CHAPTER

8

Simulation

8.1 OVERVIEW

By using a robot simulator, the time and cost for developing a controller of a humanoid robot can be reduced. At present, the employment of a robot simulator is indispensable for research on humanoid robots. This is the reason this chapter has been included at the end of this work. Robot simulators receive reference values (position, velocity, acceleration, torque) for the joints of a virtual robot. These quantities are generated by a controller designed by the user. In this way, the behavior of the robot can be simulated in a virtual world. The modeling and control methods described in the previous chapters can be tested beforehand in a robot simulator, without conducting experiments with the actual humanoid robot. Thus, it is highly beneficial to know how to use a robot simulator.

In order to control a humanoid robot, the user should prepare a control program for the intended task. First of all, the user should investigate thoroughly whether the control program for the task has already been developed and whether it is available to download from a server or not. If the source code is already available, it is possible to significantly shorten the development period, even if some modification is required. If a novel algorithm is proposed or there is no existing program, the development will proceed according to Fig. 8.1.

As a first step of the development process, a model or an algorithm for control is considered. The previous chapters of this book focus on this problem. In addition to the performance

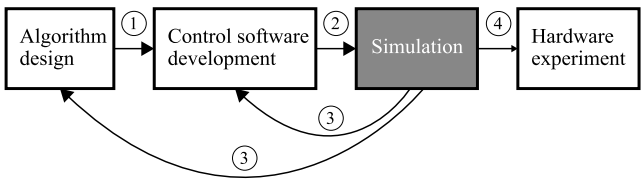


FIGURE 8.1 Common development flow of a robot controller. First, a model or an algorithm for control is considered. Second, whichever is chosen is implemented as a controller program. Third, the control program is verified in a robot simulator. If problems are found in the controller implementation or the algorithm at this stage, they should be reviewed. Finally, the controller is evaluated using an actual humanoid robot.

https://doi.org/10.1016/B978-0-12-804560-2.00015-8

of the algorithm, another factor for efficient controller development is how easy it can be implemented.

The second step is to implement the model or the algorithm as a controller program. In many cases, at this phase the user would consider hardware-specific limitations, e.g. joint angle limits and joint torque limits. It is desirable to develop an universal controller by generalizing the hardware-specific problems, so that the controller can be used for various robots. Therefore, the hardware-specific information should not be hard-coded into the program. It is also desirable to prepare files that describe the structure of the robot. There are specific formats or programming languages for this purpose, such as the SDF (Simulator Description Format), the URDF (Universal Robotic Description Format), the YAML (YAML Ain't a Markup Language), and the VRML (Virtual Reality Modeling Language). The files can then be easily imported by the simulator program.

As a third step, the control program is to be evaluated in the robot simulator. Note that, when using a human-sized humanoid robot, the evaluation of the developed controller can be a tough job since the process of conducting experiments is time consuming and costly. Also, there is the risk of breaking the robot. The process of simulation is beneficial even when a small-size humanoid robot is used. For that reason the user can test the controller under development and obtain the status of the robot that is computed during the simulation. Even when the robot does not have the means for measuring the status, the user can analyze the status in the simulator and in this way debug the controller. Any problems found at this stage of the implementation of the controller algorithm should be reviewed.

At the final step, the developed control program is evaluated using the actual humanoid robot. This phase is indispensable since, despite a successful test result, the robot simulator cannot guarantee a robot behavior that matches completely that of the actual robot. There is always a possibility that, when using the actual robot, a phenomenon may occur that did not occur at the simulation stage. Thus, experiments with the actual robot have to be carried out with great attention.

This chapter is structured as follows. Section 8.2 gives an outline and introduces some well-known robot simulators. In Section 8.3, the structure of a generic robot simulator is described and shown in flowcharts. In Section 8.4 it is explained how to use Simscape Multibody in a step-by-step manner, so that the reader can conduct a simple simulation.

8.2 ROBOT SIMULATORS

A simulator plays a key role in mechanical design, control, motion planning, and so on. The main components of a robot simulator are: a model loader, a physics engine based on multibody dynamics, contact models, and a visualization interface. In many robot simulators, users can access the functions of the simulator using a programming language such as Python, C/C++, or MATLAB. Recent robot simulators also comprise a Graphical User Interface (GUI) for controlling the simulation. With a dynamics-based simulation of a humanoid robot, it is possible to evaluate postural stability, estimate the forces between the robot and the environment, check the actuator torques generated to perform the desired motion, and so on.

The user programs the controller, which communicates with the simulator, using a programming language. The controller is tested in the simulator and is modified, if necessary, according to the simulation results. Several robot simulators, e.g. Gazebo with the Robot Operating System (ROS) and Choreonoid, can transparently handle actual and virtual robots. In other words, a controller developed in the robot simulator for a target humanoid robot can be used in the actual robot without changing the source code.

In some robot simulators, it is possible to emulate external sensors mounted on the robot, e.g. a camera, a depth sensor, or force/torque sensors. The virtual sensors are helpful to develop controllers for object handling, mapping and localization algorithms, etc. A large number of robot simulators have been developed so far [30,31,10]. Some well-known simulators are listed below.

Gazebo: Gazebo is an open-source software for robot simulation running on Linux [35,18] which is commonly used with the (ROS) [52,1]. This simulator was used in the Defense Advanced Research Projects Agency (DARPA) Virtual Robotics Challenge (VRC) [2]. The core libraries for the physics engine in Gazebo are Open Dynamics Engine [48], Bullet [7], Simbody [56], and DART [11]. The Gazebo framework can easily switch between these physics engines. Since Gazebo communicates via ROS messages, the developed controllers can be used on actual machines without modifying the source code [54]. A number of control packages for the ROS environment have been developed in the community that can be used with Gazebo. For example, MoveIt! is a well-known package for motion planning [8,43].

Choreonoid: Choreonoid is an open-source robot simulator running on Linux and Windows. It includes a choreography function [44]. The core library for the physics engine in Choreonoid is the AIST engine developed by the National Institute of Advanced Industrial Science and Technology (AIST) in Japan [45]. The library was developed for the open architecture Humanoid Robotics Platform (OpenHRP) [32]. In addition to the AIST engine, the Open Dynamics Engine [48], Bullet [7], and PhysX [51] can also be used with Choreonoid. The functions of Choreonoid can be extended by plug-in modules. For example, graspPlugin can solve the grasp planning, trajectory planning, and task planning problems [24]. The ROS plug-in provides the communication interface with ROS. This simulator will be used in the Tunnel disaster response and recovery competition during the World Robot Challenge 2018 [62].

Vortex Studio: Vortex Studio, developed by CM Labs Simulations, is an interactive real-time multibody dynamics simulation software [67]. Unlike a nonreal-time multibody dynamics tool that focuses on motion generation in a virtual world only, it is a tool for physical simulation that requires human interactive manipulation input in real-time. The full-function version of this software is for profit, but Vortex Studio Essentials with restricted functionality can be used for free.

V-Rep: V-Rep is a commercial cross-platform robot simulator and runs on Windows, macOS, and Linux [53,65]. An unlimited educational version is available for free. Robot controllers for V-Rep can be written in C/C++, Python, Java, Lua, MATLAB (Mathworks, Inc.) [41], or Octave [21]. The core libraries for the physics engine in V-Rep are Open Dynamics Engine [48], Bullet [7], Vortex Dynamics [67], and Netwon Dynamics [47]. V-Rep also has an application programming interface (API) to communicate with ROS.

Webots: Webots is a commercial cross-platform robot simulator that runs on Windows, macOS, and Linux [68]. Webots can import robot models written in VRML formats and maps for virtual environment from OpenStreetMap [49] or Google Maps [22]. The humanoid robot models of Atlas [3], DARwIn-OP [25], HOAP-2 [27], Nao [23], etc. are included in Webots. The robot controller for Webots can be written in C/C++, Python, Java, or MATLAB. The core library for the physics engine in Webots is an extended version of the Open Dynamics Engine [48]. Webots also has an interface API to communicate with ROS.

FROST: FROST is an open-source MATLAB toolkit for developing model-based controllers and planning algorithms for dynamics of legged locomotion [26,17]. FROST is an integrated software for modeling, optimization, and simulation of robotics systems focusing on the hybrid dynamical system and virtual constraints. By using FROST with a Nonlinear Programming (NLP) solver and Mathematica (Wolfram Research, Inc.) [40], a motion planning problem can be solved as a nonlinear constrained optimization problem.

Simscape Multibody: Although the simulators described above are very sophisticated, it takes time to learn how to use the simulator, and it is not the shortest route to validate algorithms focusing on the modeling and control described in this book. Simscape Multibody [57] is a commercial multibody dynamics–based simulation software and a toolbox for Simulink [59] (Mathworks, Inc.). The advantage of Simscape Multibody is that simulation models can be written easily with Simulink-like block diagrams. Also, a number of useful MATLAB/Simulink functions and blocks can be utilized. The core libraries for the Simscape Multibody physics engine are commercial and implemented in MATLAB/Simulink. In addition, a connection with ROS is available, and it is possible to use various ROS functions via the Robotics System Toolbox. The usage of Simscape Multibody will be described in Section 8.4.

8.3 STRUCTURE OF A ROBOT SIMULATOR

As mentioned in Section 8.2, a robot simulator consists of a model loader, a physics engine, and a visualization program (cf. Fig. 8.2). The physics engine, or dynamics-based simulation program, can compute the behavior of the humanoid robot based on the multibody dynamics. In order to compute the behavior, the physics engine has to include components for rigid-body physics, collision detection, and contact physics [42]. Several physical engines include a model loader and visualization functions.

The following explanation is given according to Figs. 8.3 and 8.4. Figs. 8.3 and 8.4 show the flowcharts of inverse dynamics–based simulation and forward dynamics–based simulation, respectively. In the beginning of the simulation, the robot's structure and the simulation parameters are loaded. In the simulation loop, the first step is to check the contact state between the robot and the environment and to compute the contact force. As a second step, the robot controller computes the input joint acceleration for the inverse dynamics–based simulation or the input joint torques. In the third step, the dynamics simulator computes the motion of the

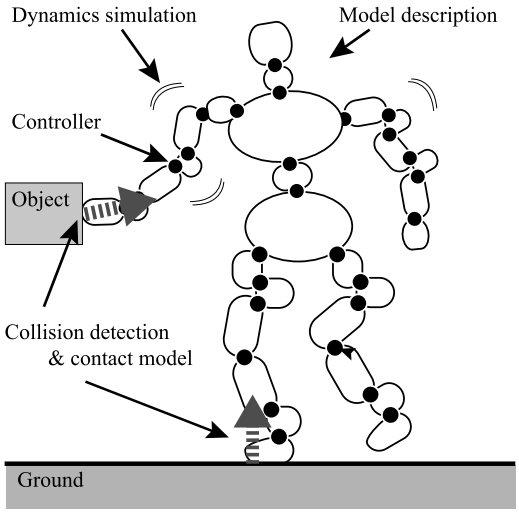


FIGURE 8.2 Required components for a robot simulator. The core libraries of the robot simulator are a model loader, a physics engine, and a visualization program. The physics engine can compute the behavior of the humanoid robot based on multibody dynamics. In order to compute the behavior, the physics engine has to include components for rigid-body physics, collision detection, and contact physics.

robot based on the input. Finally, the computed motion is visualized in the form of computer graphics. The details of each part are explained below.

Model loader: In the beginning of the simulation, the model loader parses the robot model, which includes the kinematic and dynamic parameters, and sends this information to the physics engine. There are various formats for describing the structure of a robot. YAML and VRML formats are supported by Choreonoid [5]. The Simulator Description Format (SDF) and Universal Robotic Description Format (URDF) are supported by Gazebo [38,64]. Simscape Multibody supports URDF, as explained in Section 8.4.1. In addition, the simulation parameters, e.g. the sampling time, the solver, and the contact model parameters, are loaded.

Collision detection & contact model: The collision detection program computes the interaction between the robot and the environment, or the self-collision of the robot [66,13]. By using this program, collision positions at the robot's links and the respective reaction forces can be computed based on the specific contact force model [16]. These values are substituted into the equation of motion of the robot. An implementation example of a collision detection procedure and a contact model is explained in Section 8.4.4.

Robot controller: The robot controller computes the joint-space reference values like joint torques, positions, velocities, and/or accelerations. In the case the controller computes reference joint torques, the robot motion is obtained by a forward dynamics–based simulation. This type of controller is mostly used in humanoid robots with torque-controlled actuators [12,28,60].

