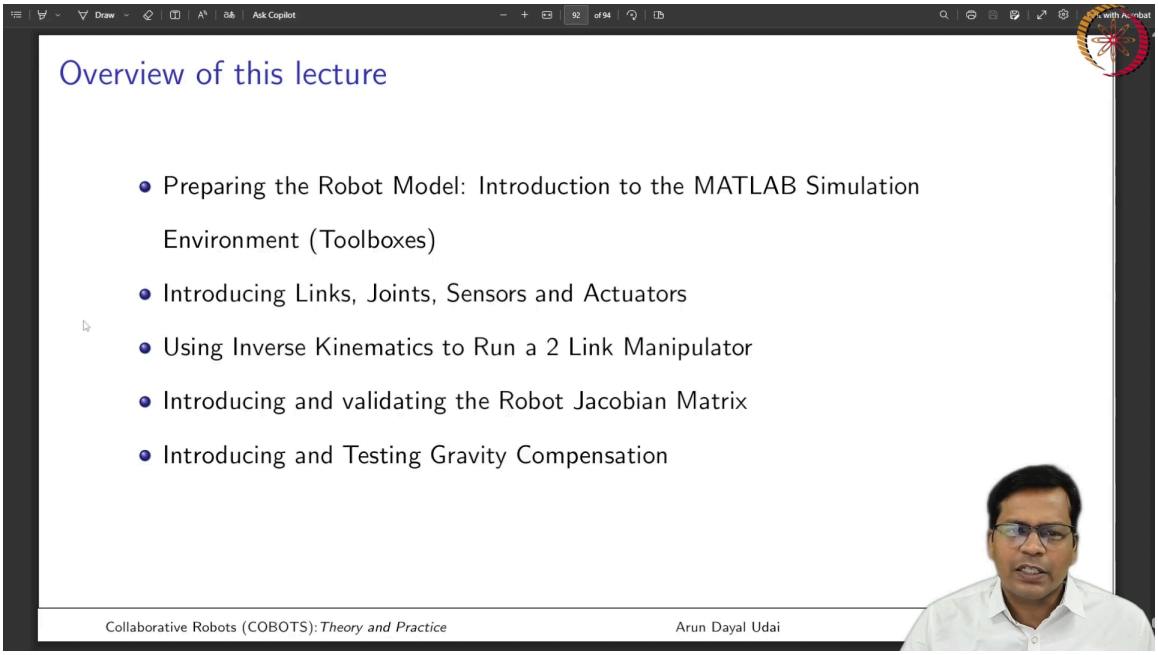
NPTEL Online Certification Courses
COLLABORATIVE ROBOTS (COBOTS): THEORY AND PRACTICE
Dr Arun Dayal Udai
Department of Mechanical Engineering
Indian Institute of Technology (ISM) Dhanbad
Week: 05
Lecture: 23

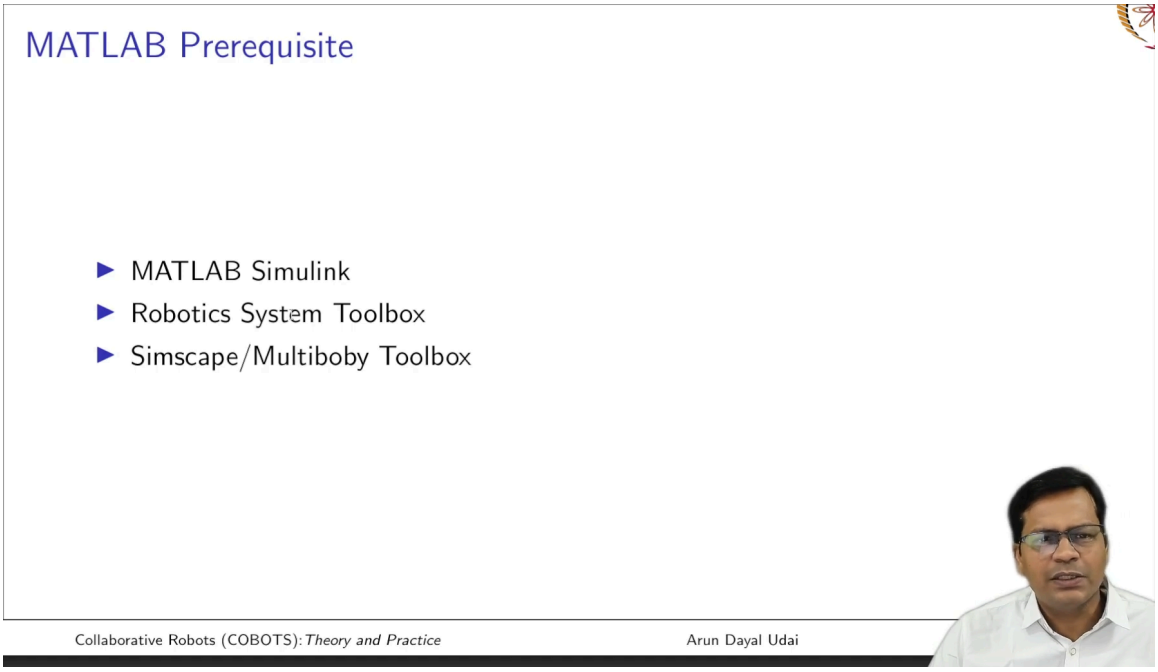
MATLAB Demonstrations

Welcome back. So, in this lecture, I will discuss how to do simulations for robot kinematics, statics, and dynamics using MATLAB's block diagram-based environment, which is commonly known as the MATLAB Simulink Environment.

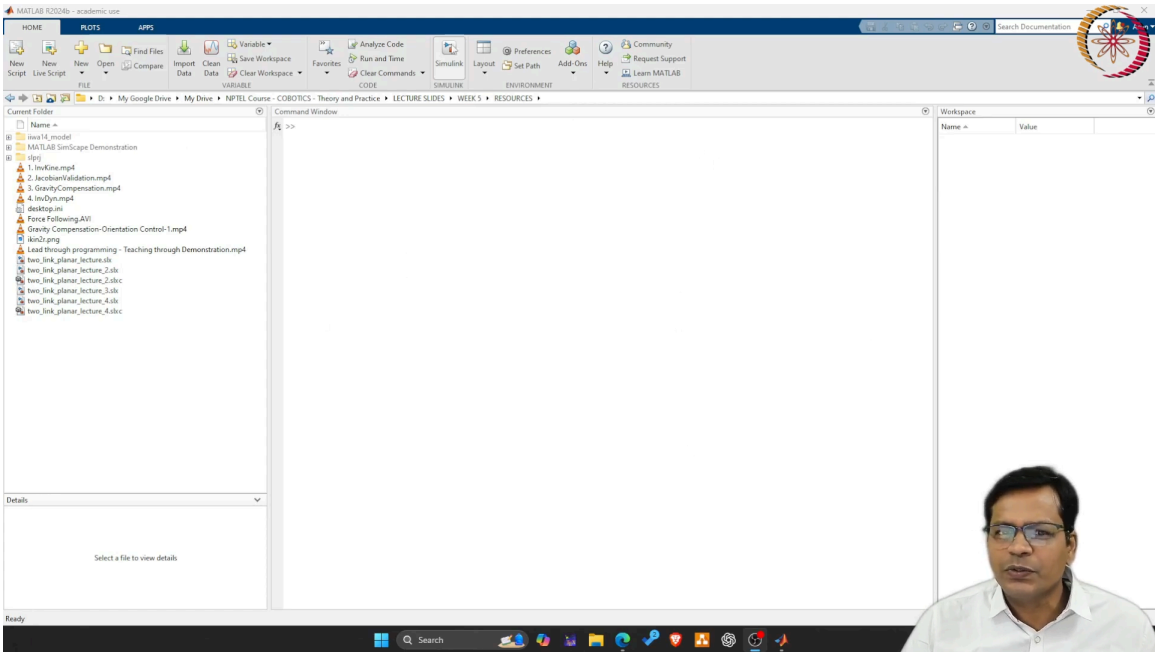


In this lecture, I will let you know how to prepare the Robot Model. I will introduce you to the MATLAB Simulation Environment and various toolboxes that are required to do such simulations. I will discuss how to introduce Links, Joints, Sensors, and Actuators to the joints so that you can actuate the joints using torque, and motion, and you can sense the joint angles and joint torques. So, I will introduce you to all those blocks. I will let you know how to introduce such blocks and run the simulator on a two-link Manipulator. I will introduce how to use the robot Jacobian and how to validate the Jacobian Matrix.

