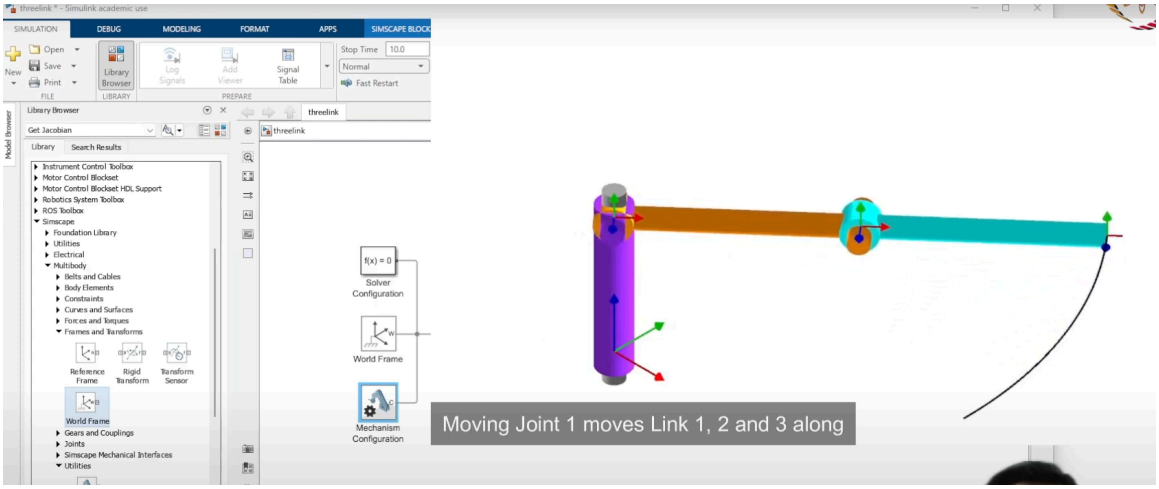


Again, the world frame I have inserted here, you can directly go to the Simscape environment multi-body and utilities. You get a mechanism configuration that I have also used here. So, this is my mechanism configuration. Frames and transforms, here you get the world frame. So, these are the blocks that I have used. So, the world frame also can be used directly from here. World frame o, you can directly click here, and you can you may get that inserted here. This is already here. So, I am deleting that.



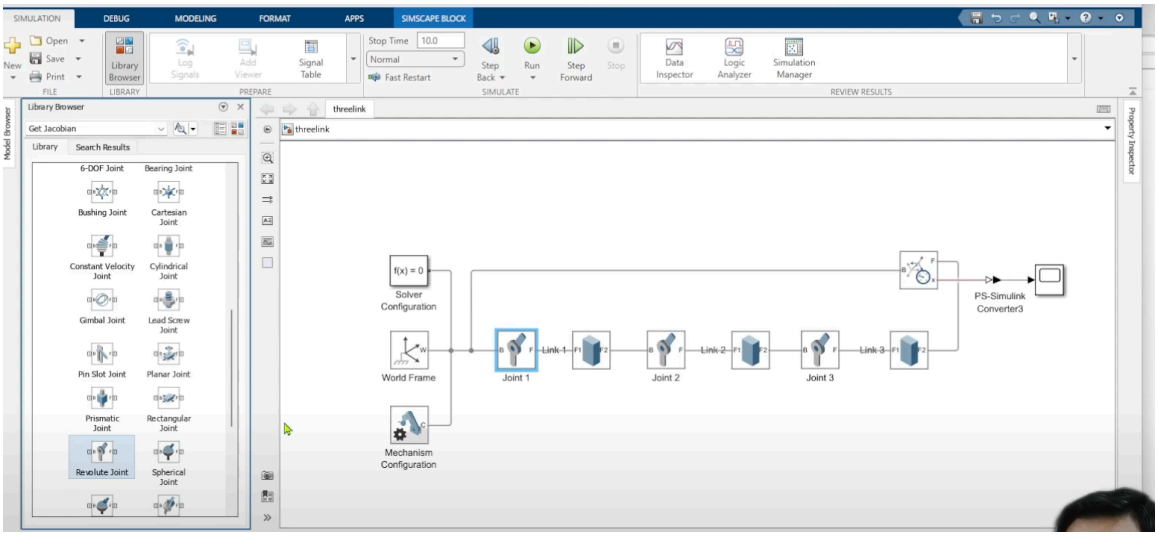
So, this is your world frame. This actually says the 000 position, which is the reference position for the whole of the robot representation. This is the solver configuration. This is a mechanism configuration. This also says the direction of acceleration due to gravity. So, now I have 0, 0 minus 9.8. So, that means 0 along x, 0 along y and along z, that is

perpendicular to the plane of the surface. You have acceleration due to gravity that is directed downward, as I have shown in the 3R manipulator also.