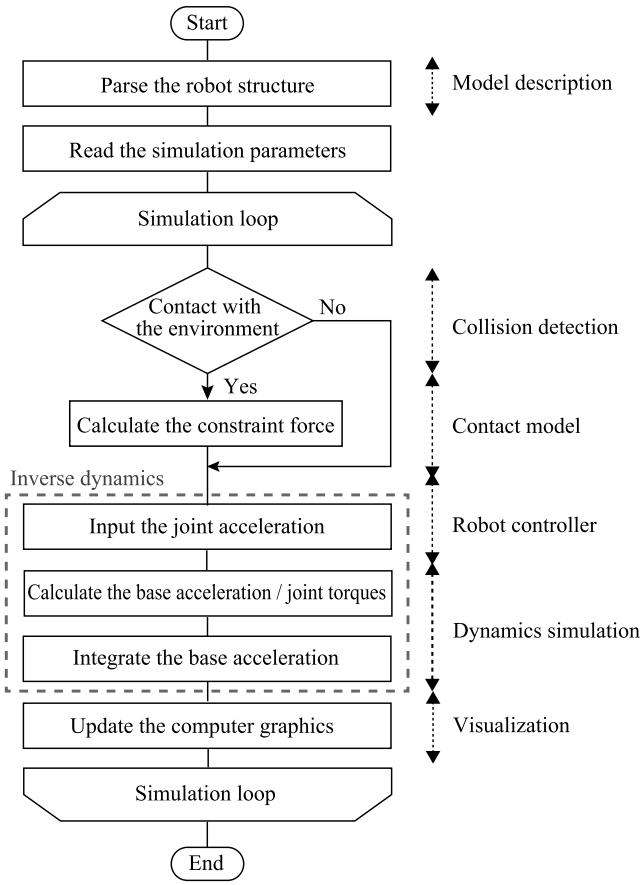


FIGURE 8.3 Flowchart of a robot simulation based on the inverse dynamics. In the beginning of the simulation, the robot's structure and the simulation parameters are loaded. In the simulation loop, the first step is to check the contact state between the robot and the environment and to compute the contact force. As a second step, the robot controller computes the input joint accelerations. In the third step, the dynamics simulator computes the motion of the robot based on the input. Finally, the computed motion is visualized in the form of computer graphics.

In the case the controller computes the joint positions, velocities, or accelerations, joint-servo controllers and joint models are required in the simulator. Fig. 8.5 shows an example of a joint-servo controller and a joint model implemented in a forward dynamics–based simulation. The joint model consists of an actuator, a driver circuit, and a reduction gear. The robot controller computes the reference joint angles while the servo controller controls the actual joint angle, according to the reference in Fig. 8.5. This type of controller is mostly used in a humanoid robot with position- or velocity-controlled actuators. Many motion-controlled humanoid robots have Harmonic drive reduction gears [33,36,50,55]. Models of such gears can be found in [34] for example. Several small-size humanoids have small, smart actuators that consist of a microcon-

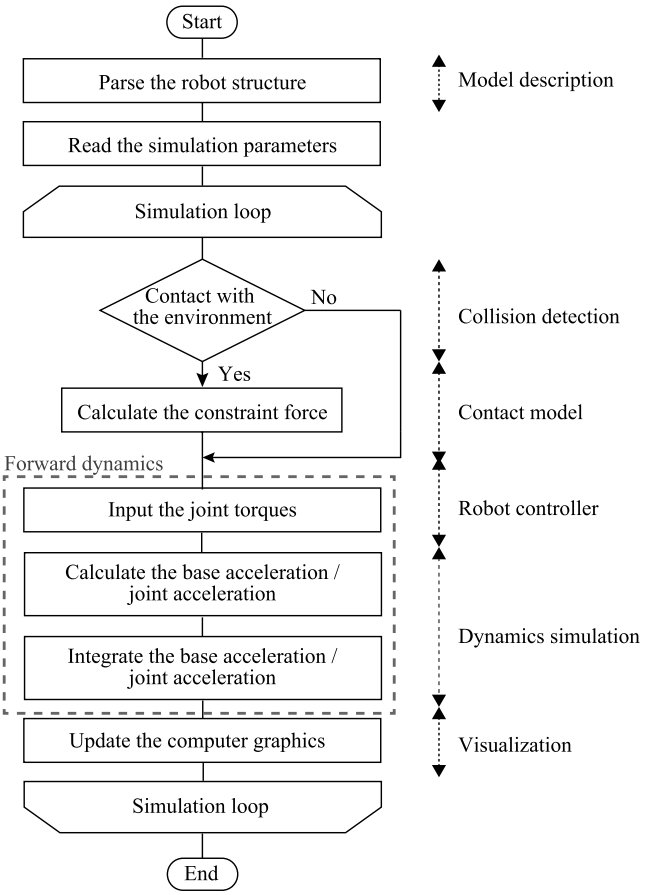


FIGURE 8.4 Flowchart of a robot simulation based on the forward dynamics. In the beginning of the simulation, the robot's structure and the simulation parameters are loaded. In the simulation loop, the first step is to check the contact state between the robot and the environment and to compute the contact force. As a second step, the robot controller computes the input joint torques. In the third step, the dynamics simulator computes the motion of the robot based on the input. Finally, the computed motion is visualized in the form of computer graphics.

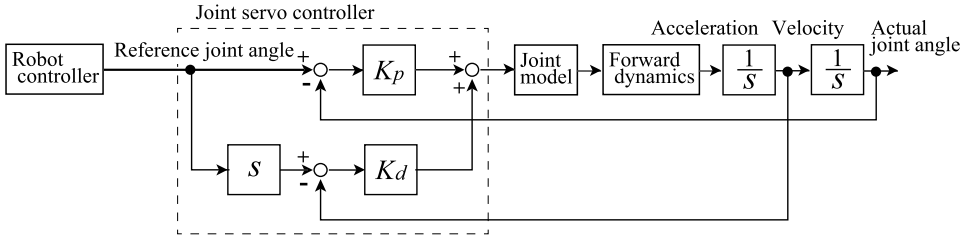


FIGURE 8.5 An example of a joint-servo controller for forward dynamics–based simulation. The joint model consists of an actuator, a driver circuit, and a reduction gear. The robot controller computes the reference joint angles and the servo controller controls the joint angle according to the reference.

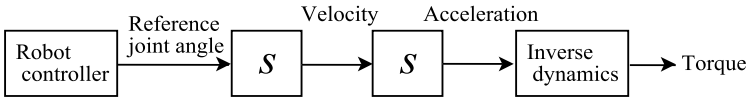


FIGURE 8.6 An example of a joint-servo controller for inverse dynamics–based simulation. The robot controller computes the reference joint angles. The second time derivative of the reference angles (acceleration) is inserted into the inverse dynamics simulator.

troller, a motor, and a reduction gear [4,25]. A well-known smart actuator is the Dynamixel AX-12, which is modeled in [37]. Also, models of the actuator and the driver are widely used, not only in the field of robotics but also in many other fields. In humanoid robots, brushless DC motors and drivers are mostly used because of their high durability. Respective models can be found in [6] for example.

In some cases, the trajectories of the joint angles are given by the robot controller and the required joint torques are computed by inverse dynamics–based simulation. In such cases, the user obtains information about the required torque and the postural stability of the robot, based on the assumption that all actuators track perfectly the reference trajectories. Fig. 8.6 shows an example of a joint-servo controller for inverse dynamics–based simulation. The robot controller computes the reference joint angles and supplies the second derivatives of the reference angles to the dynamics simulator, as shown in Fig. 8.6. In Section 8.4, it is explained how to conduct this type of simulation using MATLAB/Simulink.

Dynamics-based simulation: In inverse dynamics–based simulation, the joint acceleration is the input and the joint torque is the output. On the other hand, in forward dynamics–based simulation, the joint torque is the input and the joint acceleration is the output. The details of dynamics-based simulation are explained in Chapter 4. Generally, the complexity of computing the joint and base acceleration from (4.1) is $O(N^3)$, where N denotes the DoF in the forward dynamics–based simulation [15]. In order to accelerate the development of the controller, the robot simulation needs to be performed in a short time. Several high-speed computation methods have been developed. For example, forward dynamics based on propagation methods have computational complexity of $O(N)$ [15]. By utilizing parallel computing, $O(\log(N))$ algorithms have been developed [14,70].

By solving the equation of motion considering the contact force, accelerations in the joints and the base of the robot can be obtained. By integrating the joint and base accelerations numerically, joint velocities, angles, and the whole-body motion can be obtained.

Visualization: Visualization is an important part of the robot simulator. A humanoid robot usually comprises a number of joints and links. The numerous plots of their states are not easy to understand. Three-dimensional computer graphics can help the user to know what will happen when executing the designed motion. For the display of computer graphics, there are useful libraries. OpenGL [46], for example, is one of the graphics APIs for rendering, texture mapping, etc. Being a cross-language and a cross-platform, it is widely used in robot simulators. OpenGL is a low-level library that is usually assessed via toolkits compatible with Open Inventor [69], such as GLUT [20], GLFW [19] or Coin3D [9].

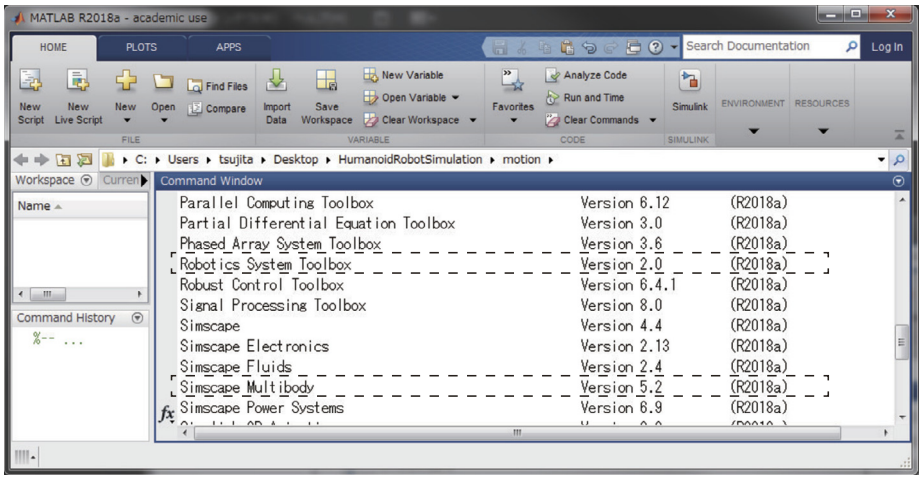


FIGURE 8.7 An example of a list with licensed components. This figure shows the output in the command window of MATLAB, generated by the "ver" command. The components in the dotted rectangles should be displayed.

8.4 DYNAMICS SIMULATION USING MATLAB/SIMULINK

In this section, it is explained how to perform dynamics-based simulation using MATLAB/Simulink with Simscape Multibody and the Robotics System Toolbox. Simscape Multibody and the Robotics System Toolbox are used to perform dynamics-based simulation and to generate the motion of the humanoid robot, respectively. To check whether Simscape Multibody and the Robotics System Toolbox are available, enter the following command in the MATLAB command window:

```
>>ver
```

As a result, the components shown in Fig. 8.7 should be displayed.

8.4.1 Generating a Robot Tree Model for Simulink

In general, it is possible to describe the structure of a robot with Simscape alone. Since the structure of a humanoid robot is quite complex, this would be quite a tedious task, though. It is much more preferable to make use of the robot CAD files or the Unified Robot Description Format (URDF) file. In what follows, it will be assumed that either is available. In the case the robot is designed by the user, the following subsection will be helpful. In the case a commercial robot that can be used with ROS is at hand, the subsection named "Using a URDF file" will be helpful.

Using CAD Files

Simscape Multibody can import the structure of the robot from its CAD files via Simscape Multibody Link [58]. At the time being, Simscape Multibody Link supports typical mid-range CAD software packages like Solidworks (Dassault Systemes SolidWorks Corp.), Creo (PTC),

and Autodesk Inventor (Autodesk Inc.). Simscape Multibody can import XML (eXtensible Markup Language) files that contain the structure of a robot. In order to export an XML file from the CAD files, the Simscape Multibody Link plug-in for the respective CAD software package must be downloaded and installed. With an appropriate maintenance contract for Simscape Multibody, it should be possible to download the plug-in from the Mathworks site [58]. The method of installing the plug-in for each CAD software package is explained in the documentation of the Simscape Multibody Link plug-in [29]. Here, a brief explanation is given for the following environment.

Operating system	**Windows 7 x64 professional**
CAD software	SolidWorks 2016 x64
Simulation software	MATLAB/Simulink R2018a x64

Step 1. Download [58] and save the archived file smlink.r2018a.win64.zip and the MATLAB file install_addon.m in a convenient folder. If the version of MATLAB is different, read the name of the ZIP file as appropriate.

Step 2. Right-click on the MATLAB icon and run with administrator privileges.

Step 3. In the command window of MATLAB, add the folder containing the downloaded ZIP file to the MATLAB path with the "addpath" command and install the add-on with the "install_addon" command:

```
>>addpath('D:\tmp')
>>install_addon('smlink.r2018a.win64.zip')
```

D:\tmp is the folder which contains the ZIP file.

Step 4. In order to connect via the Simscape Multibody Link plug-in, MATLAB must be registered as an automation server. In the command window of MATLAB, type the following command:

```
>>regmatlabserver
```

Step 5. To enable the Simscape Multibody Link plug-in for SolidWorks, enter the following command in the command window:

```
>>smlink_linksw
```

After successfully registering the plug-in, start SolidWorks.

Step 6. Open a dialog box by selecting the "Add-Ins" in the tools menu of SolidWorks. Select the "Simscape Multibody Link" check box, as shown in Fig. 8.8, and click on the "OK" button.

The abovementioned procedure completes the installation of the Simscape Multibody Link plug-in. The plug-in is ready now to export the robot model from SolidWorks.