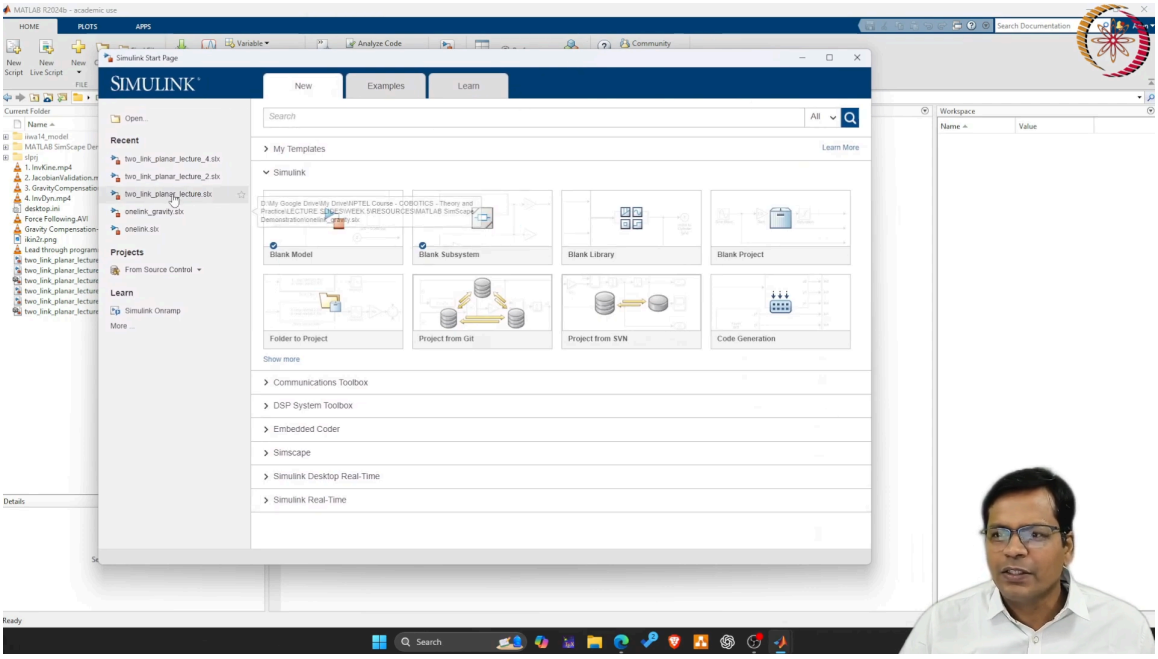
This is very important because once you introduce the Jacobian, it should be validated that it is giving you the correct values for that. So, I will introduce you to that. I will introduce you to the gravity compensation block, and I will test that on a two-link manipulator as well.



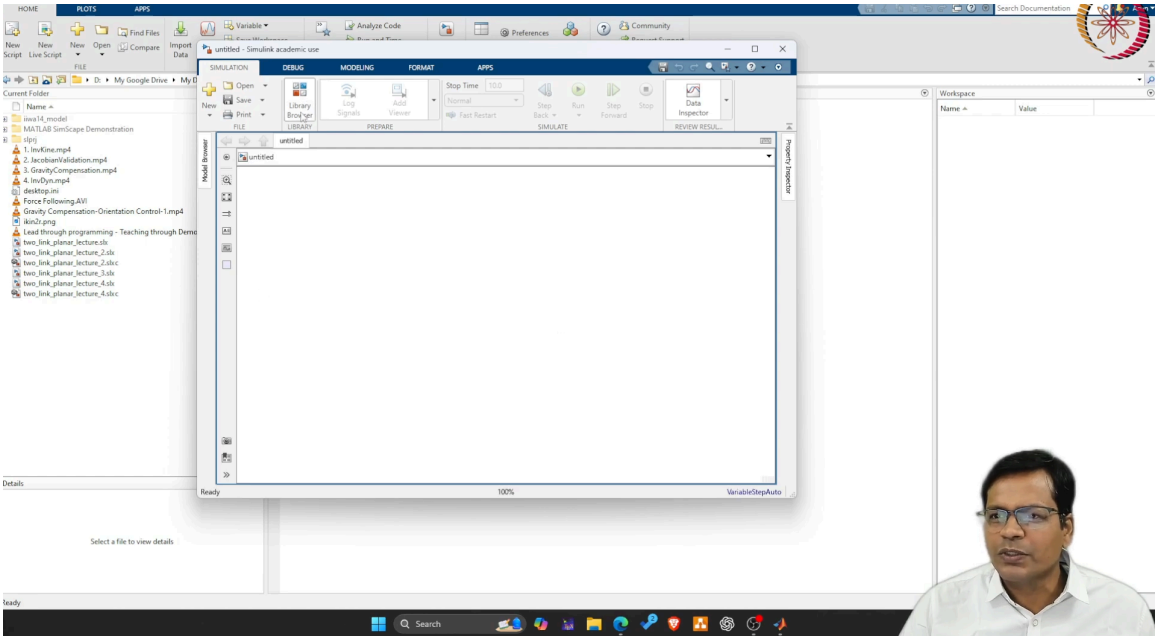
So, these are the following MATLAB prerequisites that you need to have before you do such simulations. You need to have an environment which is known as the MATLAB Simulink environment, apart from MATLAB's standard basic coding environment. You need to have toolboxes like the Robotic System Toolbox that I will be using, and I will also be using Simscape or the Multibody Simulation Toolbox, which is traditionally used for robot simulation as well. So, let us continue.



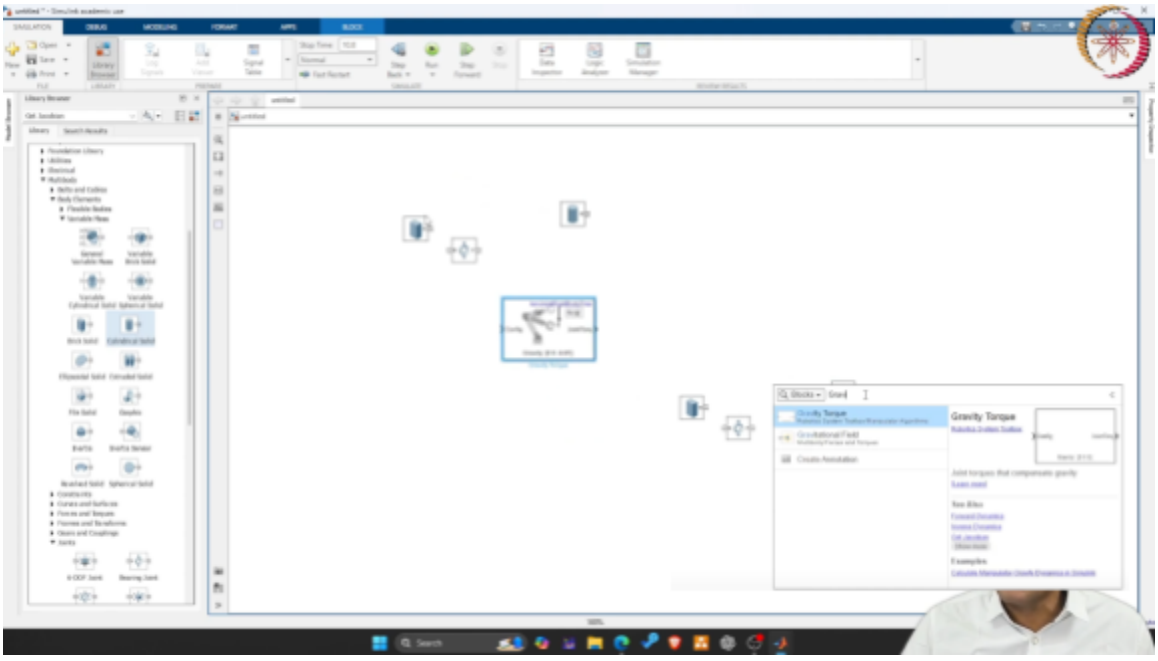
When you start MATLAB, you will see an environment that looks like this. Here, you see a block where it is written Simulink. You need to press that to get the Simulink environment here.



From there, you just need to create a model. You can do this immediately, and you will get a blank space.



You will also see the Library Browser here. Using that, you can see multiple toolboxes that are installed in MATLAB. They are visible out here. This is the Simscape environment, where you see the Multibody Toolbox, which is here.



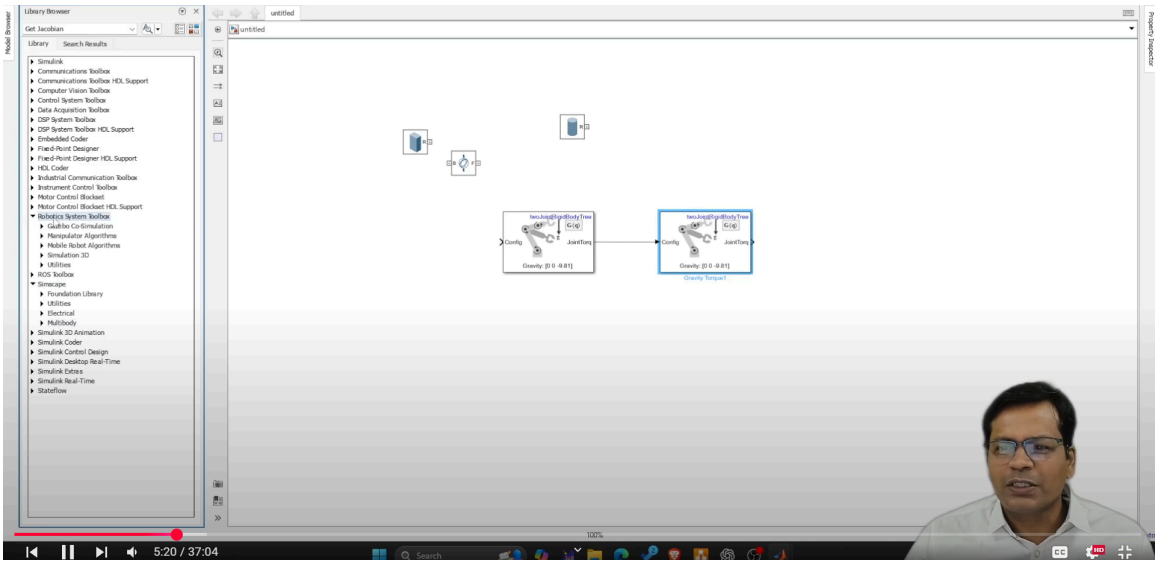
You have multiple joints of different types that can be dragged and dropped into this blank space. So that you can create the system out here using the joints and the links. So, you have body elements here, constraints here, and curves and surfaces here. Here comes

the bearing joint that I have just dragged in here. You can have a brick solid, and you can have a cylindrical solid. You can have different types of other blocks here, like variable mass flexible bodies. All the blocks are here, and they can be dragged directly from here.