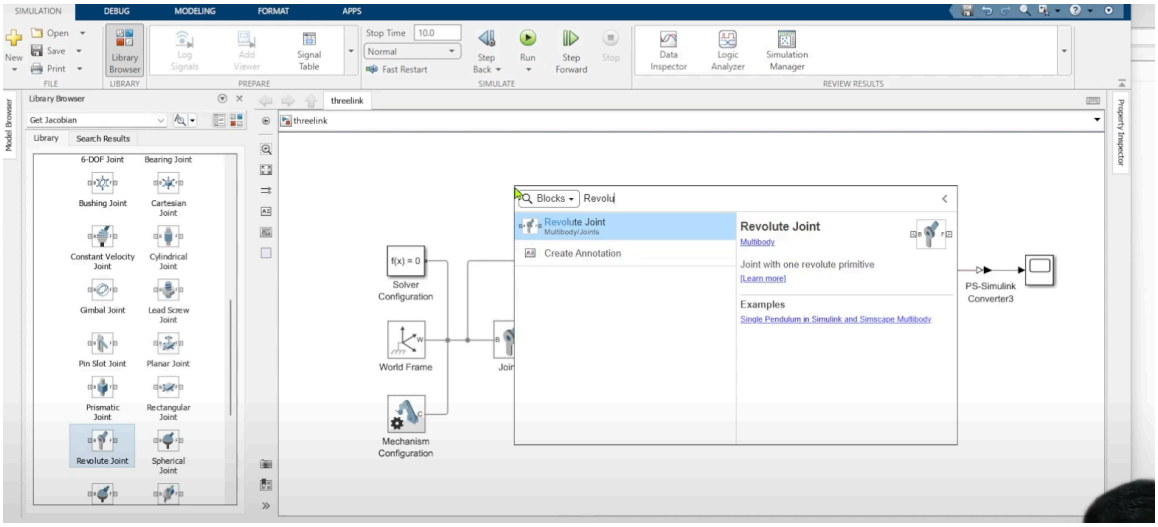


So, acceleration due to gravity becomes along the minus g direction. So, that is there here once again. So, this is my, this becomes the skeleton for any model that I will be making. So, these three blocks are always there.

Now, I start with the first joint that will mount link 1 on top of that, the way it is here again. So, you have the first link, which is the violet colour one, which is mounted on top of frame zero, which is the ground frame through joint 1. So, here is your joint. 1 joint can be found here. So, you have all the joints. So, I have dragged in the revolute joint.



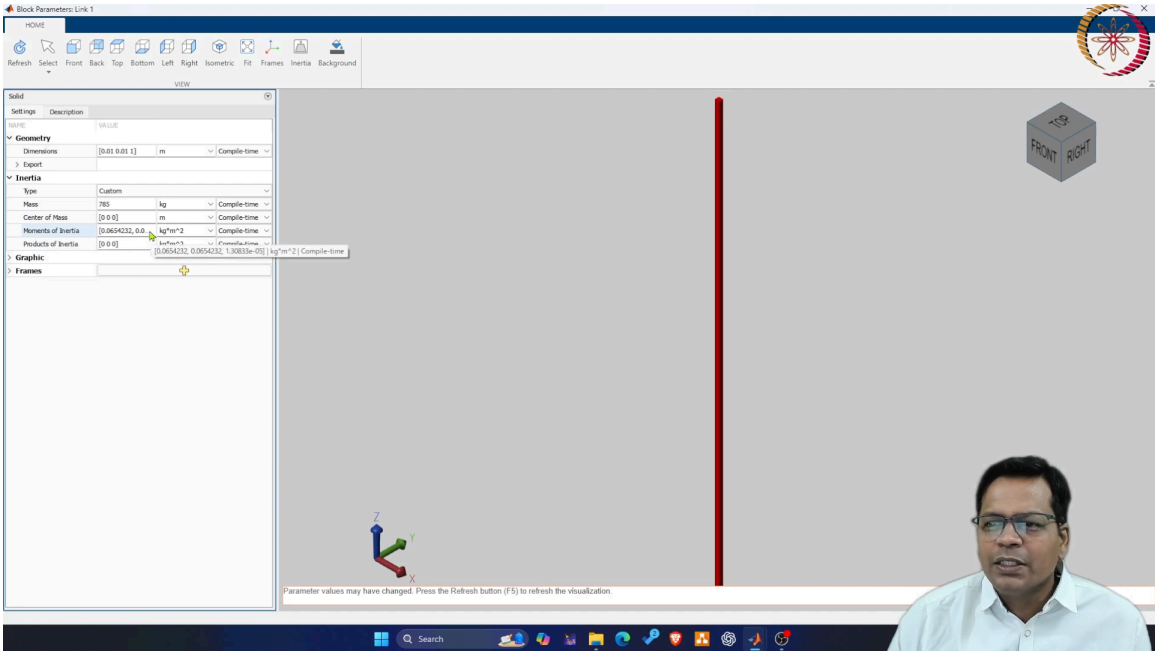
This is the revolute joint. This revolute joint I have used here. You can directly click here and type.



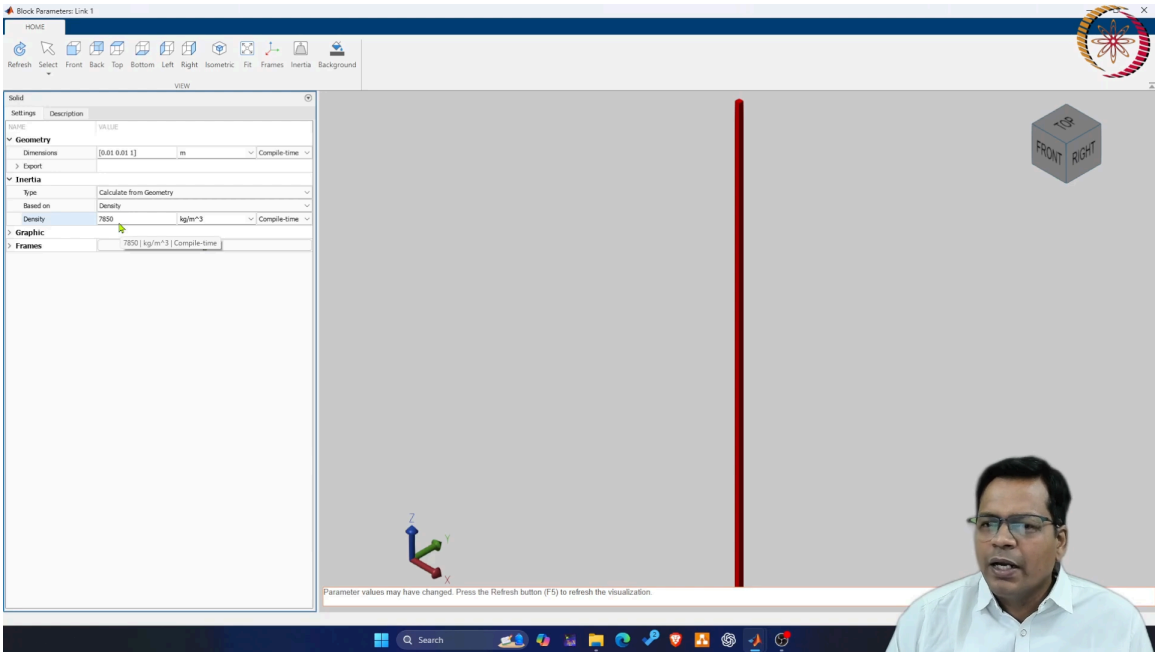
So now, you know you can directly type it here. If you know the block name, or you can directly select from here. You can choose one of them, and you can directly drag and drop and you can get to that joint. So, this is joint 1. So, I have inserted it here.