Next, it is explained how to assemble a robot with SolidWorks. In the SolidWorks assembly file, set a mate entity so that it is recognized as a joint in Simscape. An example of a hinge joint between two links is shown in Fig. 8.9. In the figure, the "Concentric" mate is set around the axis of the hinge, and the "Coincident" mate is set on the front surface of Link 1 and the back surface of Link 2. After these settings, Link 2 will be able to rotate freely in the SolidWorks assembly window. Also, the joint will be recognized in Simscape Multibody Link. Repeat this

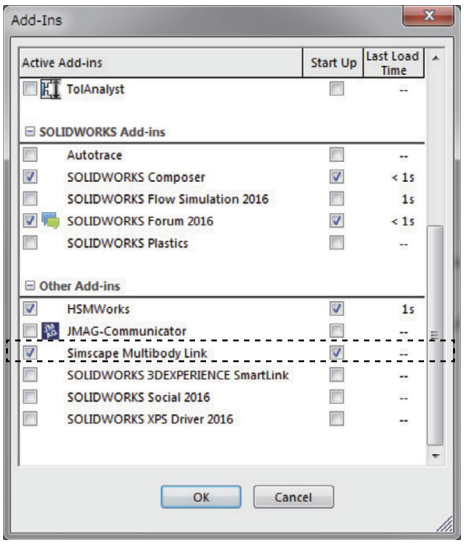


FIGURE 8.8 An example of the "Add-Ins" dialog box in SolidWorks. In order to be able to communicate with Simulink, check the "Simscape Multibody Link" box in the dotted rectangle.

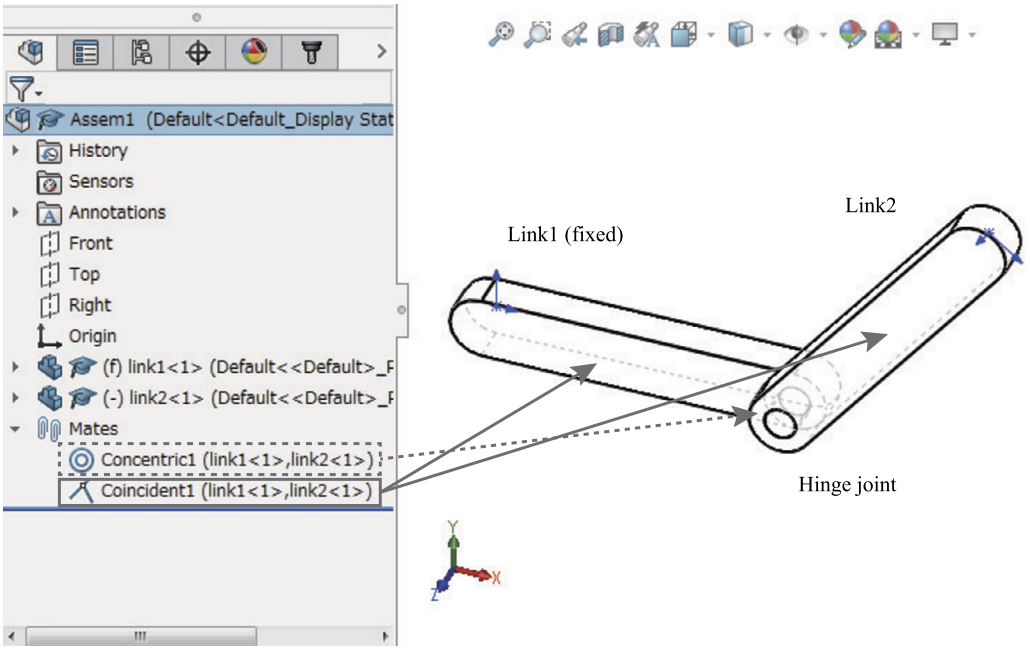


FIGURE 8.9 Example of a hinge joint setting in SolidWorks. In order to recognize the two parts as a hinge joint in Simscape, the "Concentric1" mate is set around the axis of the hinge, and the "Coincident1" mate is set on the front surface of Link 1 and the back surface of Link 2.

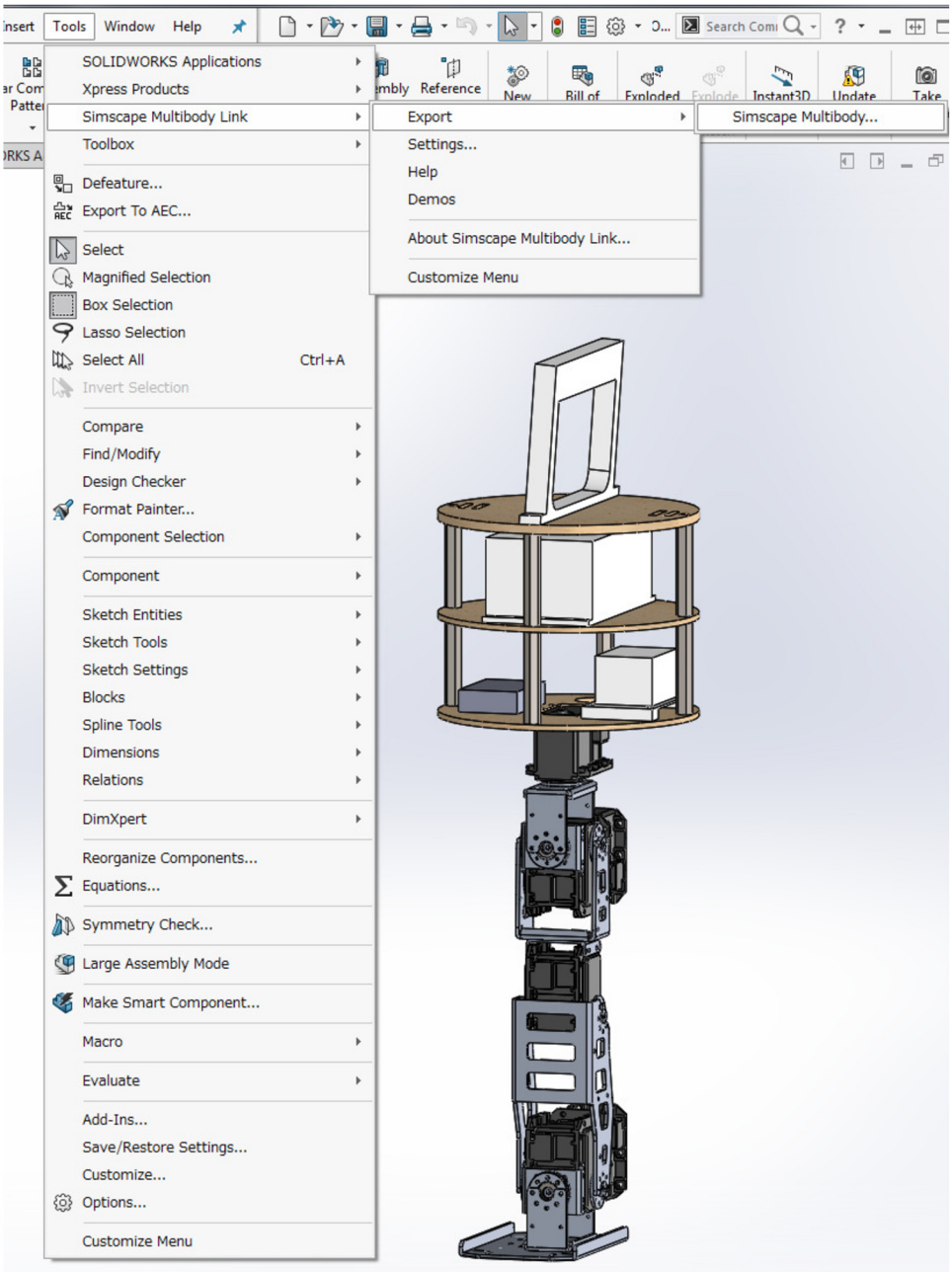


FIGURE 8.10 Design of a one-legged robot [61] using SolidWorks. The materials of all parts of this robot are specified for configuring the mass, the CoM, and the inertia.

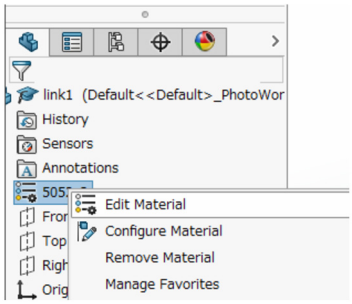
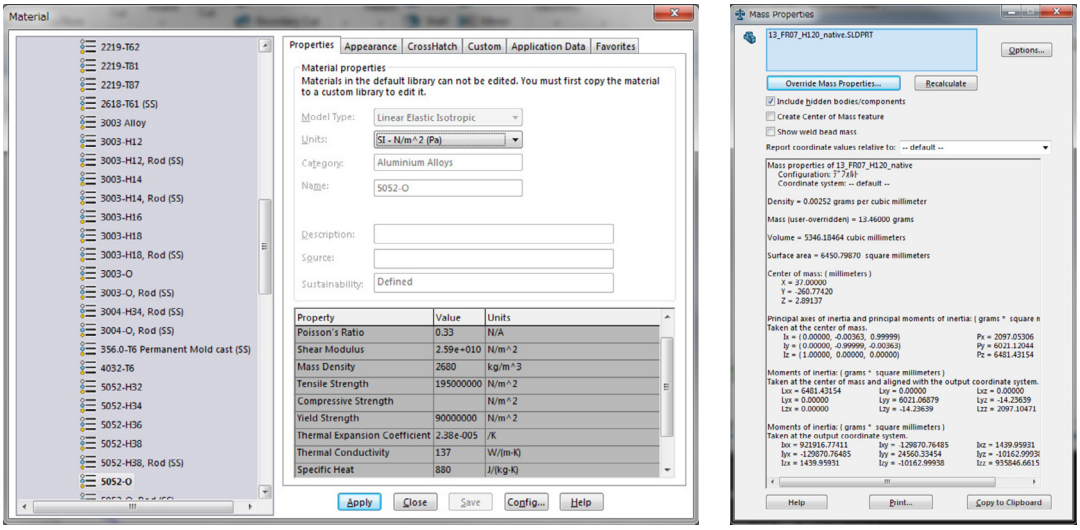


FIGURE 8.11 Menu for editing the material in the "FutureManager Design Tree." By selecting the "Edit Material," the "Material setting" window (Fig. 8.12) pops up.



(A) (B)

FIGURE 8.12 Setting the mass properties of each rigid-body part. (A) Left: The "Material" window for specifying the material of a rigid-body part. The material can be chosen from a list. (B) Right: The "Mass Properties" window shows the CoM, the principal axes of inertia, the principal moments of inertia, and the moments of inertia of a rigid-body part. Note that there are other components, such as actuators, whose density is not constant. In this case, override the mass properties by clicking on the "Override Mass Properties" button.

procedure for each humanoid robot joint and set all joints to rotate freely. Note that, in the case the robot comprises other types of joints, it is necessary to set the mates according to the respective joint type [39].

Fig. 8.10 shows an example of a robot design [61]. To determine the mass, center of gravity position, and inertia tensor of a given part, it is necessary to select the "Edit Material" in the "FutureManager Design Tree" at the left end of the window, as shown in Fig. 8.11. The material density of each rigid-body part should be specified as close as possible to the actual material density (cf. Fig. 8.12A). Note that there are other components, such as actuators,

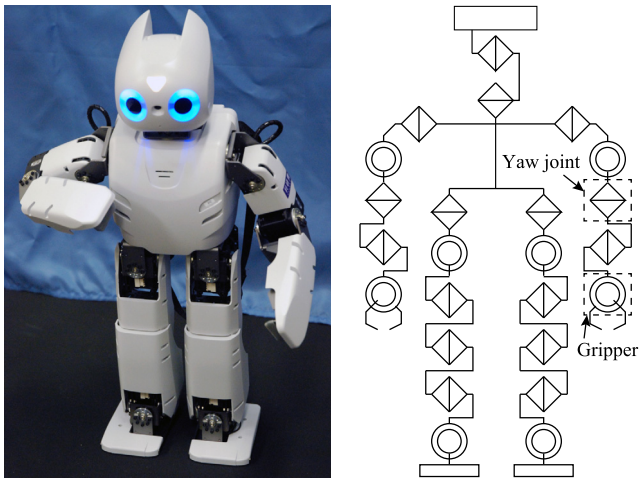


FIGURE 8.13 Small-size humanoid robot ROBOTIS-OP2. Its URDF file, used in this example, is available for free. (A) Left: overview of ROBOTIS-OP2 (basic arm model) having the same mechanics as DARwIn-OP [25]. (B) Right: DoFs of ROBOTIS-OP2 including yaw joints and grippers. The URDF files are written based on this structure. The appended yaw joint and gripper are in dotted rectangles.

whose density is not constant. In this case, click on the "Override Mass Properties" button in the "Mass Properties" window (cf. Fig. 8.12B) and specify the value. In order to check the mass properties, click on the "Mass Properties" button which is available from the "Evaluate" tab in the "CommandManager" tool bar. As shown in Fig. 8.12B, the "Mass Properties" window pops up and shows the center of mass, the principal axes of the inertia, the principal moments of inertia, and the moments of inertia of the rigid-body part. With this setting, the mass properties will be sent to the Simscape tree to be described later.

The next step is to generate the XML file and the STEP files via the Simscape Multibody Link plug-in for importing the structure and the shape of the designed robot. To this end, open Simscape Multibody Link by selecting the "Simscape Multibody Link" → the "Export" → the "Simscape Multibody" from the "Tools" menu in the menu bar of the SolidWorks assembly window, as shown in Fig. 8.10, and specify a name for saving. The saved XML and STEP files describe the structure of the robot and the shape of the links, respectively. The robot model is now ready to be loaded with Simscape Multibody.

Using the URDF File

Some robots comprise a URDF file that is written in XML. Simscape can parse the structure and mass properties of a robot described by the respective URDF file. The URDF files of commonly used robots are readily available from the ROS package.