Alternatively, you can directly double-click here and search for a particular block, such as the gravity compensation or gravity torque block that can directly be created over here without actually browsing it from the left panel. You can directly get it from here. So, there are multiple ways to insert a block and create a block diagram-based simulation here.

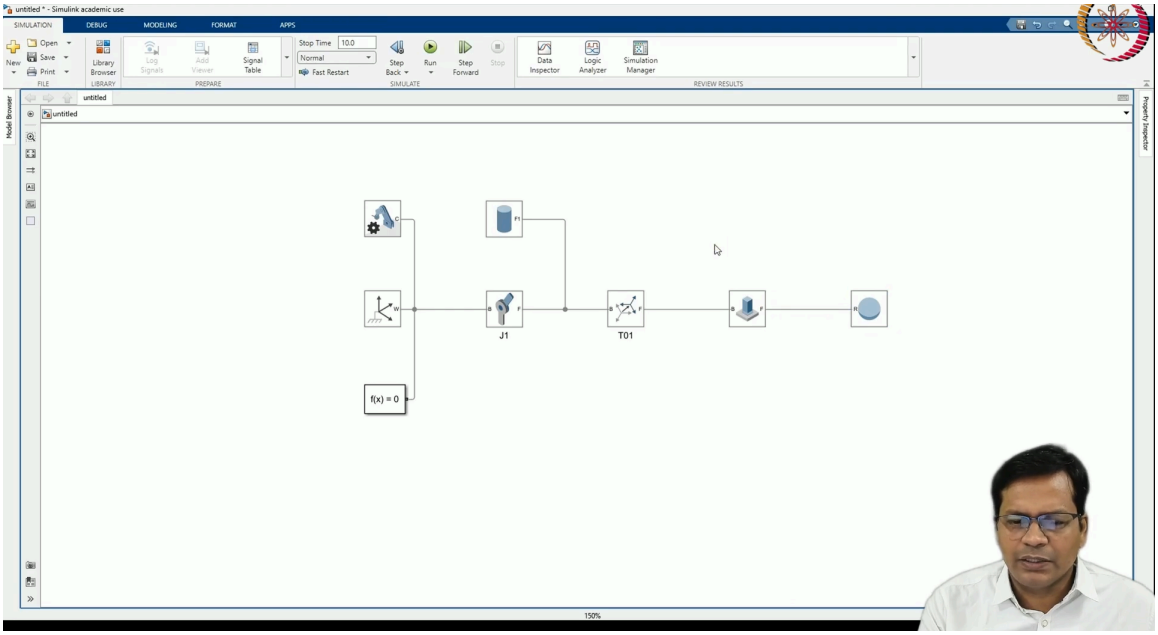
So, this is the environment. These are the toolboxes that I will be using, and then you also have the Robotic System Toolbox that you see here. It includes manipulator algorithms, such as forward dynamics, get Jacobian, get transformation, and gravity torque. So, the gravity torque that I inserted directly by double-clicking and searching for it out here. I could have directly dragged it from here, and I would get exactly the same kind of block.

So, both are the same that I need to use here. So, you see, both are the same. So, in order to connect two blocks, you can just put it somewhere here and click it; it connects. You can create a circuit diagram of various things and various interconnections based on certain logic here. So, this is your environment.

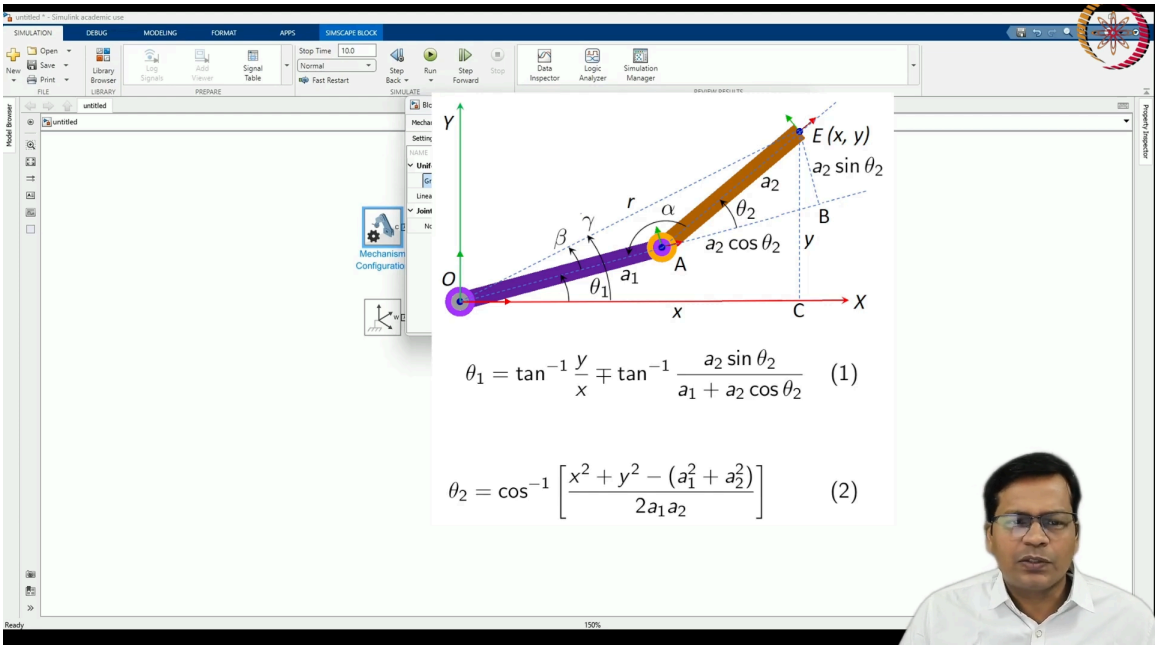


So, there are two major block diagram-based toolboxes that I'll be using. The first is Simscape Multibody, and the second one is the Robotic System Toolbox, where there are

multiple utilities and algorithms. In the Robotic System Toolbox, you have manipulator algorithms that contain most of the things that I'll be required for my simulation.



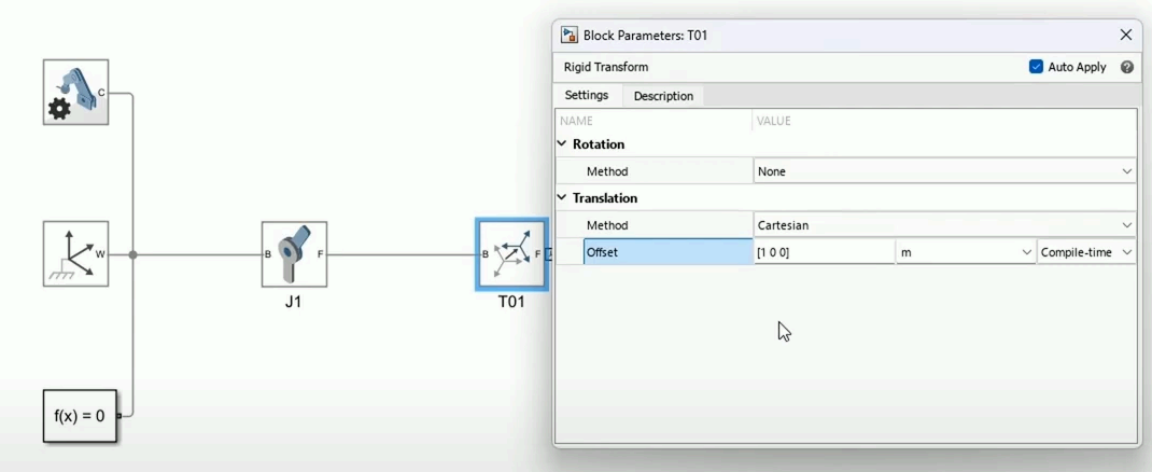
Let us continue and start doing something now, so here I go. So, this is the model that I am creating here. This is the blank space. I have first introduced the block which is called the world frame here.



The second block that I am inserting here is the mechanism configuration. I am setting

the parameter here, which is the direction of g, because this is the mechanism configuration. All the components which are related to the configuration of the environment are here. So, it has a gravity component as well. So, if you remember, your two-link manipulator looks like this. So, you see, you have the x-axis along this, the y-axis like this, and you have link 1, which is shown in violet colour, and you have a dark yellow colour, which is link 2. This is the manipulator that I am going to simulate. So, you see, along the negative y-direction, you should have g.

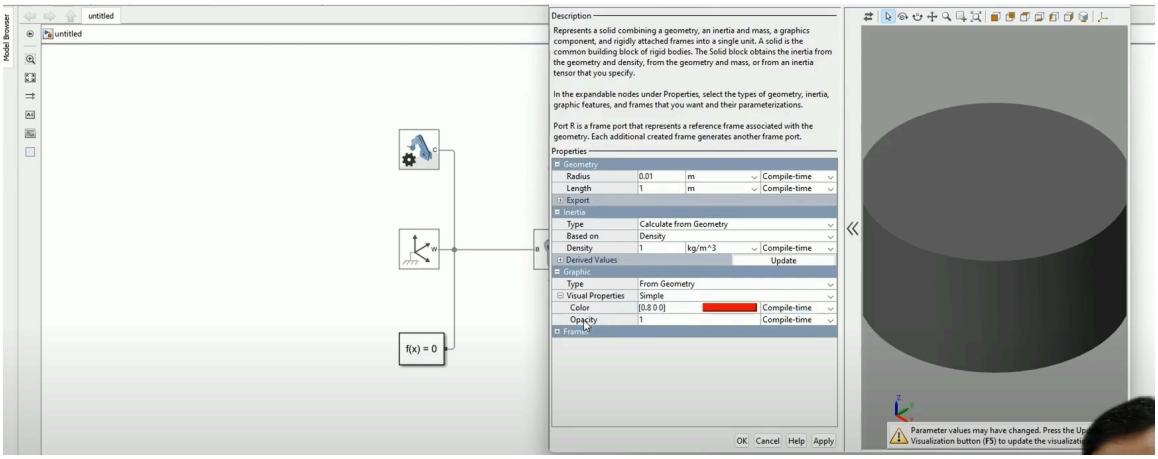
So, this is how the g direction, that is, acceleration due to gravity, is directed downward here, and x is along the positive direction along this way. So, this is how it is, and z is perpendicular to this plane. We need not bother here because it is a planar robot.