Now, joint 1 has B and F. You can read it. This is B, and this is F. B means the base. F means the follower. So, the base is the joint. The base is connected to the world frame. The follower will contain my link. So, this link can have various body elements. For body elements here, I have chosen a brick solid. It can be a cylindrical solid or anything. So, this is my brick solid element, which is here.

So, now I have dragged it, but I also have to model it. I have to give it some length. So, double-click on it to select its property. Dimensions are 0.01. If you see 0.01, that is along x. So, this is my link. So, along x, it is 0.01. So, it is I want to make it very much slender. So, this is the reason I have taken very negligible dimensions along x. Similarly, along y along z, that is, along this direction. Length of the link: I have kept it as 1 meter. So, the total length of this is 1 meter.



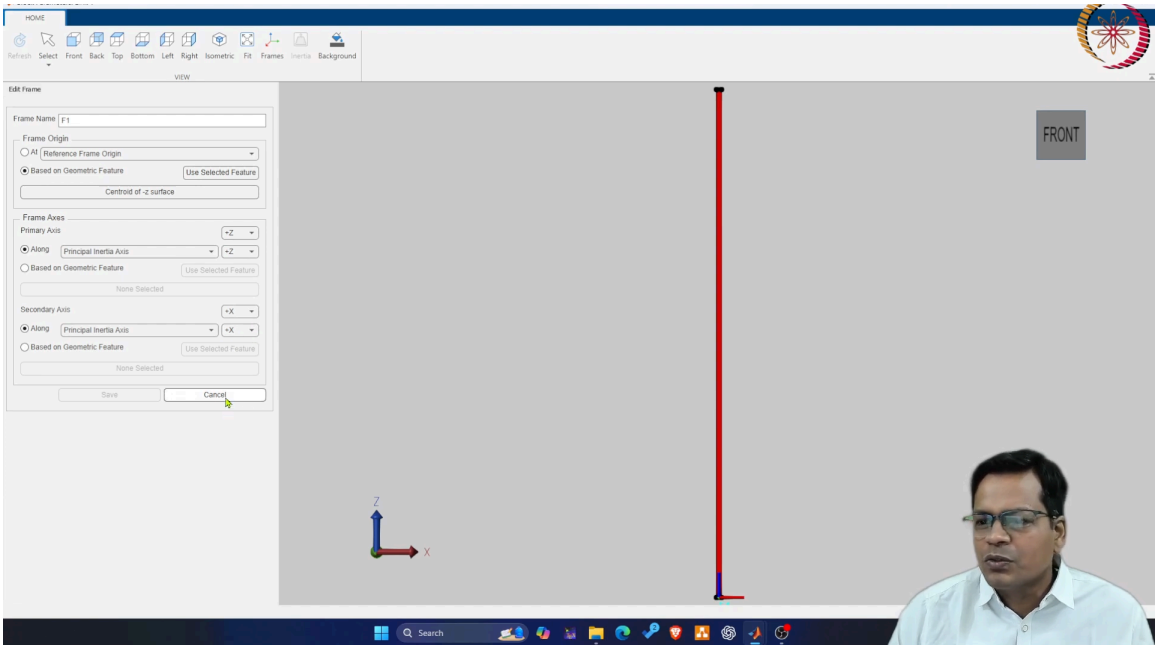
Now, the inertia property I have chosen is calculated from geometry. Density I have chosen, or you can directly key in the custom mass or the point mass, you can directly key in here, okay? Again, the density it will be calculated based on that. So, this is very much useful to calculate the moment of inertia and other factors that are there. So, if you say it is a custom one, then you are asked to insert the mass of the link, the centre of mass location moment of inertia, that is, Ixx, Iyy and Izz. Those are and product moment of inertia can be inserted as well.