In this section, the robot model of ROBOTIS-OP2 is used as an example. ROBOTIS-OP2 is a small-size humanoid robot (cf. Fig. 8.13A) having the same mechanics as DARwIn-OP [25],

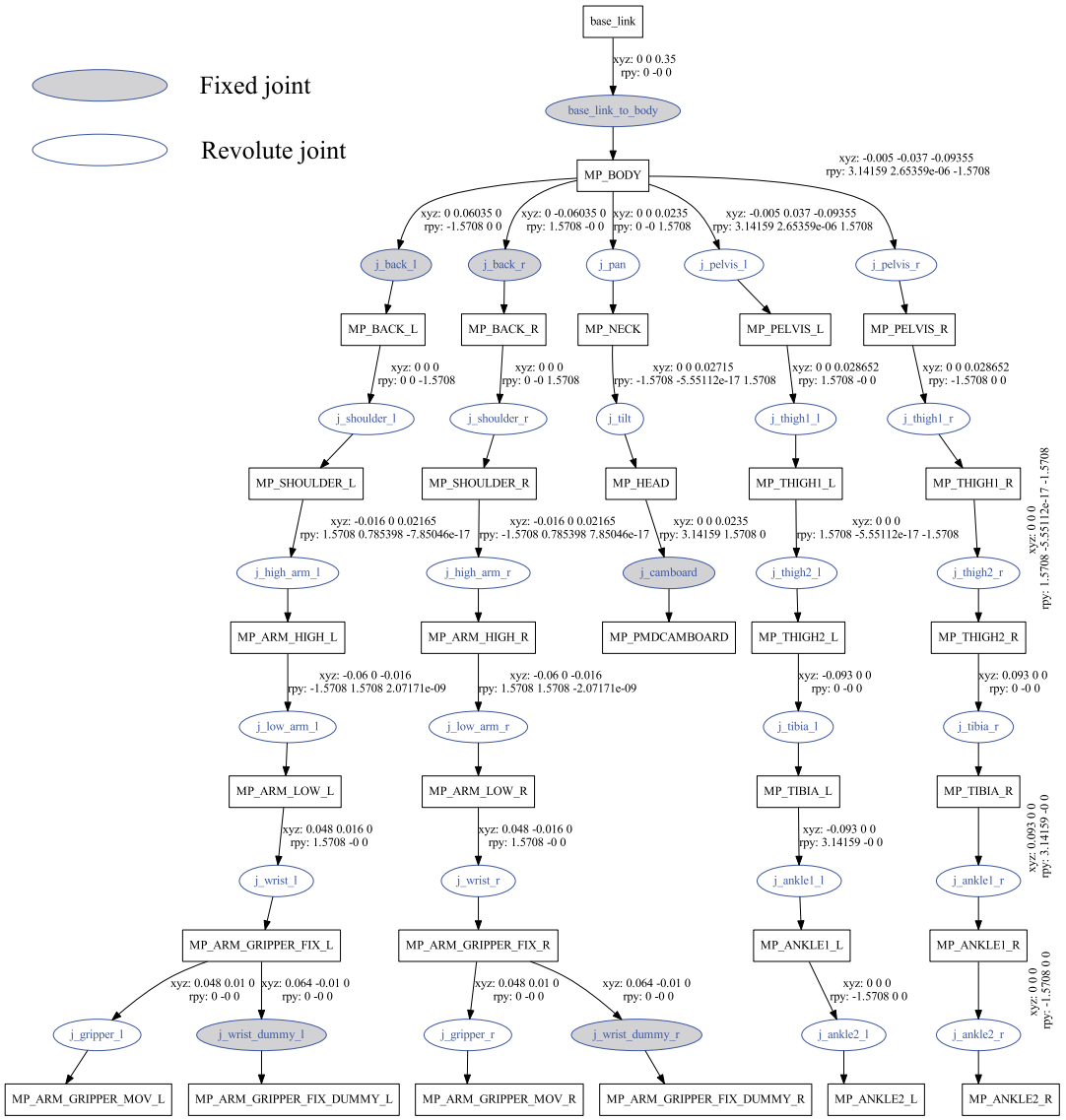


FIGURE 8.14 Structure of ROBOTIS-OP2. Fixed and revolute joints are denoted by gray and white ovals, respectively. A body that forms a joint is denoted by a white rectangle. An arrow expresses the connection between a joint and the body. The numerical value next to the arrow indicates the relationship for translation and orientation between the joint and the body.

with an enhanced embedded computer. Fig. 8.13B shows the DoFs of ROBOTIS-OP2, including yaw joints and grippers. The URDF files distributed on GitHub [63] are written based on this structure. Fig. 8.14 shows the connection between the links and the joints based on

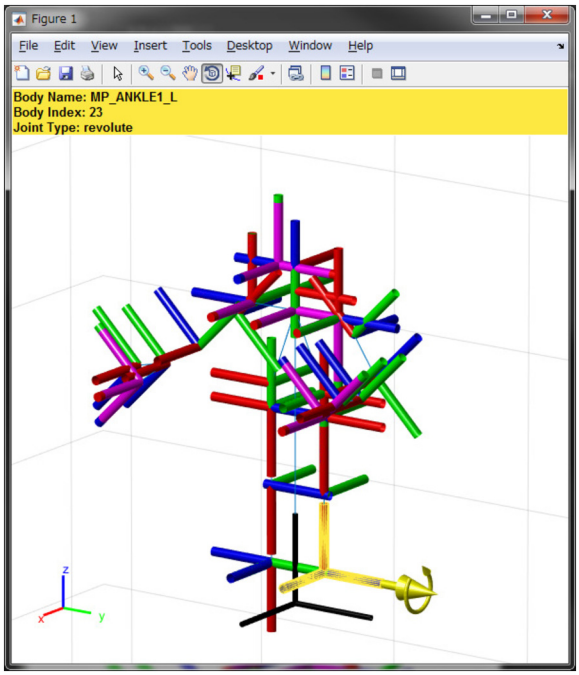


FIGURE 8.15 Visualization of the frames of ROBOTIS-OP2. By using the Robotics System Toolbox, the defined frames in the URDF files can be visualized as shown in this figure. Fixed joint frames are shown in magenta. Movable joint frames have three colors. The red, green, and blue cylinders denote the x-, y-, and z-axes, respectively. When selecting a movable joint frame, the frame is highlighted in yellow and the definition of the rotation axis is expressed by an arrow. The name of the frame is displayed in the upper part of the window.

the URDF file. Fixed and revolute joints are denoted by gray and white ovals, respectively. A body which is connected to a joint is denoted by a white rectangle. An arrow expresses the connection between a joint and the body. The numerical value next to the arrow indicates the relationship of translation/orientation between the joint and the body.

The joint frames of the robot can be visualized as shown in Fig. 8.15, by typing the following command (included in the Robotics System Toolbox):

```
>>cd 'location_of_URDF_files';
>>op2=importrobot('robotis_op.urdf');
>>op2.show('visuals','off')
```

where the "importrobot" command imports the rigid-body tree from the URDF file.

The colors of the x-, y-, and z-axes are according to the RGB convention; the fixed joints are shown in magenta. When selecting a movable joint frame, the frame is highlighted in yellow and the definition of the rotation axis is expressed by an arrow. The name of the frame is displayed in the upper part of the window.

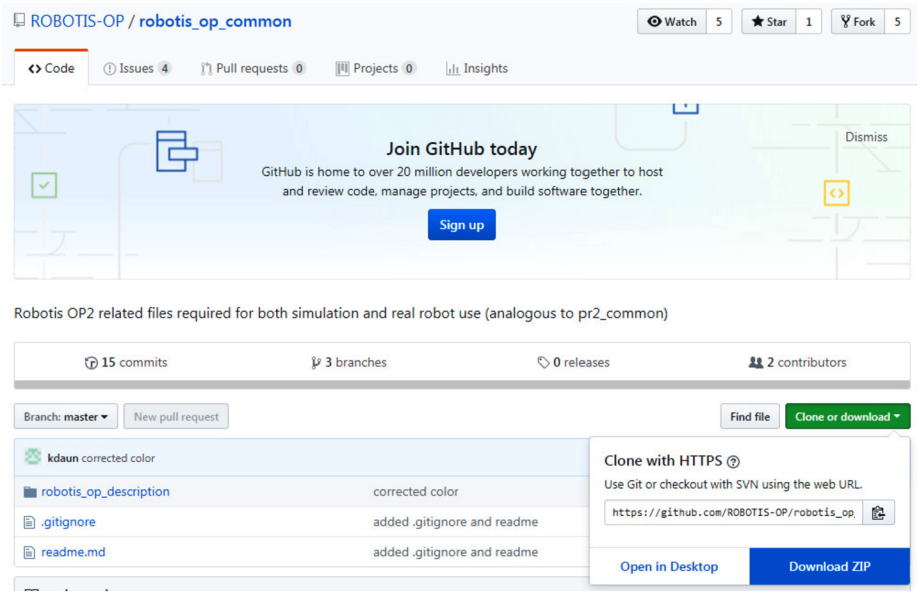


FIGURE 8.16 A URDF file and mesh files distributed on GitHub [63]. Click on the "Clone or download" button, and then click on the "Download ZIP." The URDF file and the mesh files can be obtained by extracting the downloaded file.

8.4.2 Generating the Simulink Model

In the case of an XML file, exported through the Simscape Multibody Link plug-in and imported into Simscape, type the following commands in the MATLAB window:

```
>>cd 'location_of_exported_files'
>>smimport('robot.xml')
```

In the case of a URDF file imported into Simscape, type the following commands:

```
>>cd 'location_of_URDF_file'
>>smimport('robot.urdf')
```

The details of using the URDF file of ROBOTIS-OP2 are explained below. From GitHub [63], download the ZIP file containing the URDF file that includes the robot's structure and mass properties and the mesh file that describes the appearance of each part. Click on the "Clone or download" button; then click on the "Download ZIP" as shown in Fig. 8.16 and extract the downloaded file. Place the URDF file and the STL files on the folder tree, as shown in Fig. 8.17. In order to be able to read the mesh file in Mechanics Explorer (the viewer of Simscape), set the path to "D:\MATLAB\HRS\RobotModels\robotis_op_description\meshes\" as in this example. The "Set Path" button is available from the "HOME" tab in the MATLAB main window. Next, type the following commands in the MATLAB window:

```
>>cd D:\MATLAB\HRS\RobotModels\robotis_op_description\robots\
>>smimport('robotis_op.urdf')
```

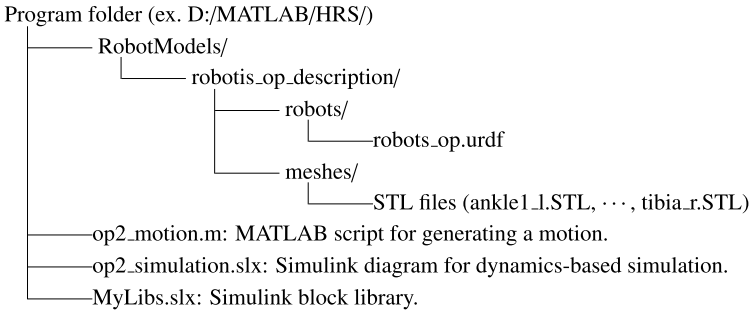
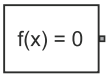
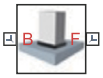
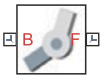


FIGURE 8.17 Example of a folder tree. Place the URDF file and the STL files downloaded from GitHub [63] on this folder tree. The RobotModels folder contains specific files for the robot. The MATLAB script file named op2_motion.m is a program file for generation of motion (cf. Section 8.4.6). The op2_simulation.slx and MyLibs.slx are simulink diagrams for dynamics-based simulation (cf. Section 8.4.2).

TABLE 8.1 Definition of the Simscape blocks used in the Simscape tree shown in Fig. 8.18. These blocks describe the mechanical properties and the simulation configuration

	World frame	This block represents the inertial reference frame.
	Mechanism configuration	This block sets mechanical and simulation parameters that apply to an entire machine.
	Solver configuration	This block defines the solver settings to use for simulation.
	Weld joint	This block represents a fixed joint between two frames.
	Revolute joint	This block represents a revolute joint acting between two frames.
	6-DOF joint	This block represents a 6-DoF joint acting between two frames.

The structure of ROBOTIS-OP2 imported into Simscape is shown in Fig. 8.18. The model tree is divided into right leg, left leg, neck, right hand, and left hand. The areas surrounded by a square box are grouped by the "area" button at the left end of the Simulink editor. The definition of each block is described in Table 8.1.