So, here I go. So, yes. So, what will I do? I will just put my gravity component in an array form, and it is 0, minus 9.80665, and 0. That means along x, you have acceleration due to gravity that is 0; along y, it is minus 9.8; and along z, it is again 0. So, this is the configuration that I have done. The world frame, basically the first block that I inserted, tells you where your 0 0 0 is placed. So, that block is shown here. So now, I am connecting both the blocks. They will use each other. This is the solver configuration. This is the kind of integrator that this simulation environment is going to use. By default, it is ODE 4 5 here.

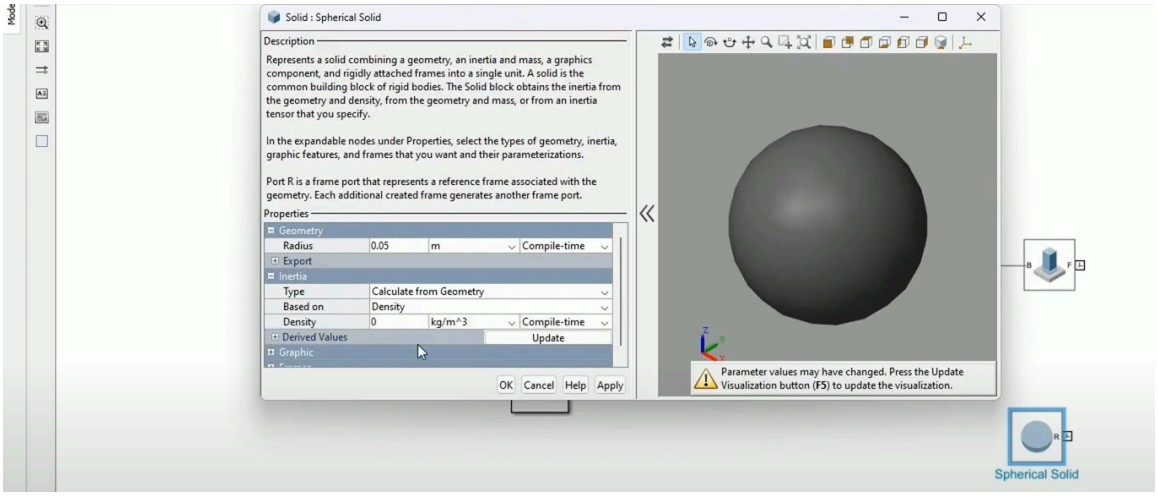
Now, I will introduce the first revolute joint. So, that is the first joint which is attached to the ground. I will connect it to the ground here. The second one is the rigid body transformation, which takes it from joint 1 to the end of link 1. So, this virtually creates

the frames. This joint is the location of the frame, that is, frame one, which is attached at the end of link one, and I have renamed it as J1, that is, joint one. This is T01. So, the method is Cartesian, and The offset is 1 0 0, which is along x. So, you know, all the distances along the link length are measured along x. So, it is 1 0 0, which means the link length will be 1 meter. The unit is set as a meter over here.

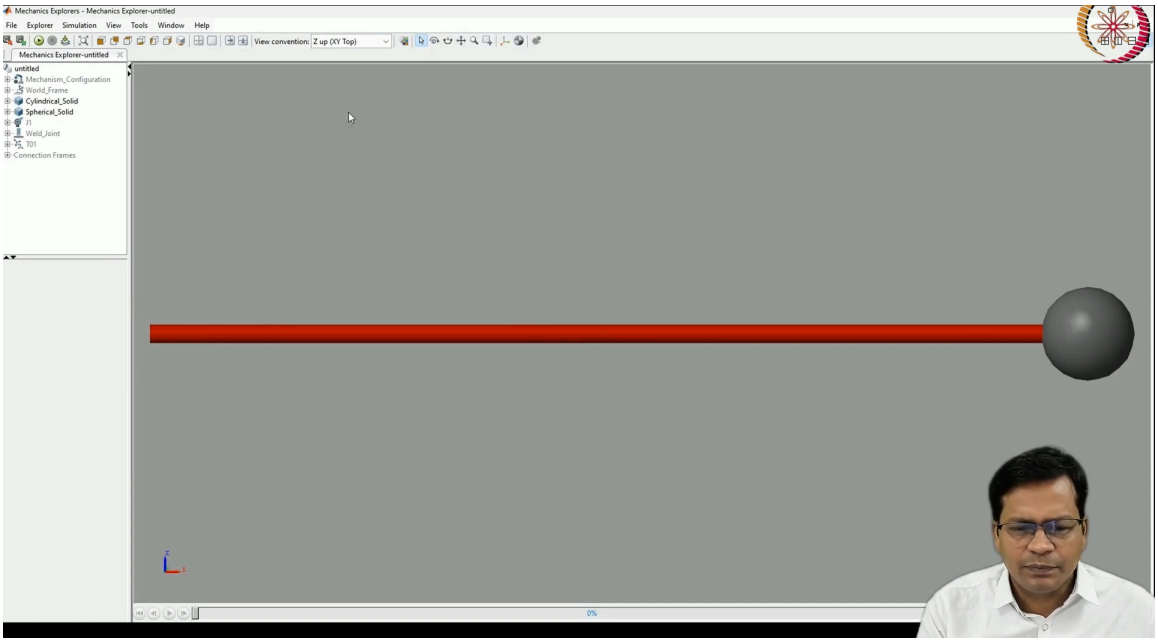


Now, I will place the cylindrical solid block. The radius is negligible. I have set it as 0.01. The inertia parameter I'll set as 1 kg per meter cube. So, that is basically the density of the cylindrical block that I have placed. This is going to become your link. So based on the volume, it will create the mass. If the mass is evenly distributed, it automatically comes at the centroid of that geometry.

Colour, so as to distinguish between different links, I have set it as 0.800. That means it is a 0 to 1 scale for R, G, and B. If it is 0.8 for R and 0 for green and blue components, it becomes almost like red. Now, I have shown the port where I will be connecting. So, this is the frame that I am going to place. I'll put it based on geometry at the end of the link. So, I have selected the surface here, and that direction I have set as the plus X-axis along that direction. You know the link length should be along the X-axis. So, the primary axis will be plus X. This is your link, and I'll connect it to joint 1.

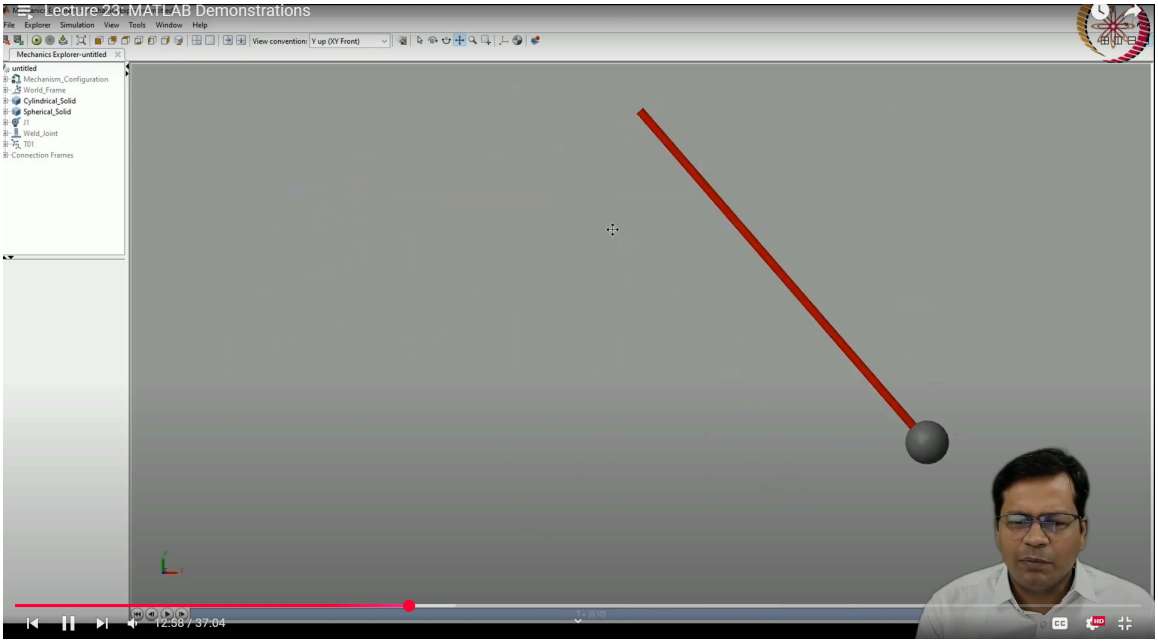


Now, this is just one link that was first made. I have created a weld joint, and I have fixed another frame, which is a blank frame of radius 0.05 mm. It is almost negligible, just to create the frame at the tip of link 1. I am introducing this. This can be your gripper as well. I have set this as 0 density so that it should not have any mass. Whatever the mass is, it will come directly at the link only. Rotated it and connected it. So, this is a one-link system here.