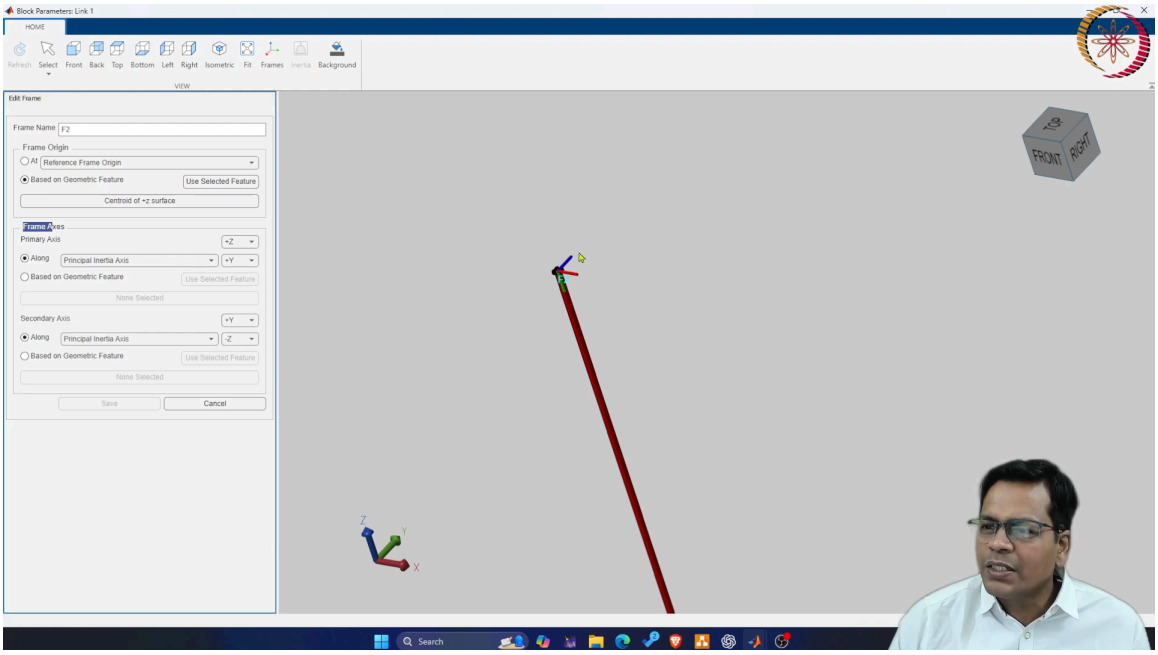
So, this is using a custom way but I have kept it as default by using geometry here as default, and I have just put the density. So, it will automatically calculate those parameters and the total mass. The mass centre will assume it is evenly distributed mass, and that will be there. Graphic location from geometry. Visual properties you can choose. I have chosen red colour here. Opacity is one. It can be opaque. It can be transparent, semi-transparent or something like that, but that I haven't touched upon just using the colour here. There are two frames. One of them connects this to the link to the joint that is over here that is at the bottom of this link, which is frame one. So, that is chosen like this. So, this is your space where you can select the link. So, again, now you can do pan zoom something like this. Using the shift button and mouse centre button, I am doing the panning operation, or you can do the scroll button to just zoom in and zoom out, okay. So, panning, zooming in and zooming out. Select to certain standard views from here. From the toolbox, you can do completely fit different isometric views can be done. This is how you can manipulate your object which is there in this workspace. From here, you can directly select the front view, isometric, top view, left view and something like that. So this is how I am doing it.

So now, you see, along the length of this link, I have a Z axis. Because I want to rotate this link about its own z-axis. So, the joint will come here the joint will connect this. So,

it is along the z-axis, and the other two axes are here. So, y is aligned along y. x is aligned along x. z like this. So, this is how it is okay.

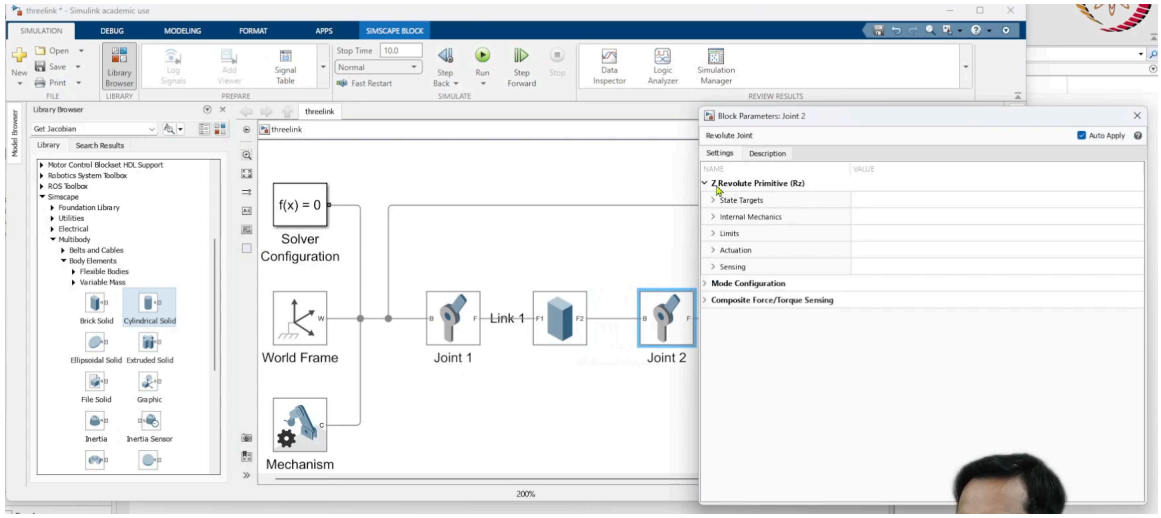


Now, frame two. So, yes, again in frame 1 also, you see Z is primary, Z is aligned along principal axis Z. Secondary axis is if you select two of the axis, the third one gets calculated automatically. So, if you select the X axis along plus axis. So, this is the default I have not touched upon. So, that is that became my frame 1.