It can be seen that the base link (the "base_link") is fixed to the world frame (the inertial frame) and that five branches originate at the torso in Fig. 8.18. The upper two branches are the lower limbs, the central branch is the head, and the lower two branches are the upper

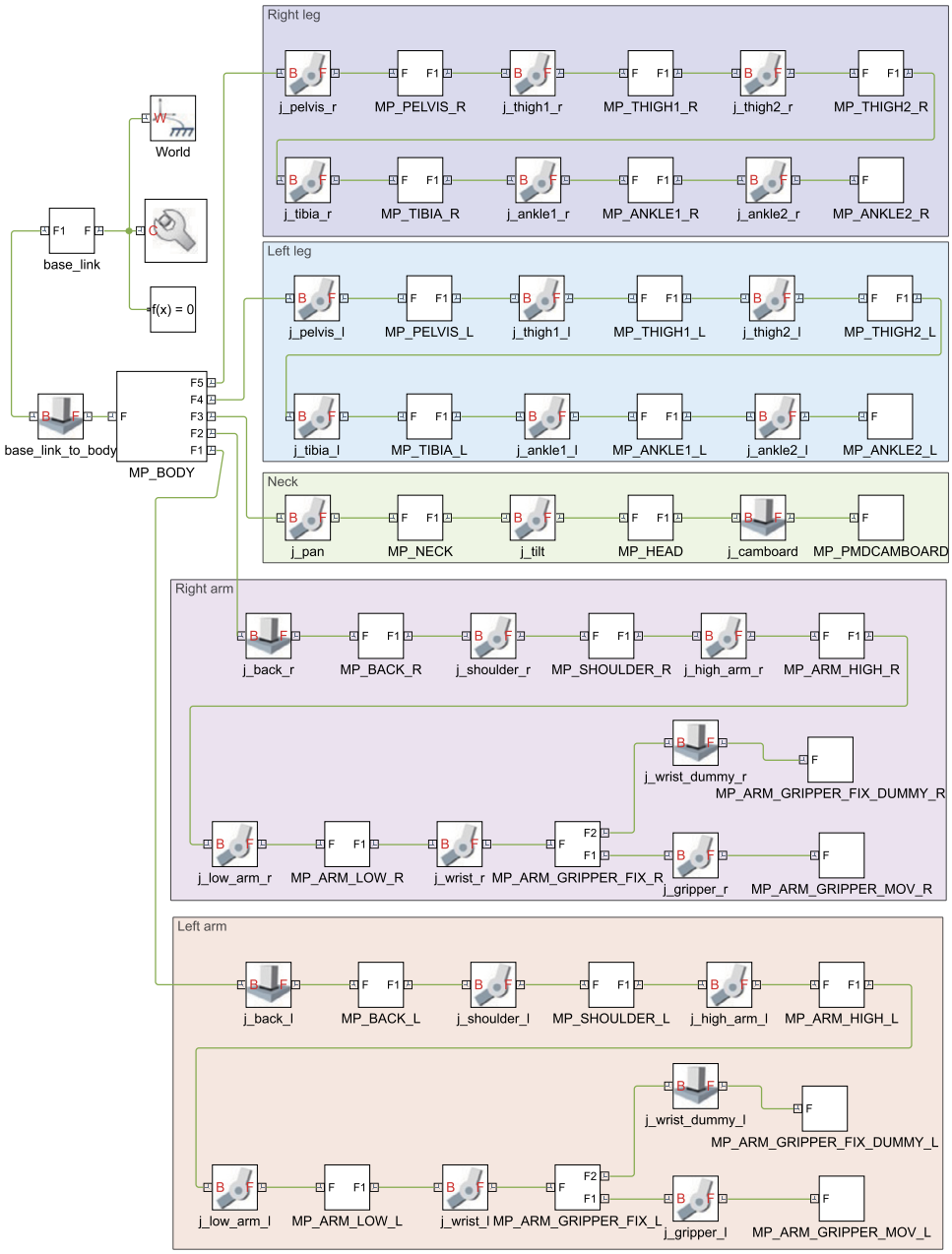


FIGURE 8.18 The Simscape tree of ROBOTIS-OP2. The model tree is divided into right leg, left leg, neck, right hand, and left hand. The areas surrounded by a square box are grouped by the "area" button at the left side of the Simulink editor.

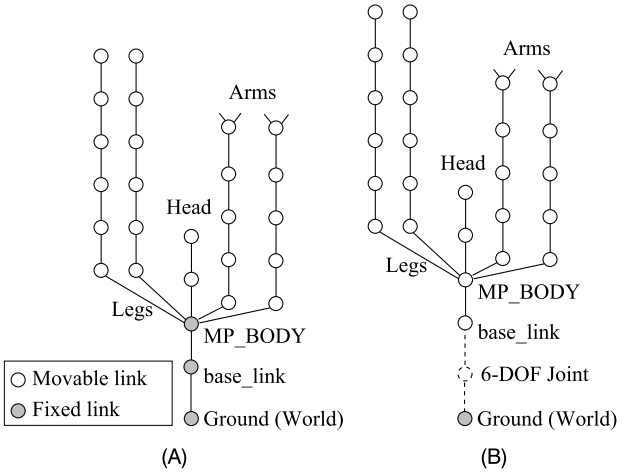


FIGURE 8.19 Tree connectivity structure of the kinematic chain of ROBOTIS-OP2. (A) Left: The original tree connectivity structure. The link named base_link is fixed on the ground. (B) Right: A free joint of six DoFs is attached between the base_link and the ground. This modification changes the fixed model to a free-floating model.

limbs. Fig. 8.19A shows the tree connectivity structure of the Simscape tree. In this state, since the base link is fixed to the world frame, a free joint of six DoFs (a rigid-body joint) is attached between the "base_link" block and the "World" block, as shown in Fig. 8.19B. In this way, the torso can move freely w.r.t. the world frame.

By double clicking on the icon of the wrench mark and opening the property of "Mechanical Configuration," confirm the value of "Gravity." This value should be [0 0 −9.8] as is often used in the field of humanoid robots (cf. Fig. 8.21).

8.4.3 Joint Mode Configuration

The default state after import is such that the input mode of each joint is unspecified. Double click on the joint block and specify the joint input mode as shown in Fig. 8.22. In order to calculate the joint torque from the given angular acceleration of a joint using the inverse dynamics, users need to configure the "Actuation" item in Fig. 8.22. Set the "Automatically Computed" in the "Torque" configuration of the "Actuation" item as shown in Fig. 8.22A. In addition, set the "Provided by Input" in the "Motion" configuration. In order to output the computed torque from this block, check the "Actuator Torque" box in the "Sensing" item.

On the other hand, in order to calculate the joint acceleration from the given torque of a joint by the forward dynamics, set the "Provided by Input" in the "Torque" configuration of the "Actuation" item as shown in Fig. 8.22B. In addition, set the "Automatically Computed" in the "Motion" configuration. In order to output the computed joint acceleration from this block, check the "Acceleration" box in the "Sensing" item. If the angular velocity and/or the angle are needed (calculated by integrating the computed acceleration), also check the "Velocity" box and/or the "Position" box in the "Sensing" item.

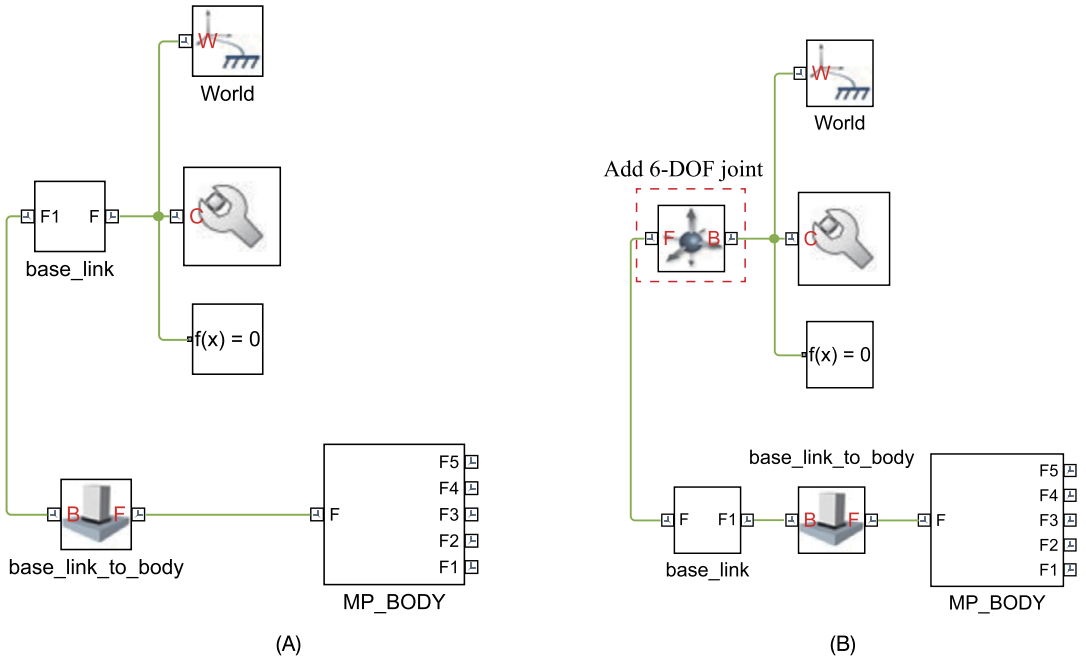


FIGURE 8.20 (A) Left: The original diagram of the Simscape tree between the "World" block and the "MP_BODY" block. Since the "base_link" block is connected directly to the "World" block, the "base_link" is fixed. (B) Right: A free joint block named "6-DOF joint" is attached between the "base_link" block and the "World" block.

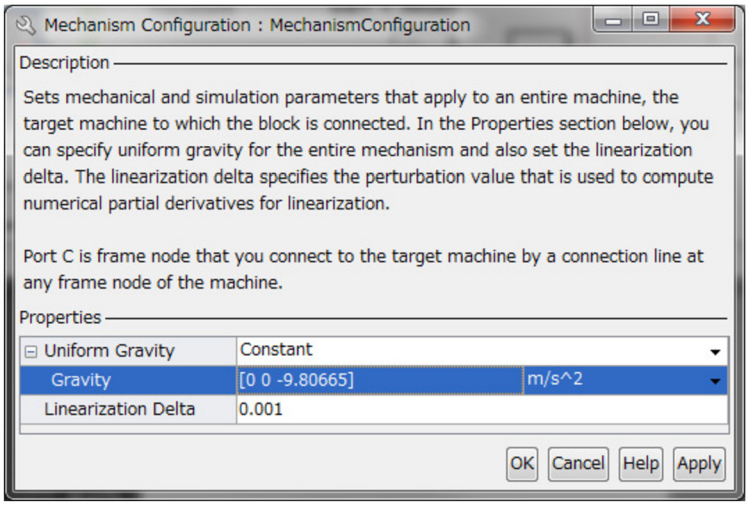


FIGURE 8.21 The "Mechanism Configuration" window. By double clicking on the icon of the wrench mark in the Simscape diagram shown in Fig. 8.20, this window pops up and confirms the value of "Gravity." This value should be $[0\ 0\ -9.8]$, as shown in this figure.

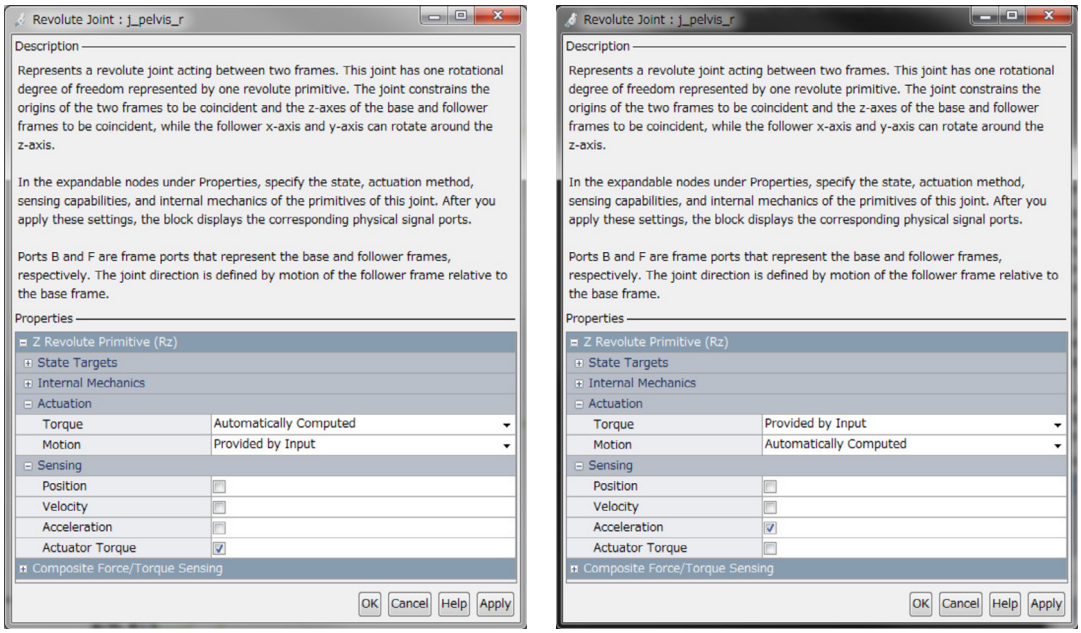


FIGURE 8.22 Joint configuration for a revolute joint. (A) Left: In order to calculate the joint torque from the given angular acceleration to the joint using the inverse dynamics, users need to configure the "Actuation" item in this property window. Set the "Automatically Computed" in the "Torque" configuration of the "Actuation" item as shown in this figure. In addition, set the "Provided by Input" in the "Motion" configuration. In order to output the computed torque from this block, check the "Actuator Torque" box in the "Sensing" item. (B) Right: In order to calculate the joint acceleration from the given torque of a joint by the forward dynamics, set the "Provided by Input" in the "Torque" configuration of the "Actuation" item as shown in this figure. In addition, set the "Automatically Computed" in the "Motion" configuration. In order to output the computed joint acceleration from this block, check the "Acceleration" box in the "Sensing" item.