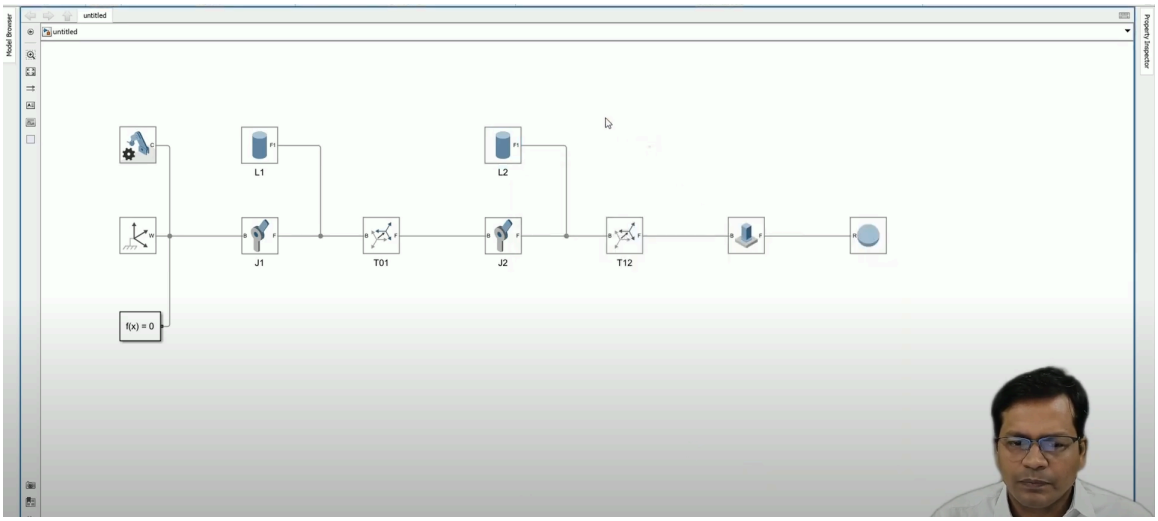


So, this is my diagram. So, updating the diagram is done. That can be done directly from the icon which is there, or you can right-click and update the diagram the way I have done it here. So, this is panning. You can zoom by simply selecting the icon, or you can

pan by selecting the icon, or you can use the mouse buttons, the mouse center button, and drag it or scroll it to zoom in and zoom out. This is the view I have selected. This is how it looked.

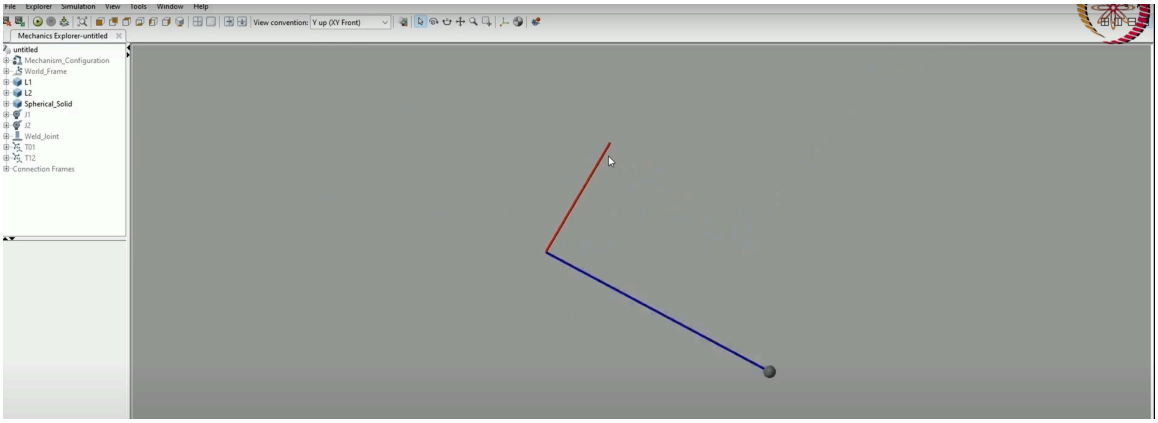


If I run this simulation, it should swing like a pendulum. Isn't it? Because there is zero torque by default, which is there at the joint. So, it swings like a pendulum because there is acceleration due to gravity acting downward.



Now, I will extend this to a two-link manipulator. You can change the solver here, also.

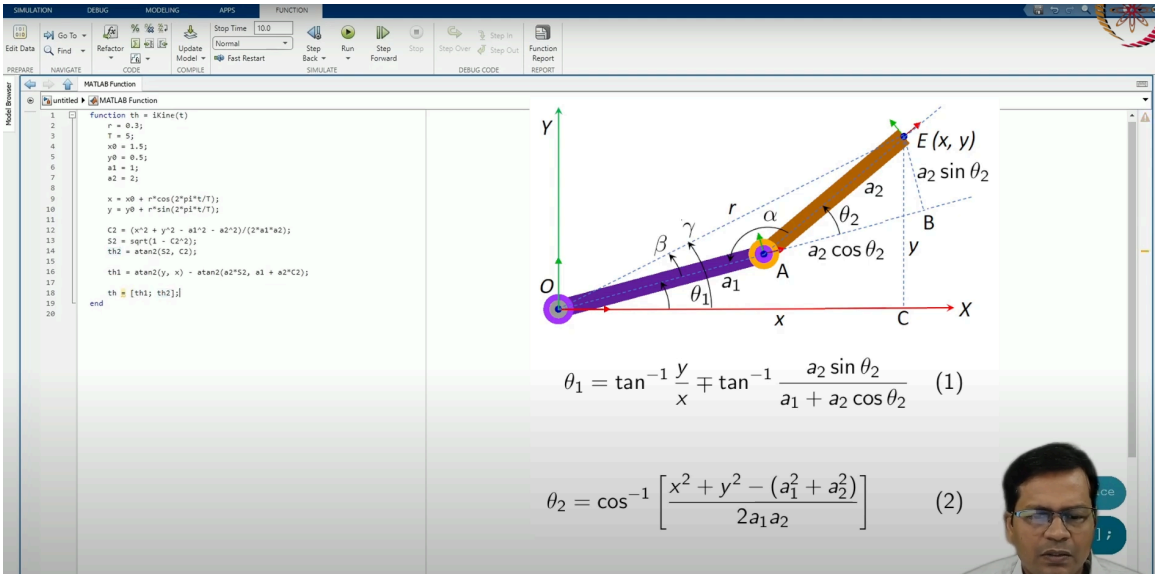
There are various ways to solve the problem, such as forward dynamic simulation, which is done by integrating the dynamic equations of motion.



Now, I will put this at the end. I will introduce a second joint here and name it J2. This is my second joint. So, I will specify a position target. So, you can insert a position at the joints, or you can specify a torque at the joints, also. Actuation methods can be different. It can be actuated using torque, or it can be actuated using motion. So, the other will be automatically calculated.

The state target is basically the starting point of the joint. So, that is set as 0. A similar transformation has to go here. So, it has to be taken to the tip of link 2. So, this becomes T01, and then this becomes T12. This is joint 2. A similar cylindrical solid, I will insert it here. I will rename it as link 2. Earlier, it was link 1, and now, it is link 2. Copy and paste. So, you can simply copy-paste, or you can insert a new block altogether and rename it. Again, the offset will be 2 from the Cartesian frame. So, this makes the link length 2 meters for the second link. The length is 2 meters. Graphic color effect. So, this time, I want it to be blue. So, 0, 0, 0.8. So, that becomes almost blue.

Now, if I run this, it becomes a two-link pendulum, and it simply swings because no torque is applied at the joint, and it is torque-actuated.



Now, this is a function that I am going to do now that you are ready with your whole system of a 2R manipulator with 2 joints and 2 links of 1 meter and 2 meters. So, now you are ready to do anything. So, let us start doing this. Inverse kinematics-based motion here, okay? If you remember your old inverse kinematics equation, the same I am going to put it here. Just keep watching. I am just inserting the equations in MATLAB programming language here. This is a MATLAB function block. This function can take in the end effector position and give you the joint angles, okay? What I am going to do here I'll insert a trajectory at the end, okay? Let's say I want to have a circular trajectory for the tip of the two-link manipulator, okay? So, in those positions I'll keep on pushing to the inverse kinematics function, and in turn, it will give me the joint angles, okay? So, if I feed those joint angles to the joints, it will start moving. The end effector will start moving in a circular trajectory that is planned.