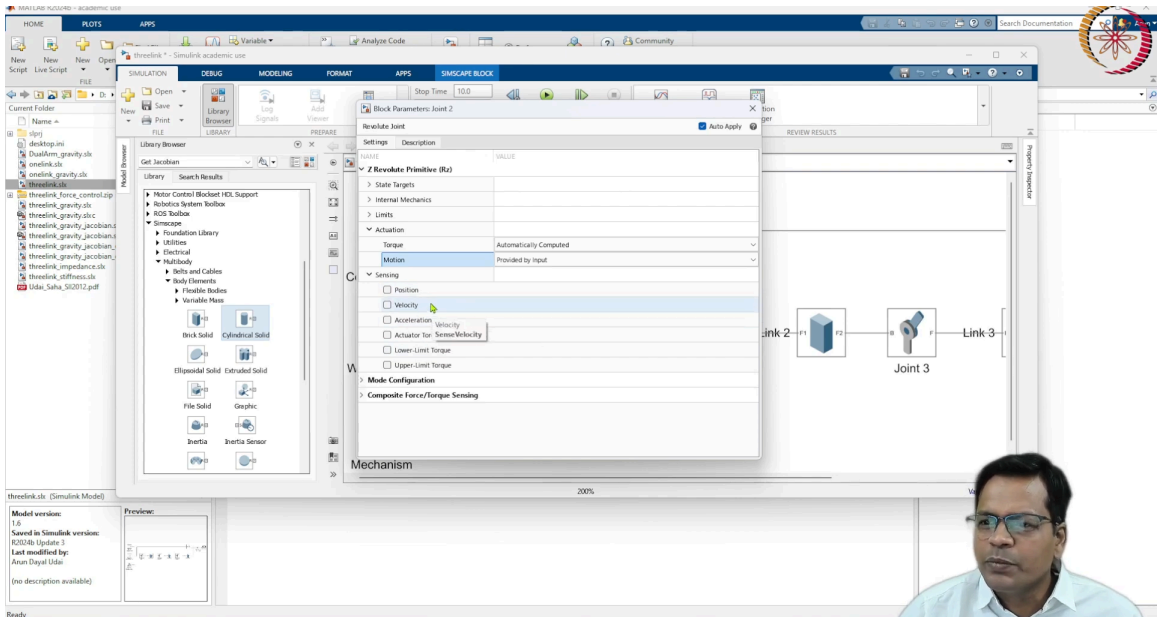


So, now going to frame 2. This is how I want my frame 2 to be. Why do you know? Because the next link is there is like this, you see. As per the DH parameter z-axis should be like this. So, I have to rotate this z-axis. It cannot remain along aligned with the principal axes which are here okay. You have to rotate it because you want your DH parameter to be set like this. So, all the rotation of this revolute joint rotates about the Z axis. So, I have to put it like that. So, this is the reason the Z axis is like this.

So, now I will rotate the frame, which is by default, which was there. So, by default, it is aligned with the principal axes, which are here. So, I have to rotate it like the principal axis was along z. So, that will now be along plus y. So, you see, my z-axis principal axis is along the y-axis. Again, the second one secondary axis, is the principal axis y principal axis. y you saw, it is the Z axis. Now, it is the secondary axis plus y is along minus z. So, plus y, this green colour, which you can see here, is along minus z. So, now two axes are defined third one is the x-axis, which comes directly along the x-axis. So, this is how it is okay. You know you have to have the z-axis along the axis of rotation of the revolute joint. So, that is the reason based on DH, also, it has to be like this. I have rotated it like this. So, I have put the second frame here. So, that is my link 1.

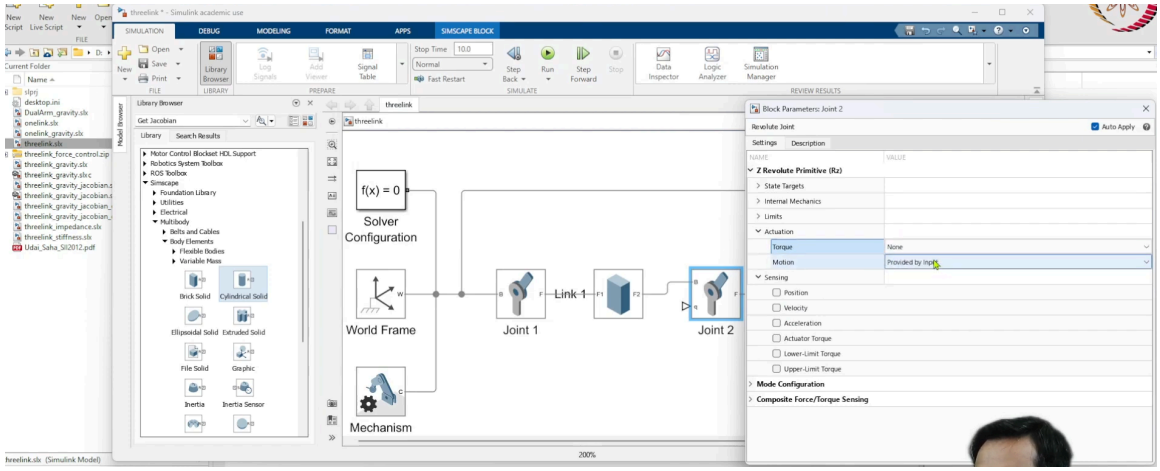


Now, I will put again joint 2. Joint 2 is here. Any joint you see has state targets. State targets mean the value specifies the position of the initial position of this joint. So, that will come here. Limits, if any, you can specify the lower and upper limits of the joint rotation. Actuation can be torque; it can be provided by the input, and motion gets automatically calculated. So, torque is provided, and motion is calculated automatically. That means MATLAB does the forward dynamics here.