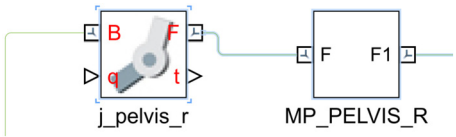


FIGURE 8.23 When the joint is configured for calculation using the inverse dynamics, the input port "q" and the output port "t" appear in the block. The input port is connected to the desired trajectory signal.

When the joint is configured for calculation using the inverse dynamics, the input port "q" and the output port "t" appear in the block as shown in Fig. 8.23. The input port is connected to the desired trajectory signal of the follower frame w.r.t. the base frame along the joint primitive axis.

Fig. 8.24 shows an example of how to calculate the torque using a sinusoidal trajectory as the target trajectory in a joint. The "Sine Wave" block that generates the desired trajectory of the joint is a standard source block in Simulink. However, it cannot be directly connected to

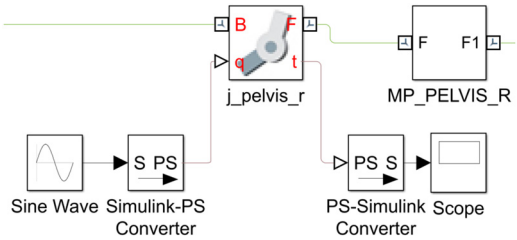
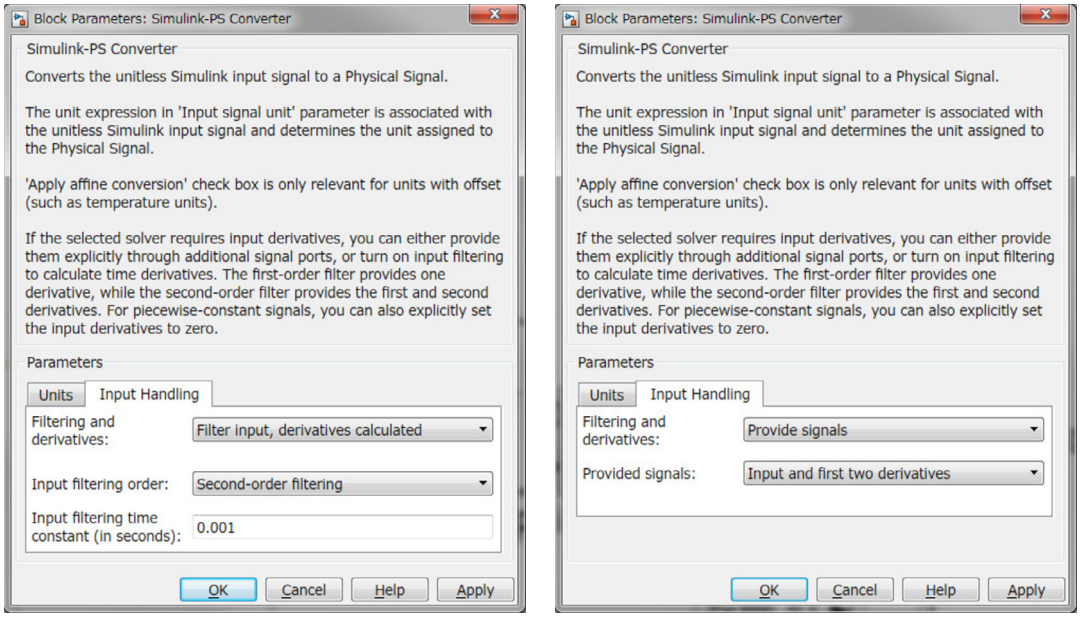


FIGURE 8.24 An example of desired trajectory input. The "Sine Wave" block that generates the desired trajectory of the joint is a standard source block in Simulink, but it cannot be directly connected to the Simscape block. Therefore, the source block is connected via the "Simulink-PS Converter" block to the signal input "q" of the joint.



(A) (B)

FIGURE 8.25 By double clicking on the "Simulink-PS Converter" block, the calculation method for angular acceleration can be configured. (A) Left: By setting the "Filtering and derivatives:" item to the "Filter input, derivatives calculated" and the "Input filtering order" item to the "Second-order filtering," the input trajectory is filtered and its second derivative is calculated in the "Simulink-PS Converter" block for the simulation using the inverse dynamics. (B) Right: Alternatively, the calculation method for acceleration can be specified by setting the "Filtering and derivatives" item to the "Provide signals."

the Simscape block. Therefore, the output of the source block is connected via the "Simulink-PS Converter" block to the "q" input of the joint.

In order to calculate the joint torque using the inverse dynamics, it is also necessary to obtain the angular acceleration from the signal input "q." The calculation method of the angular acceleration can be configured by double clicking on the "Simulink-PS Converter" block as shown in Fig. 8.25. As shown in Fig. 8.25A, by setting the "Filtering and derivatives:" item to

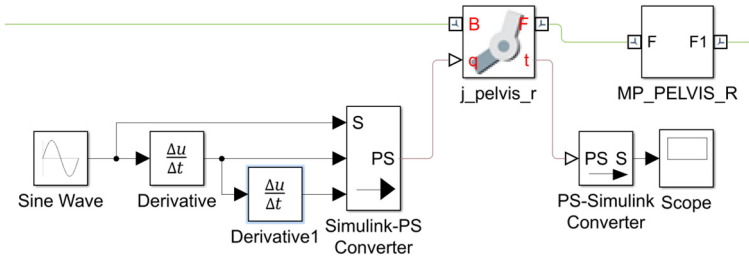
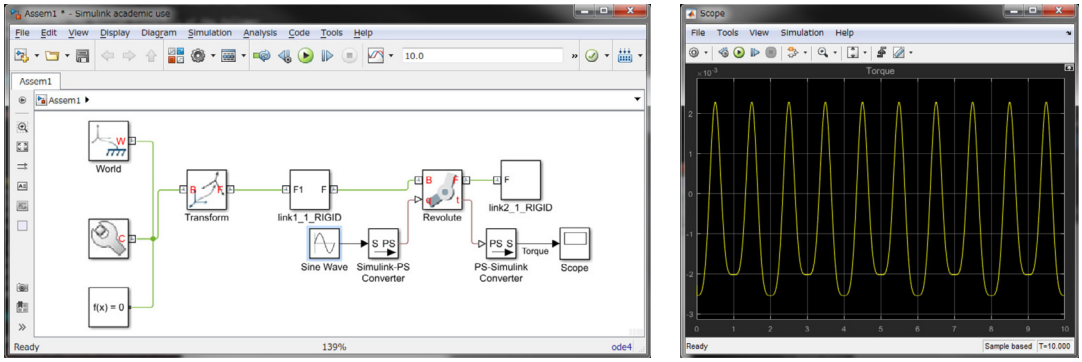


FIGURE 8.26 An example of a user-defined calculation method for angular acceleration. The angular acceleration and velocity are derived by using the two "Derivative" blocks (included in the Simulink standard blocks).



(A) (B)

FIGURE 8.27 Inverse dynamics–based simulation of the two-link arm. (A) Left: The Simulink window for editing the diagram and starting the simulation. The simulation is initialized by clicking on the play icon (▷) in this window. (B) Right: The scope window, displaying the computed joint torque, is connected to the output port of the joint block via the "PS-Simulink Converter" block.

the "Filter input, derivatives calculated" and the "Input filtering order" item to the "Second-order filtering", the input trajectory is filtered and its second derivative is calculated in the "Simulink-PS Converter" block. Alternatively, the calculation method for acceleration can be specified by setting the "Filtering and derivatives" item to the "Provide signals" as shown in Fig. 8.25B. When an item in the "Provide signals" is set to "Input and first two derivatives," inputs for the first-order and second-order differentiation appear in the input signal of the "Simulink-PS converter" block. By doing this, the calculation method of acceleration can be set voluntarily as shown in Fig. 8.26. In addition, it is also possible to input the speed and position into the "Simulink-PS Converter" block, after motion generation in terms of acceleration and then integrating. To observe or handle the computed torque for the input acceleration from Simulink, the "PS-Simulink Converter" block is used. In the example shown in Fig. 8.26, the computed torque is input into the "Scope" block. On the other hand, to input torque, simply convert it with the "Simulink-PS Converter" block without differentiation.

Fig. 8.27 shows an example of inverse dynamics calculation of the two link arm shown in Fig. 8.9. Here, simulation is performed assuming that Link 1 is fixed. By the "smim-

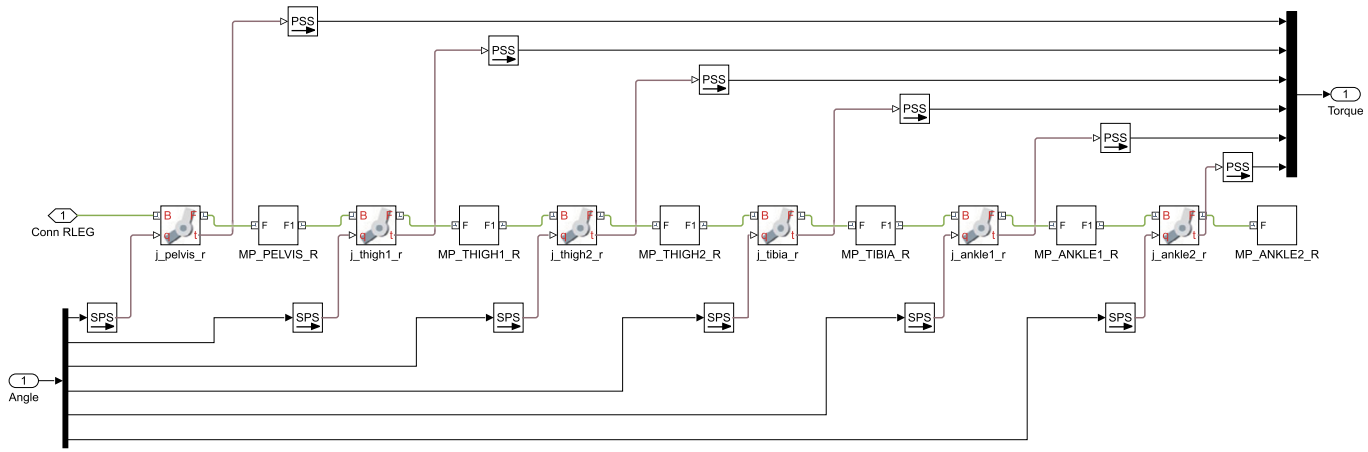


FIGURE 8.28 Right leg tree for the inverse dynamics calculation. This subsystem is the "Right leg" area shown in Fig. 8.18. It can be created by a right click on the area and selecting the "Create Subsystem from Area" as shown in Fig. 8.29. This subsystem uses a reference joint angle vector as input and generates a computed torque vector.

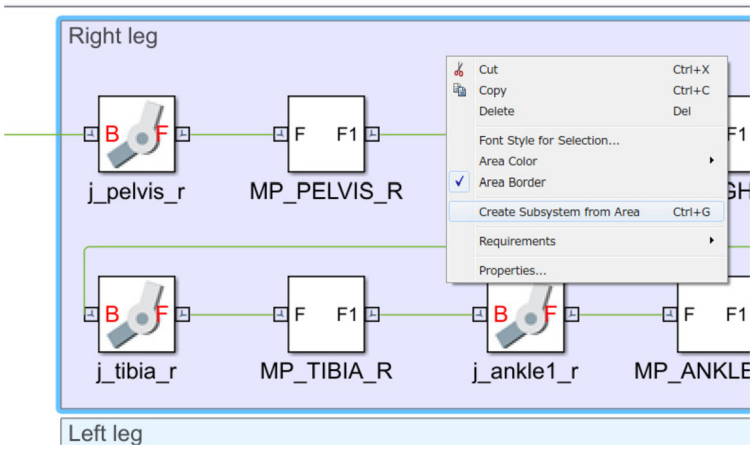


FIGURE 8.29 Creating a subsystem from an area. A subsystem can be created by a right click on the area and selecting "Create Subsystem from Area."

port" command, the "Revolute" joint block is generated between the "link1_1_RIGID" and "link2_1_RIGID" blocks, and a desired joint trajectory is specified for the joint block with a sinusoidal waveform as shown in Fig. 8.9A. The "Scope" block is connected to the output port of the joint block via the "PS-Simulink Converter" block. By clicking on the play icon (▷) in the Simulink window, the simulation starts and the computed torque is plotted as shown in Fig. 8.9B.

Inverse dynamics–based simulation of a whole-body motion of a humanoid robot can be performed by setting the abovementioned procedures for all joints. Fig. 8.28 is a subsystem created from the "Right leg" area shown in Fig. 8.18. It represents an example for inverse dynamics calculation. This subsystem can be created by a right click on the area and selecting "Create Subsystem from Area" as shown in Fig. 8.29. This subsystem uses a reference joint angle vector as an input and generates a computed torque vector.

In order to increase the readability of the diagram, it is helpful to create subsystems for all areas as shown in Fig. 8.30. All subsystems have an input port for the reference joint angle vector and an output port for the computed torque vector. In this example, the reference joint angle vector will be generated by the MATLAB script and saved as a workspace variable. The variable is then imported into Simulink by using the "From Workspace" block, as shown in Fig. 8.31. In order to prevent unintended movement, set the "Form output after final data value by" item to "Holding final value" as shown in Fig. 8.32. The variable is a time-series object comprising information for the time and the 24 joint angles. Since the number of inputs of the arms, the neck, and the legs are five, two, and six, respectively, the composite joint angle vector is decomposed by the "Demux" block, by setting its parameters as shown in Fig. 8.33. In order to observe the computed torque at all joints, the outputs of the subsystems are connected to the "Scope" block (included in the Simulink standard blocks).