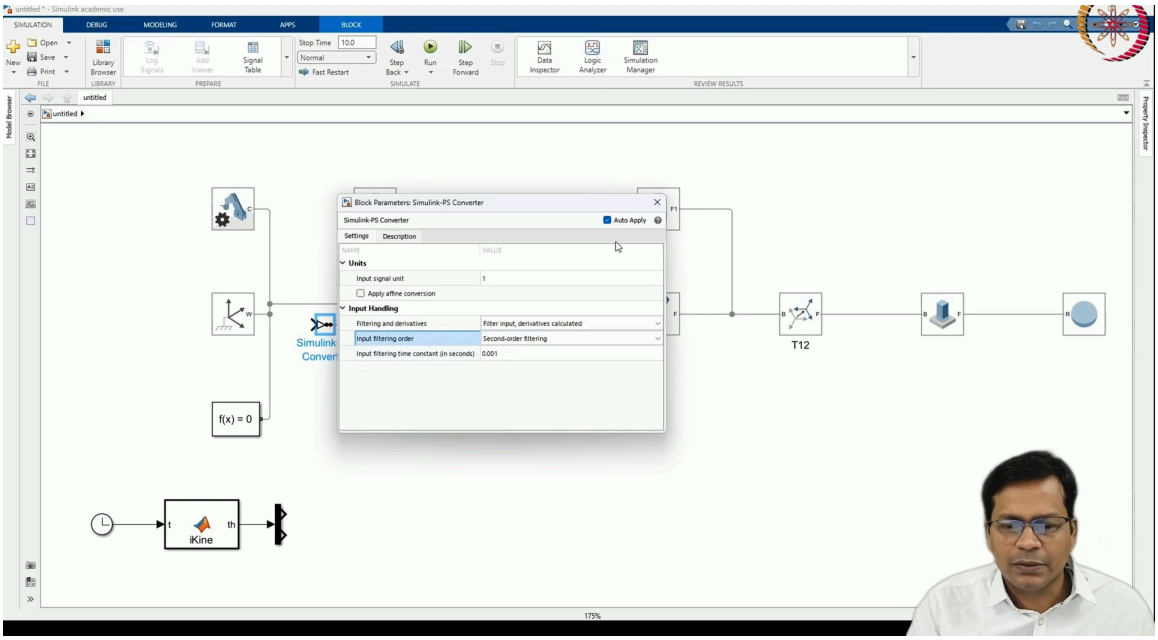
So, let us now start coding it. So, the same, what you can see here is being put here. So, on the right hand, you can see the two-link inverse kinematic solution by equations 1 and 2. So, what I am doing here I am setting r equal to 0.3. That is, the radius of the circular trajectory is 0.3. Total time as 5. x0, y0 is the centre of the circle. It should be within the workspace for the entire planned trajectory. It should not go beyond the two-link arm reach, okay? So, I remember it was one meter and two meters for the link lengths for the first link and the second link. So, the total distance, the total circle, should not go beyond that reach, which is three meters overall from the 0 0 frame. So, x0, y0 is 1.5, 0.5. Link

length 1 is 1 meter. Link length 2 is 2 meters. x is equal to x0 plus r cos theta. Theta will be calculated based on 2 pi t by total time (T). t is the instantaneous time, and T is the total time. So, for one complete total time, it should make one full revolution from 0 to 2 pi. Similarly, y is equal to y0 plus r sin theta. That is the basic equation of the parametric equation of a circle.

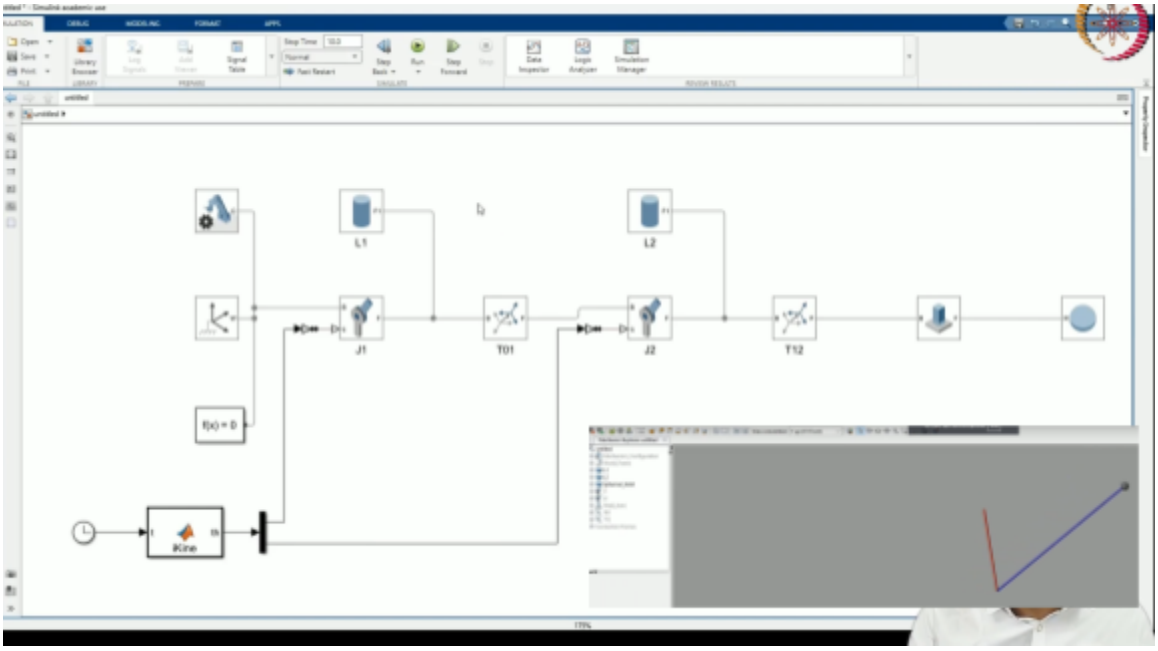
Now, I am putting in the inverse kinematic solution that is there in equation 2. So, this is the cosine of theta 2 being put here. Exactly, equation number 2 is coded here in MATLAB programming language. This is cos theta 2, and sin theta 2 can be calculated based on sin theta is equal to the square root of 1 minus cos square theta. So, that is what is put here. Done. If you know cosine and sine, you use the atan2 solution to calculate theta 2. Your solution should not be ambiguous. So, for that reason, cos theta 2 and sin theta 2 are calculated exactly using this method. Cos theta through the formula sin is directly using the cosine, and theta 1 is as per equation number 1.

So, once this is coded, you are done with theta 2 and theta 1; you have to return these angles directly on th parameter that is defined here. So, th is a vector that will have theta 1 and theta 2, and that is stored now in the variable th.

So, time can be taken from the clock. This block takes the input from the clock block. Now, th I will just do Demux. Demux basically will separate out the two variables that are stored within the th variable. So, it is just to Demux that it will. So, the first variable will get into a joint one. So, joint one will take the motion input provided by the input and calculate the torque automatically. Similarly, joint 2 will take motion input, and torque will be calculated automatically.

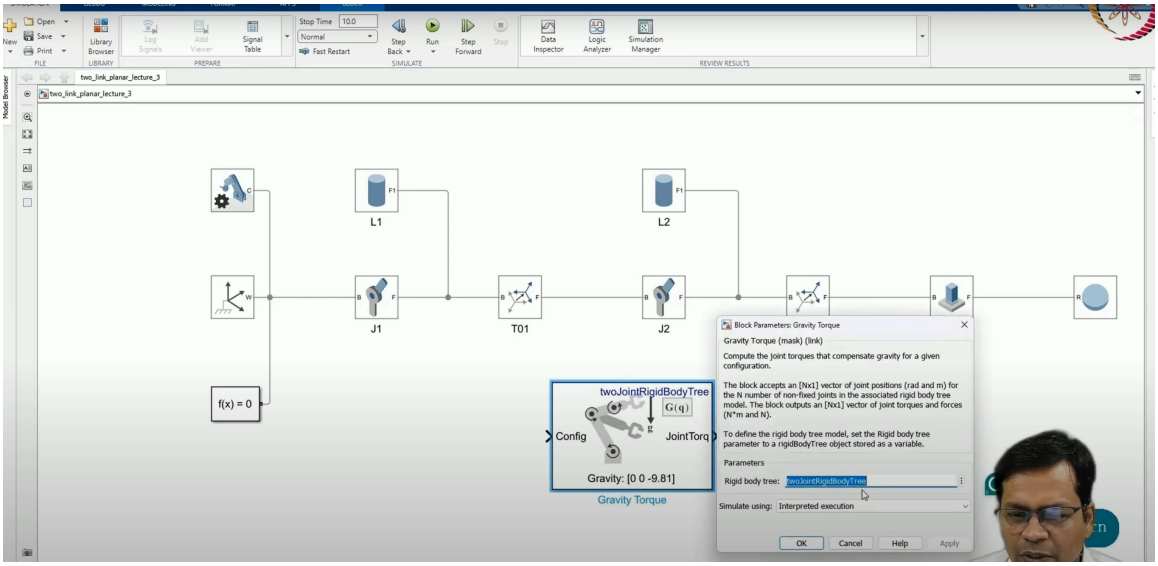


Now, you need to convert the values which are available from the Simulink block. So, the inverse kinematics block after Demux gives values in Simulink data format that has to be converted to PS data format, that is, physical system, because the whole of this is a multibody system, and the data formats are different. So, there is a converter that converts Simulink data format to physical system data format. So, joint angle basically once it comes out of this TH, it has to go like this. So, what I have done here is filter input derivatives are also calculated, and second-order filtering is done. So, this contains both the joint rates as well as the joint angles. So, both are calculated, and they are fed to the physical system block that is joint 1.

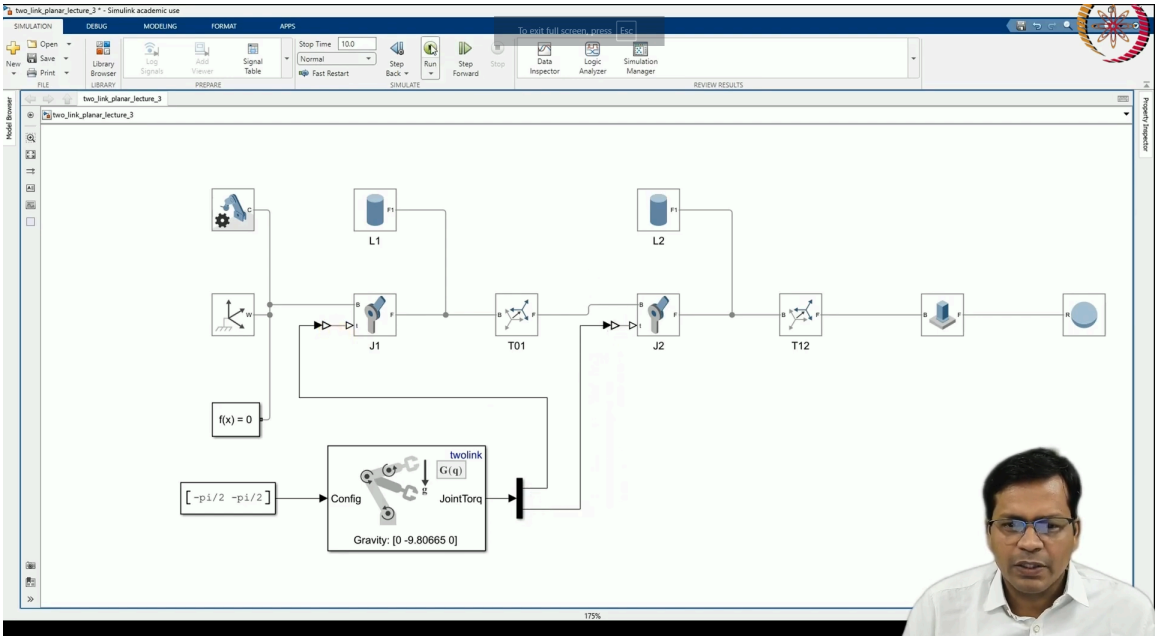


Similarly, for joint 2. joint angle one and joint angle two, after being calculated from the inverse kinematics block, go to the joint angles. Now, I'll run this simulation. You see, it is making a circle in its workspace. I'll pan it. I'll bring it to the centre. It is done very smoothly, and it never goes out of the boundaries. That is to be decided beforehand only.