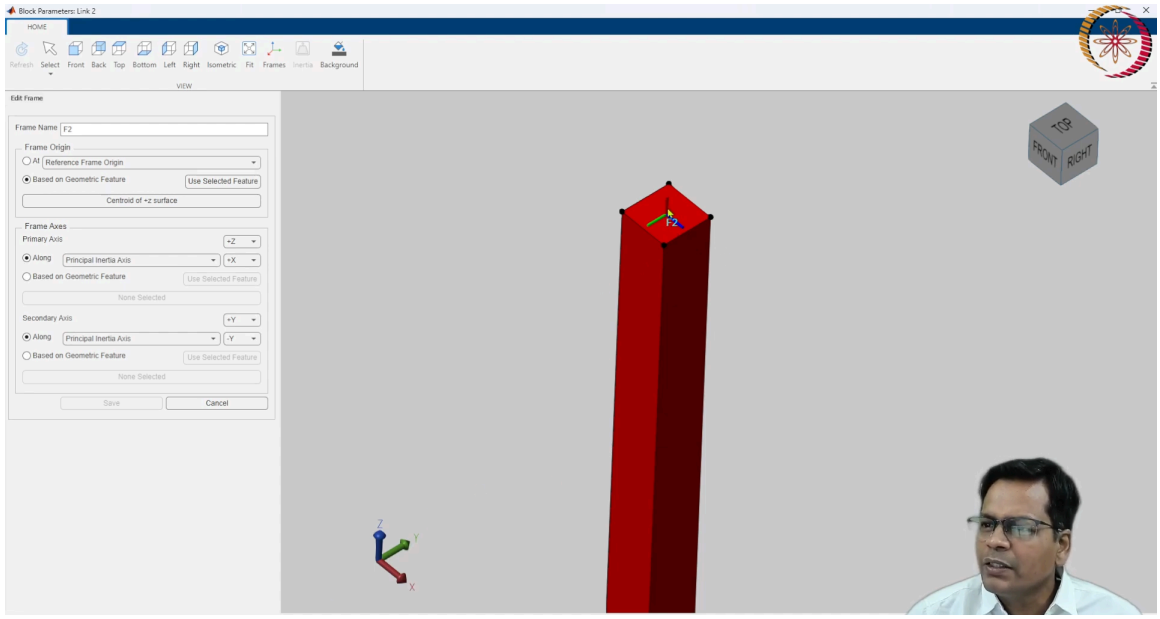


So, you see, torque provided by the input, motion can be calculated, or it can be otherwise: motion provided by the input, torque getting calculated automatically. So, it is inverse then. So, forward or inverse, you can do it here.

Sensing, you can sense the joint position, joint velocity, acceleration, actuator torque, lower limit, upper limit, and torque. So, these are the senses which you can have. Normally, we would require the position and angular velocity of the joint, which will need to be sensed to calculate the Jacobian to calculate the joint angular position. We may also require actuator torque. So, these things we can do to the joint.



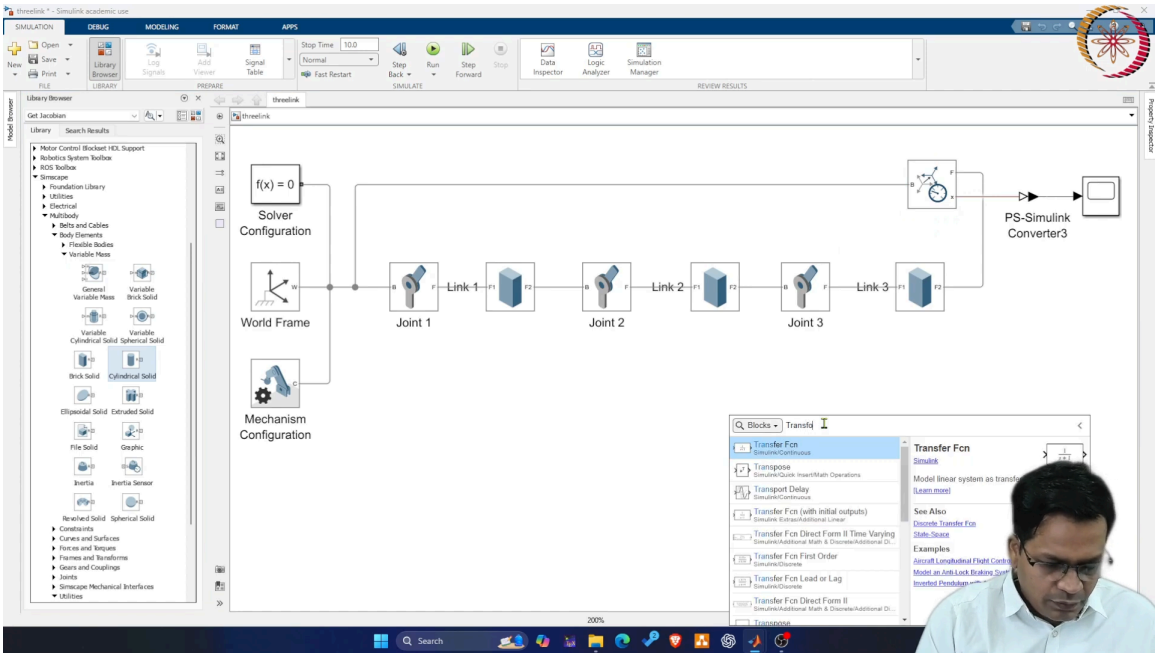
Now, let us calculate the torque. I will provide none. The motion is getting automatically computed. So, this is the default setting I have put here because this is the initial arm, Joint, and then link 2. Link 2, we have built here.



So, again, you have that was the link 1 and now gets connected to the link 2. So, over here again, you have to choose. So, inertia properties I have set it, left it as it is.

Dimensions again, I have kept it as 1 meter along x along y. It has negligible dimensions. Frame 1 is something like this, you see. In frame 1, you have the axis of rotation that is z is along this direction and in frame 2 again, that becomes my axis of rotation that is again along the z-axis because over here, the axes are perpendicular to the link lengths that are for the link 2 and final link that is link 3. both would be very very similar. So, for both of them, I have just kept it as it is.

So, you see, z is directed along positive x. y along negative y and x along positive z. So, this is the setting for this configuration that I have kept okay. So, this is my link.