The minimum setting to perform the inverse dynamics simulation is finalized by the abovementioned procedure. However, there will be no reaction force from the floor. Thus,

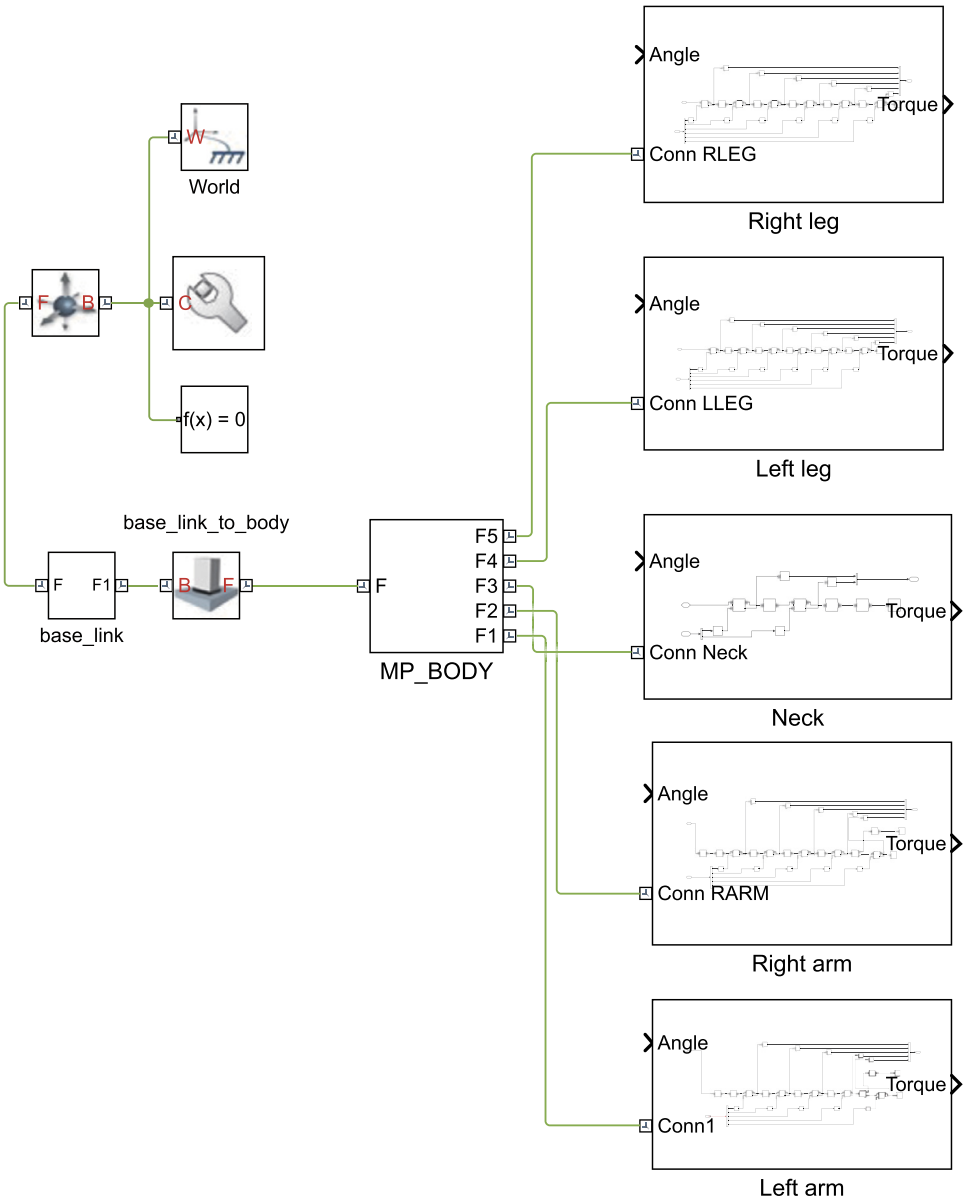


FIGURE 8.30 A Simscape tree with five subsystems. In order to increase the readability of the complicated diagram of the humanoid robot, it is helpful to create subsystems for the branches of the kinematic chain. All subsystems have an input port for the reference joint angle vector and an output port for the computed torque vector.

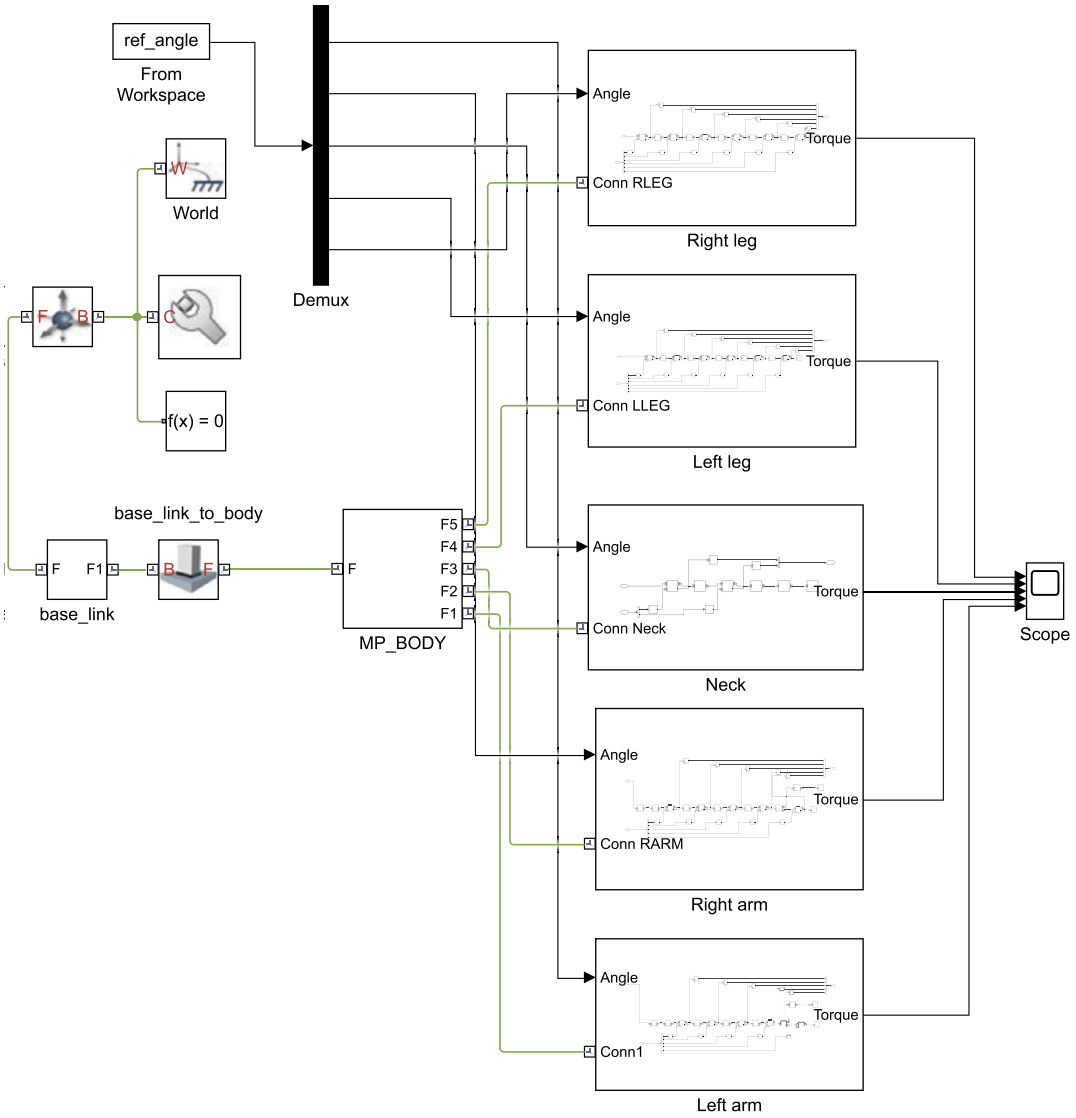


FIGURE 8.31 The joint reference is an input from the MATLAB workspace. The reference joint angle vector will be generated by the MATLAB script and saved as a workspace variable. The variable is then imported into Simulink by using the "From Workspace" block. The variable is a time-series object comprising information for the time and all of the 24 joint angles. The composite joint angle vector is then decomposed by the "Demux" block.

when the simulation is initialized, the robot will fall down. In order to check the settings so far, set "Gravity" to zero. This item can be configured by double clicking on the icon of the wrench mark in Fig. 8.31. Then try a simulation with reference joint angles.

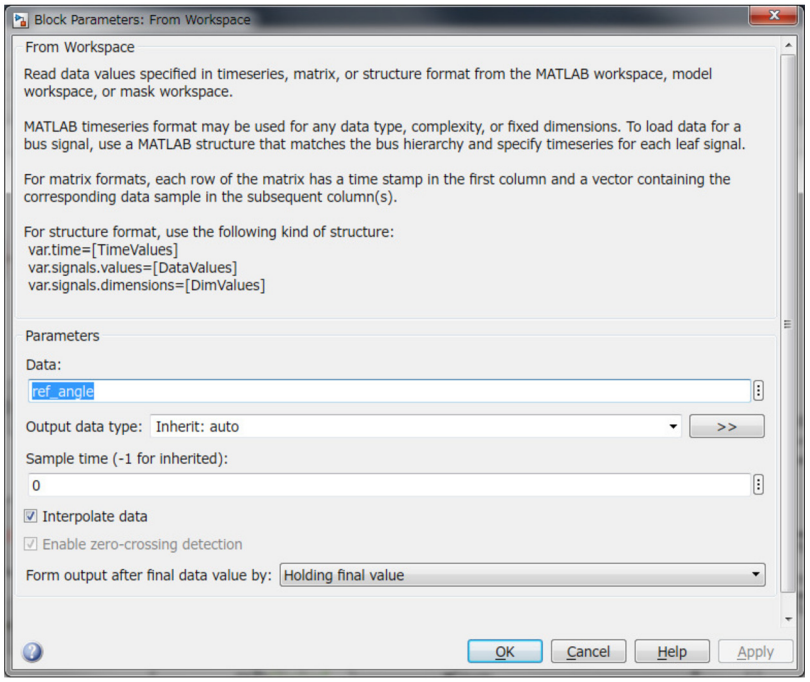


FIGURE 8.32 Property of the "Form Workspace" block. In order to prevent unintended movement, set the "Form output after final data value by" item to "Holding final value," as shown in this figure.

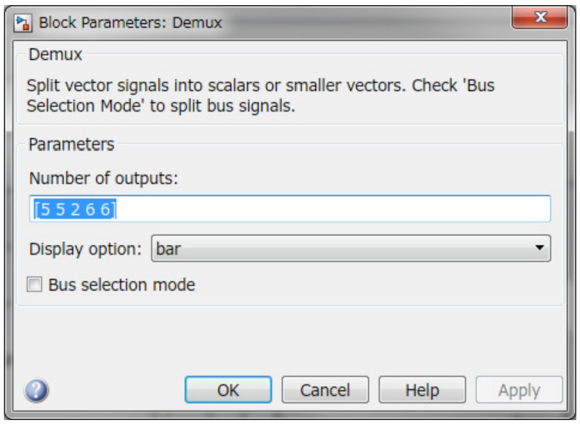


FIGURE 8.33 Parameters of the "Demux" block. The composite reference joint angle vector is decomposed by the "Demux" block into angles for the legs, the neck and the arms. Since the number of inputs of the arms, the neck, and the legs are five, two, and six, respectively, set the "Number of outputs:" item to [5 5 2 6 6].

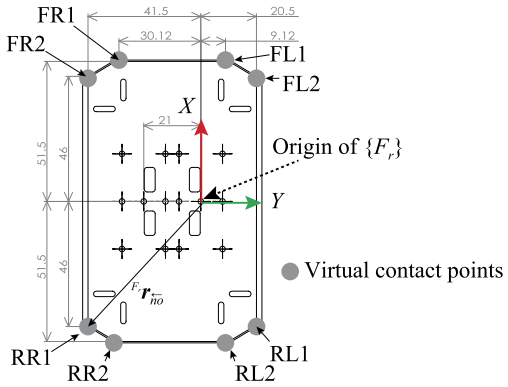


FIGURE 8.34 Definition of contact points on the sole (a view of the sole from below). Eight virtual contact points are placed per sole so that the robot has 16 contact points in total. The force acting on each point is calculated by a spring-damper model. The friction cones are as shown in Fig. 8.35.

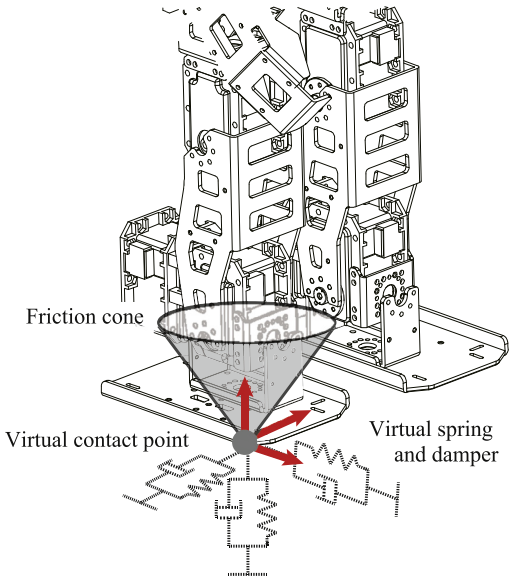


FIGURE 8.35 Contact force model based on the virtual spring-damper model and the friction cone. The force acting at each point shown in Fig. 8.34 is computed based on this model. The reaction force in the normal direction and the tangential forces are expressed by (8.1) and (8.2), respectively.