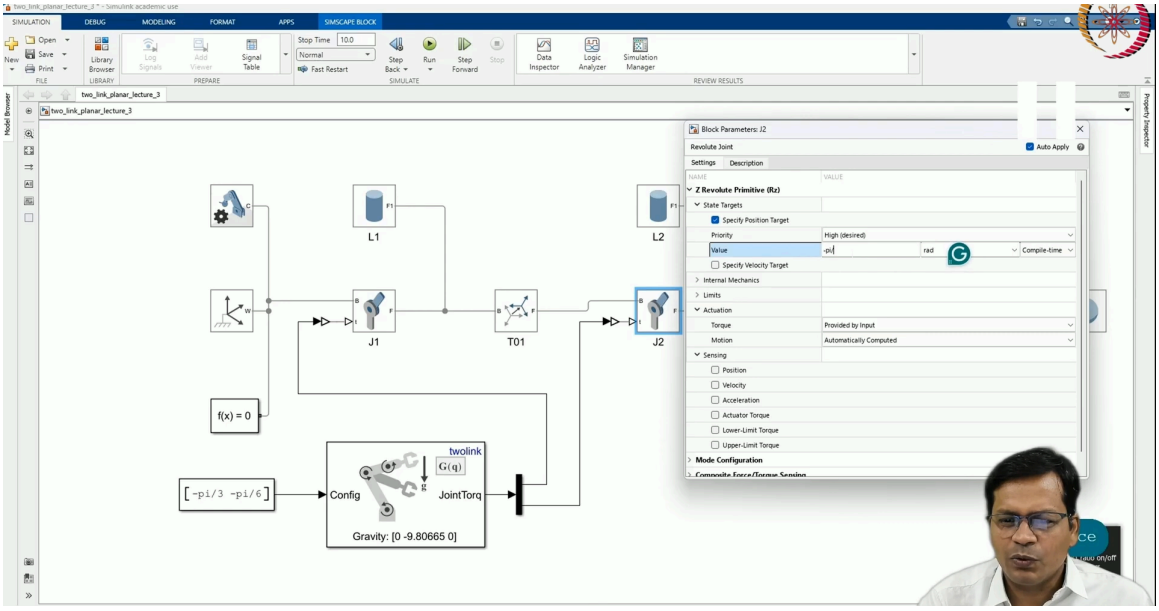
So, now I will make some modifications to this inverse kinematics function. I will increase the radius of this circle now and see what happens. I will run it once again, and in this case, it makes a bigger circle in the workspace. Again, it is not going out of the boundary now. Those checks I haven't put here. All the kinematic solutions do exist within the workspace. Checks for that can also be put in the inverse kinematics programming block, the MATLAB function block. You see, I keep changing the parameters for the circle, and it operates with that. So, the circle centre location can be changed, and link lengths can be changed. So, link length you cannot change. You see, if you have to change it, you have to change it in both locations, that is, the a1 and a2 variables here, as well as in the blocks, that is, the link length block. Everything needs to be changed.



So, now let us use the gravity compensation block. We have already done it using algorithms that we developed earlier in the statics one of the lectures. So, again, I will use to link as a variable and import the Simulink model here to create the object for it. It is to link planar, this one lecture3.slx, and it will create an object here in the workspace. You see, it is created. So, I will use the block gravity torque. It uses the rigid body tree name as tooling. That is put here. Gravity direction, you have to put the same as it is in the configuration. This is the constant that I am putting here that gives you the range minus pi by 2, minus pi by 2, that is the configuration, initial configuration where you want to keep it stationary, the value here minus pi by 2 provided by the input, torque is provided by the input, torque is provided by the input again. These are the joint configuration, and motion is calculated automatically.

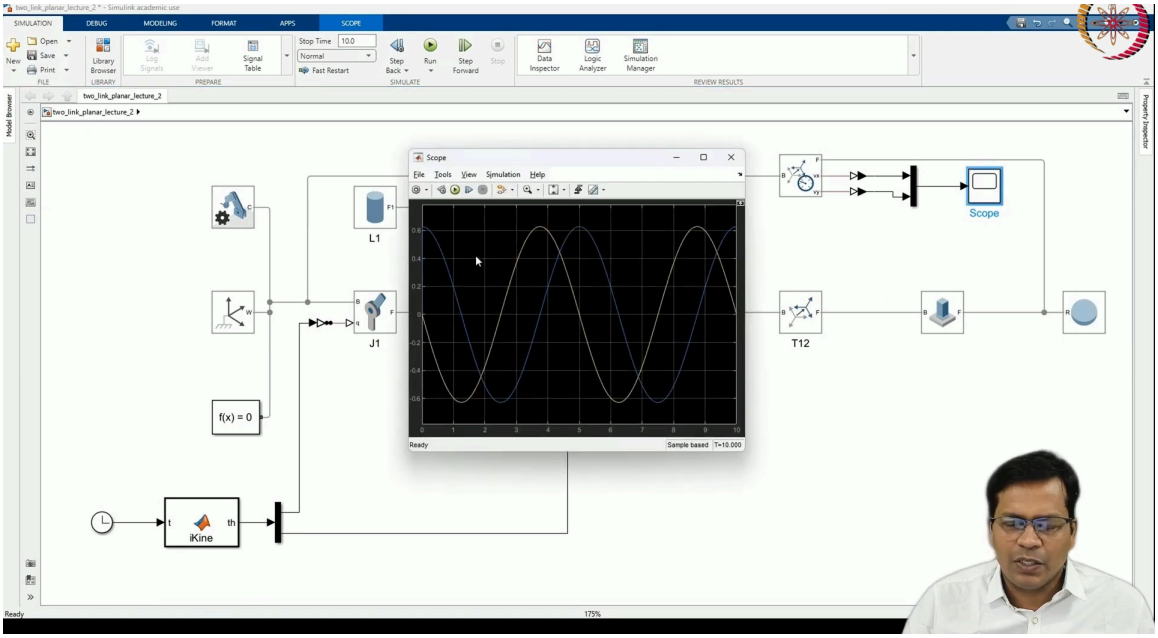


So, joint torque again, I will split them to both the joints. Use the Simulink to PS converter and physical system data converter. Simulink to PS converter that goes to both the blocks now. Now, I will run this simulation. So, angles are minus pi by 2 and minus pi by 2. So, you see for this particular pose, it is continuously calculating the joint torque, and it is fed to the robot joints. So, exactly that much torque is available at the joint, and it makes the robot in that particular pose a stationary robot. So, it becomes a stationary robot. It is not moving because the torque is not varying, and it is controlled by the torque. So, torque was sent as an input, and motion was calculated automatically. That was set at the joint only. So, when you feed in this gravity torque to the joints for this particular angle, minus pi by 2 and minus pi by 2 for joint angle 1 and angle 2, respectively, it makes this robot in this particular pose.



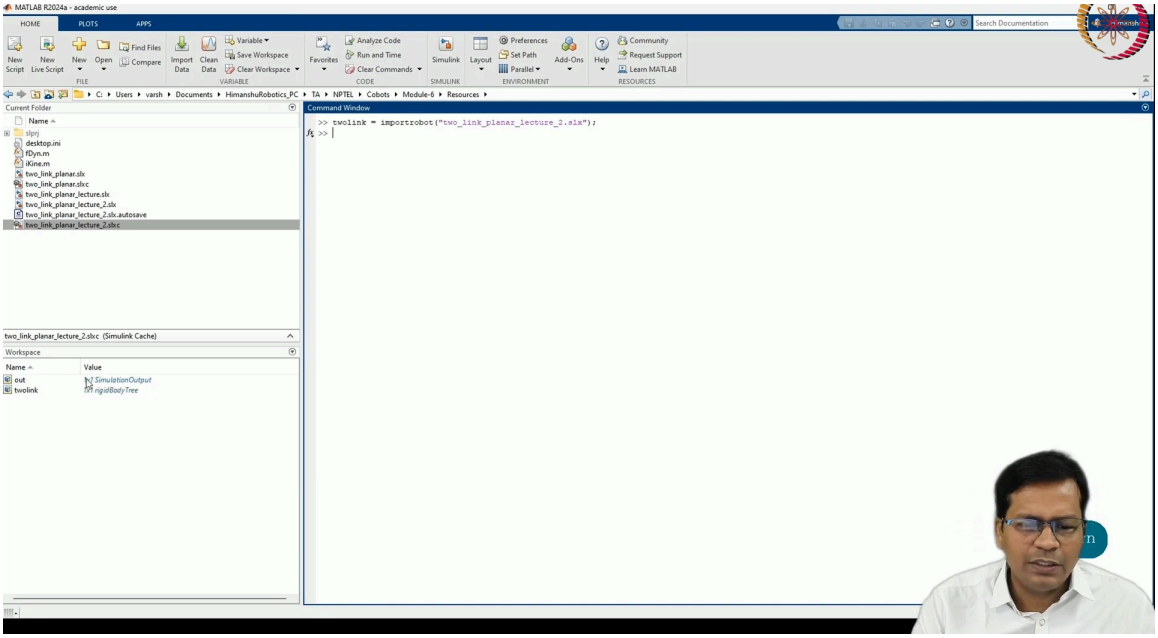
Now, I will change the joint angles, so I'll make it as minus pi by 3 and minus pi by 6. The same is to be set here also, so minus pi by 3 That is the state target. I have set it as minus pi by 3, and you can notice here that torque is provided by the input, and motion is calculated automatically. Same for joint 2. For joint 2, it is pi by 6, minus pi by 6. Now, you run it again and see it is a new position that is taken up by the robot. So, it remains in that particular position. Position exactly with those two angles, so it is started over there, and you are continuously feeding it with that joint torque corresponding to that particular angle from the gravity block. It takes in that torque, and the robot doesn't move at all. Okay, you can try changing the joint angle once again. This time, I am putting it as minus pi by 2 and 0. The same should be put here as an initial state: minus pi by 2, and for the second joint, it is 0. Make it go once again, and it remains like this.