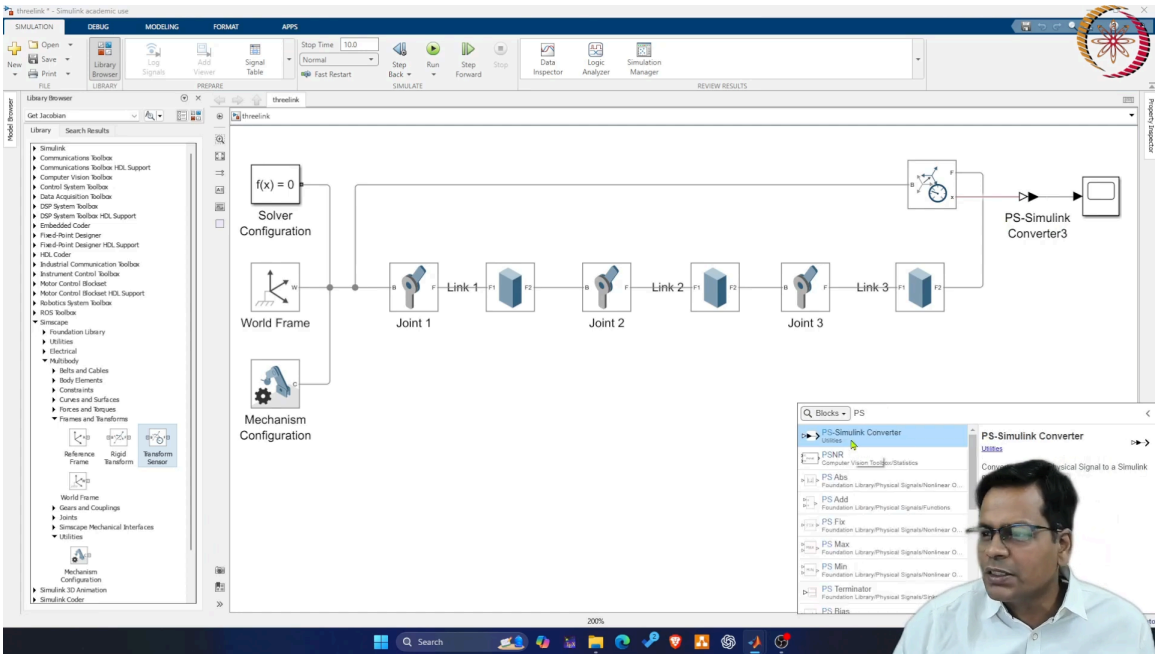


The third link is the exact replica of the second link. So again, you can directly copy or paste it, or you can recreate it, is up to you. Now, everything comes in a series way. It is a serial chain robot. So, it is like this. So, you have solver configuration, world frame, and mechanism configuration. Here comes your joint one. With the perpendicular to the surface, there is a link. You haven't edited frame one. Just frame two needs to be edited, which has to be kept parallel to the ground surface axis of rotation. Joint 2 is here, and link 2, link 2, axis Z, and axis Z should be along the axis of rotation. Then comes joint 3; link 3 is exactly similar to link 2, and this is your frame 2, which is the end effector frame here.

Now, there is a block which is known as a transform sensor. That also can be introduced from here directly, okay. So, if you want, you can directly click here and say transform sensor.

So, that is there in the multi-body. Again, okay. Frames and transform. So, you can go here directly. Multi-body frames and transforms. So, here is your Transform sensor. This basically does the forward kinematics and lets you know all the end effector transformations with respect to the base transformation. So, over here, I have selected the base to be the 000 frame, the first frame, the ground frame and the end effector frame; I have selected the tip of the last link, that is, frame 2, as the follower, base, and follower. So, in between them, it will calculate the transformation.

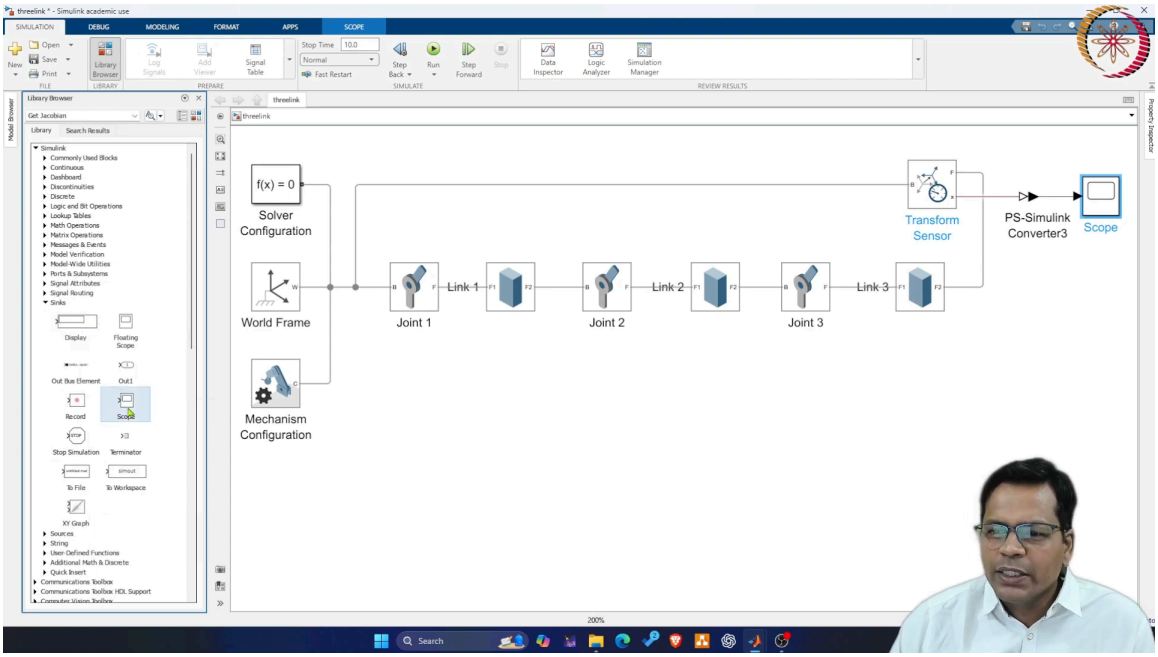
So, you can get the angular velocity of the end effector, angular acceleration, translation along x, translation along y, and translation along z. So, you can get all the forward kinematics computation using a transform sensor. So, it gives you that.