You can try changing this multiple times, and you can validate it based on your hand calculation. Minus 3 pi by 4 and minus pi by 6. You run it, and it goes to a different position now. So this is how you can keep doing this, and you can validate your gravity compensation block. Maybe you can take some positive values also to confirm your exact formulation. So, this time, I am taking it as a positive value also: pi by 2 and pi by 2 for joint angles 1 and 2, pi by 2, and this one is pi by 2. Once again, I will run it. So, this is how this gravity block basically can extract the joint torque corresponding to the joint angle.

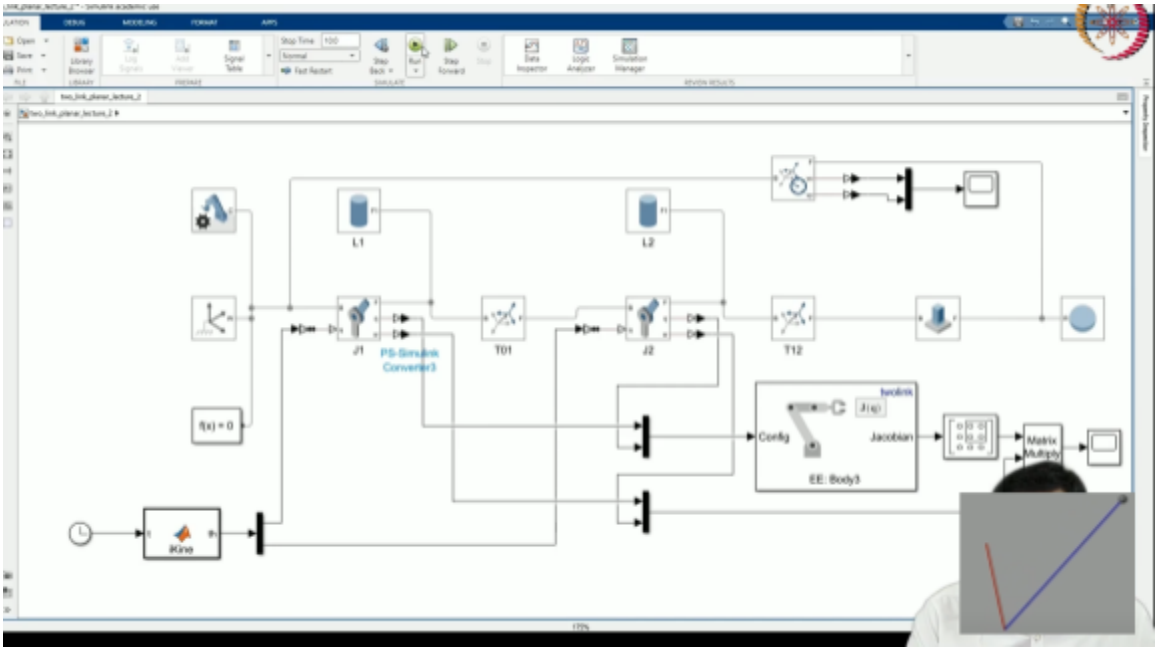


Now, let us use the Jacobian function block. Again, these blocks can be dragged directly from the Simulink library browser, or you can just double-click and insert the blocks. So, I am using the Transform sensor: Why do I want to know the transformation of the end effector frame from the base frame? This is basically doing the forward kinematics. I am calculating the x and y position and velocity. The position can be directly extracted from the end effector frame, but the velocity has to be taken from this block. These two are Simulink data to physical system blocks that I have used earlier. This time, it is otherwise. So, you have to convert the physical system data to Simulink data. So, this is basically extracting the vx and vy, which are the end effector Cartesian velocities along x and y.

Again, I will use a Mux, which will combine them and put them into one single stream, and I will put a scope here to visualise the data coming from this block. I will run it, and I will see how it is coming. You see when the simulation is executing, creating a circle, how the end effector velocities are happening. So, once it is run, I can directly double-click on the scope and get the velocities.

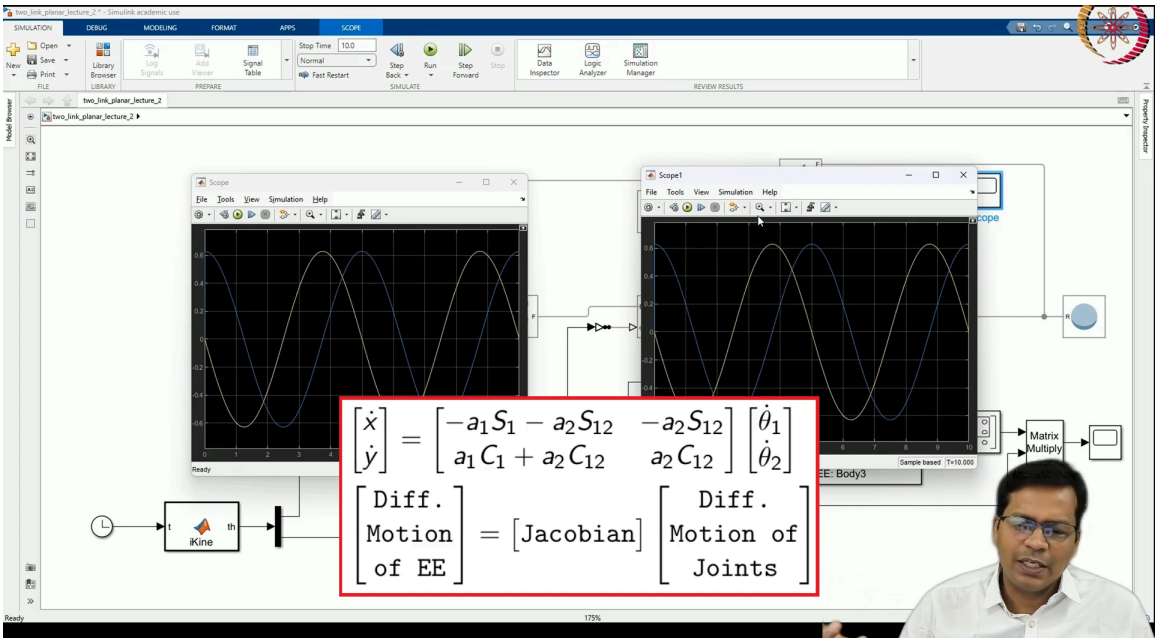


Now, what I will do is import the system model and create an object out of it. This is required in order to use the Jacobian function block. So, I will import the robot, and my simulation model for this is to link the planar. That is for lecture2.slx. So, this is the complete name of the file, and enter it. It creates an object in the workspace.



Now, I will double-click here. I will get the sensing position and velocity from the joint. So, I will check the joint angle position as well as the velocity. So, if I take the joint angle velocities. You need to convert it again from the physical system to Simulink data, from

the physical system to Simulink data. So, both of them need to be converted to Simulink data now. I will pack them together now. Both the joint angle positions are packed together as a single vector. Again, I will pack the angular velocity, and the joint angular velocities here that is simply the joint velocities. Now, I have inserted the block called Jacobian. Get Jacobian. It takes in the rigid body tree object that you have created in the workspace. The body here is body 3. This is a sub-matrix. I am extracting the sub-matrix from the complete Jacobian because, you know, you just have a complete Jacobian, but the complete Jacobian has end effector angular velocity as well as the Cartesian velocity component because I know I just need the end effector Cartesian velocities. So, I am extracting the elements from this. So, the range of rows. So, the index from the starting row is fourth, and the ending row is fifth. So, actually, there are six rows in total. So, it gives me the Jacobian. If I multiply it with joint rates, it should give me the end effector rates. So, you know that formula directly. So now the scope should give me the end effector velocity. The body was body 3, which is the final body.



Now, if I run the simulation, Once the simulation is complete, I can check the velocities that come directly using the transformation and the Jacobian matrix. Both of them are next to each other here. You see, both of them are the same. This is how you validate your Jacobian. Everything should be consistent, and you have used the formula here. Which one, basically? It is Jacobian multiplied by joint rates gives you the end effector

velocity. I got it. Using the Jacobian matrix as well as using the direct transformation matrix as well. Both are the same.

So, that is all for this lecture. In the next lecture, we will start doing Robot Dynamics. Thanks a lot.