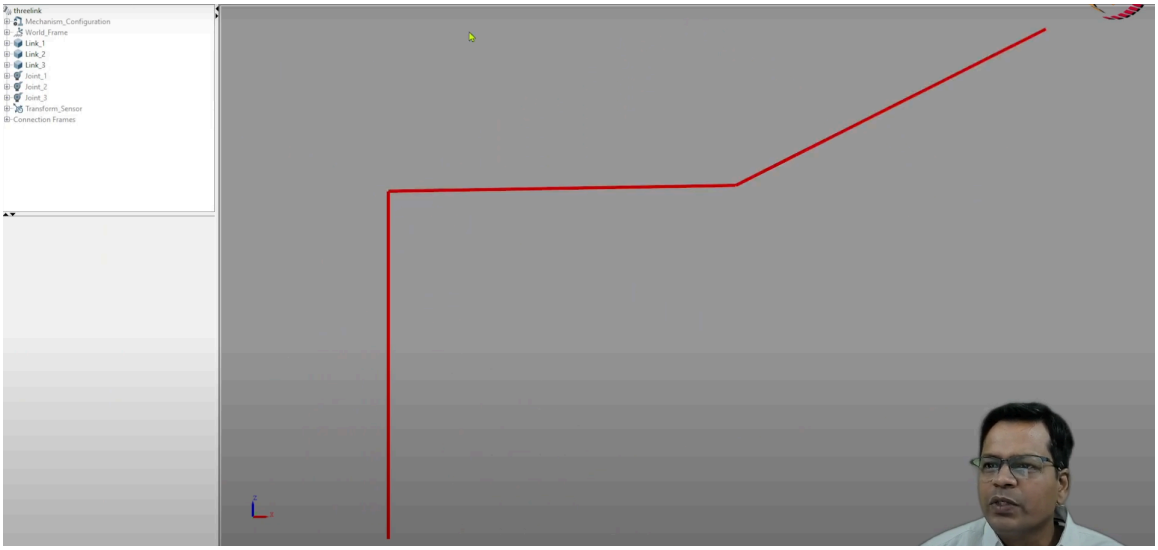
So now, I am just choosing to sense the translation along X. Always remember, all the systems here use a data format which is of physical system data format, and by default, Simulink uses a different data format. So, you have to convert the data. So, there is a converter called Simulink to PS converter and a PS to Simulink converter. You can

directly insert it from here. PS to Simulink converter is there in the utilities. So here you get it: Simulink to PS and PS to Simulink block. These are there.

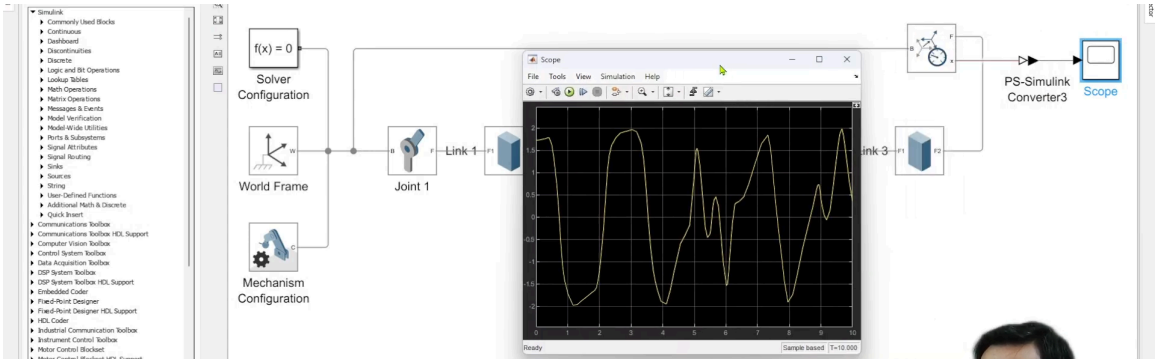
Using that, you can convert that so as to see the plot. So the plot is there. This is the scope where you can see the plot of x while the robot moves. The x-coordinate of the tip where you have put the sensor here.



So Simulink in the sinks so everything that is terminating, storing the data, displaying it, plotting that, so everything comes here. So these are the sinks, and there are sources also, if you want to generate any signal. These are the sources. So these are the sinks that I have introduced here. So that is a scope. I want to see the plot because this is a general Simulink block, so you need to convert the data from the physical system to Simulink.



So now, I am ready with the three-link system, and I can simulate my robot. So you can directly run it from here. So, it swings. Immediately, this is again the workspace where you can see the robot animation here. Zoom it, pan it, you can do anything from these blocks. See the default views. So now, I am ready with the model in which I can put all the parameters. All the control parameters create the control loop with this robot, and I can simulate that. So, this is my robot, which is now ready for simulation.



You can see the plot here also. This is how it has moved. You could have sensed X, Y, Z, all and seen the plots. So, this is how I have prepared my robot.

So now, you have seen the introduction of the World Frame, Joint Blocks, Multibody Elements, Multibody Frames and Transforms, Transform Sensors, Simulink Converters, Simulink Sinks, and Scopes. So, using this, you have now created a 3R Spatial Arm, which is ready to be simulated.

Demonstrations

- Calculating and Testing the Robot Jacobian: Using MATLAB Embedded Code and Robotics Toolbox
- Calculating and Testing the Gravity compensation torque: Using, MATLAB Embedded Code and Robotics Toolbox
- Implementing the Environment Interaction model.
- Testing the arm for open loop positioning and free-fall forward dynamics simulation.

Later, we'll also see a demonstration for calculating and testing the robot Jacobian that I'll be doing in the next module. Calculating and testing the gravity compensation torque using MATLAB Embedded code and also using the Robotics Toolbox. I'll be implementing an Environment Interaction model. This I will be doing in my next lecture. Finally, I will test the arm in open-loop positioning and do a free-fall forward dynamics simulation.

So, that is all for this lecture. So, in the next lecture, I will be doing all this. That is all. Thanks a lot.