8.4.4 Modeling of Contact Forces

Assuming the robot is in contact with the floor only at the soles, the contact forces between the soles and the floor are modeled based on the sample point contact model described in Section 3.3.2. As shown in Fig. 8.34, eight virtual contact points are placed at each sole. Thus, the robot comprises 16 contact points in total. The force acting at each point is calculated by a spring-damper model. The friction cones are shown in Fig. 8.35. Here, the surface of the

horizontal floor is assumed to be at zero ground level. Therefore, when the height of the nth virtual contact point z_n is negative, the point will penetrate the floor and a reaction force will be generated based on the depth and speed of the penetration.

Using the spring-damper model, the reaction force in the normal direction is expressed by the following equations:

$$f_{k_{nz}} = \begin{cases} -K_z \Delta z_n - D_z \dot{z}_n & (z_n < 0), \\ 0 & (z \geq 0), \end{cases} \tag{8.1}$$

where K_z and D_z denote the spring constant and the damping coefficient of the virtual spring and damper, respectively; Δz_n stands for the penetration depth of the point w.r.t. the floor surface. The tangential forces can be obtained from (3.9) and (3.10) as

$$f_{k_{nt}} = \begin{cases} -K_t \Delta t_n - D_t \dot{t}_n & (z_n < 0,\ \sqrt{f_{k_{nx}}^2 + f_{k_{ny}}^2} \leq \mu_k f_{k_{nz}}), \\ -\mu_k f_{k_{nz}} \dfrac{v_{k_{nt}}}{\sqrt{v_{k_{nx}}^2 + v_{k_{ny}}^2}} & (z_n < 0,\ \sqrt{f_{k_{nx}}^2 + f_{k_{ny}}^2} > \mu_k f_{k_{nz}}), \\ 0 & (z_n \geq 0), \end{cases} \tag{8.2}$$

where $t \in \{x, y\}$, and μ_k (> 0) denotes the constant static friction coefficient. Fig. 8.36 shows a Simulink diagram based on this contact force model. This diagram is repeatedly used as many times as the number of the virtual contact points. Therefore, it is recommend to save it as a library (My_Libs.slx) in the same folder as op2_simulation.slx, as shown in Fig. 8.17. The Physical Modeling Connection (PMC) port named "External Force" is connected to the virtual contact point (cf. Fig. 8.37) and the "Transform Sensor" block generates the position of the point. The "External Force and Torque" block applies the computed reaction force to the point.

The bottom area, surrounded by a rectangle, displays the calculation method of the normal force expressed in (8.1). The "Saturation" block is connected as shown in Fig. 8.36, in order to avoid generating attractive forces; Fig. 8.38 shows the property of the "Saturation" block. As shown in the figure, set the "Upper limit" item to "inf" and the "Lower limit" item to "0." The middle area, surrounded by a rectangle, displays the calculation method of the tangential force as given in the first row of (8.2). The output of the "Sample and Hold" block follows the input when the point does not penetrate the floor, or it slides on it when $\Delta t_{i,n}$ is zero. Otherwise, the output is on hold, i.e. when the point penetrates the floor and sticks onto it. Then, $\Delta t_{i,n}$ is computed. The "Sample and Hold" block is included in the Simscape Power Systems toolbox. If this toolbox is not available, a block that has the same function as the "Sample and Hold" block can be made, as shown in Fig. 8.39. This block can be used instead of the "Sample and Hold" block. The MATLAB code for this block is as follows.

Listing 8.1: MATLAB function for the block that has the same function as the "Sample and Hold" block.

```
function y = fcn(In,h,S)
if S>0
    y=In;
else
    y=h;
end
```

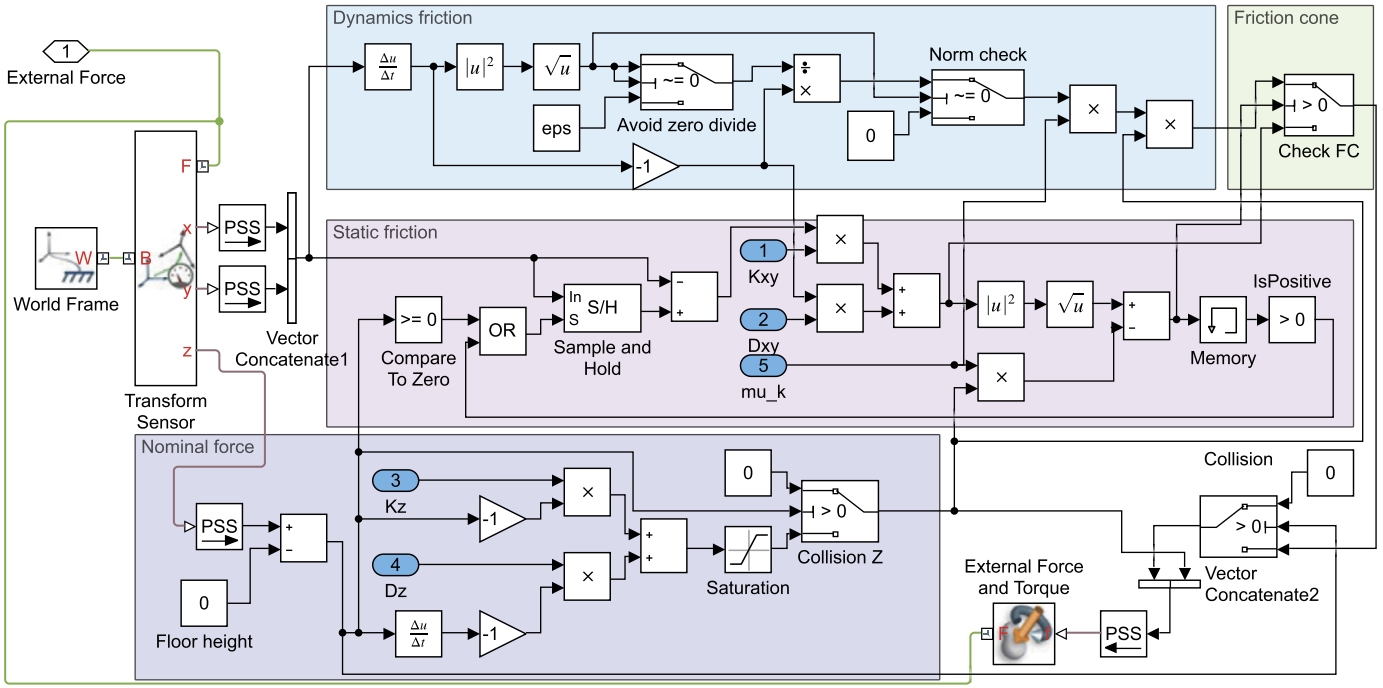


FIGURE 8.36 Modeling of the contact forces in Simulink based on (3.9) and (3.10). The bottom area, surrounded by a rectangle, displays the calculation method of the nominal force expressed in (8.1). The middle area, surrounded by a rectangle, displays the calculation method of the tangential force, as given in the first row of (8.2). The left-top area, surrounded by a rectangle, displays the calculation method of the tangential force as given in the second row of (8.2). The right-top area, surrounded by a rectangle, switches the tangential force based on the friction cone.

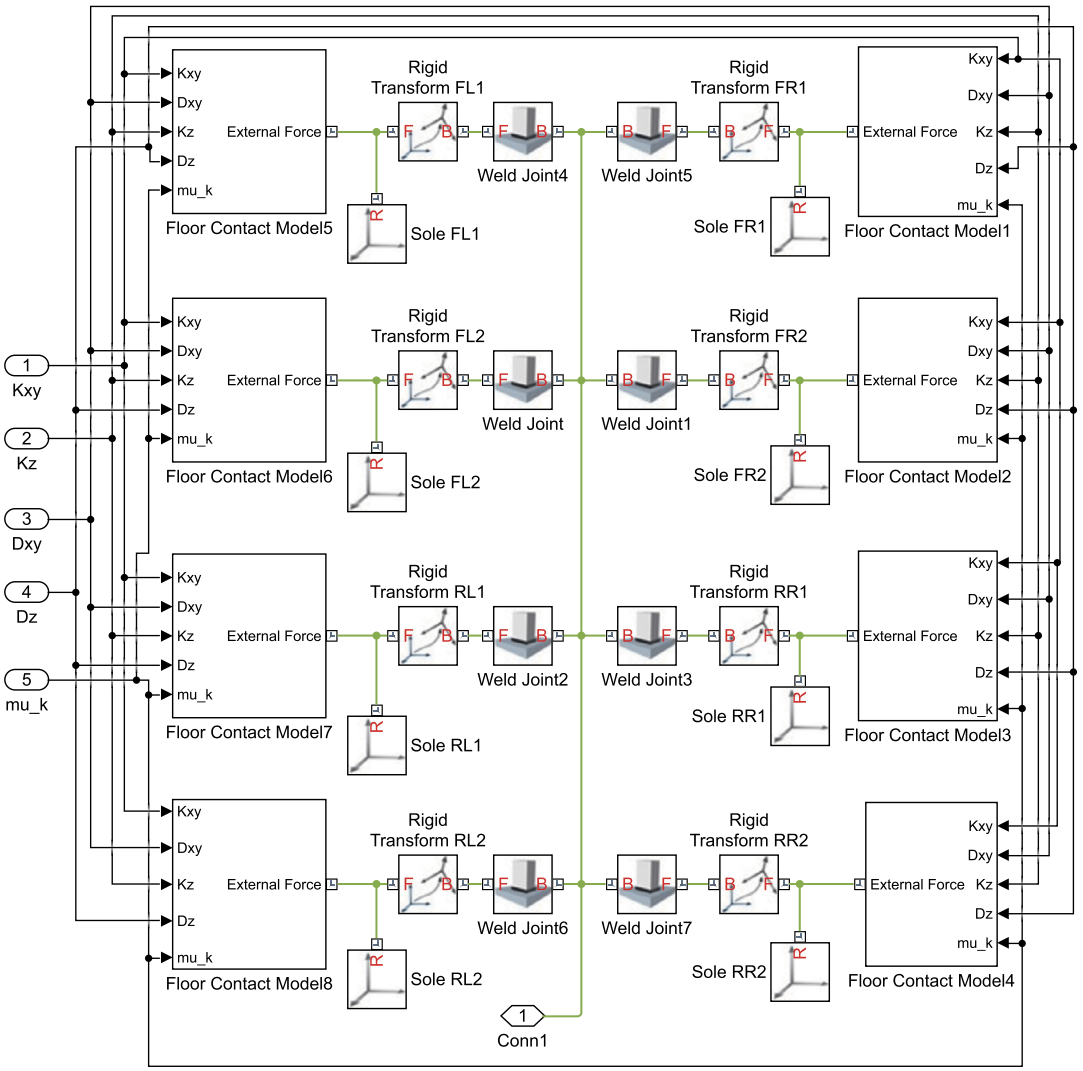


FIGURE 8.37 Definition of the virtual contact points on the right sole. The sum of the floor reaction forces given in the diagram shown in Fig. 8.36 is transmitted to the leg via the origin of the sole shown in Fig. 8.34.

The left-top area, surrounded by a rectangle, displays the calculation method for the tangential force expressed in the second row of (8.2). The right-top area, surrounded by a rectangle, switches the tangential force based on the friction cone. The computed force is an input into the "External Force and Torque" block via the "PS-Simulink Converter" block. In order to apply the force to the point, check the "Force" item in the property of the "External Force and Torque" block. Since (8.1) and (8.2) denote forces at the virtual point in the world frame, set the "Force Resolution Frame" item to "World," as shown in Fig. 8.40.

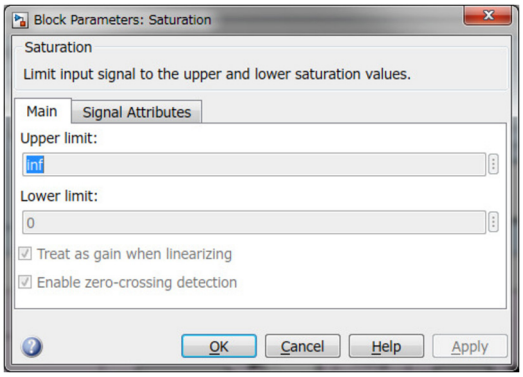


FIGURE 8.38 In order to avoid generating attractive forces, the "Saturation" block is connected in Fig. 8.37. This figure shows the property of the "Saturation" block. As shown in this figure, set the "Upper limit:" item to "inf" and the "Lower limit:" item to "0."

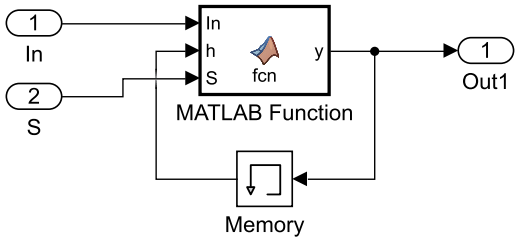


FIGURE 8.39 A block that has the same function as the "Sample and Hold" block.

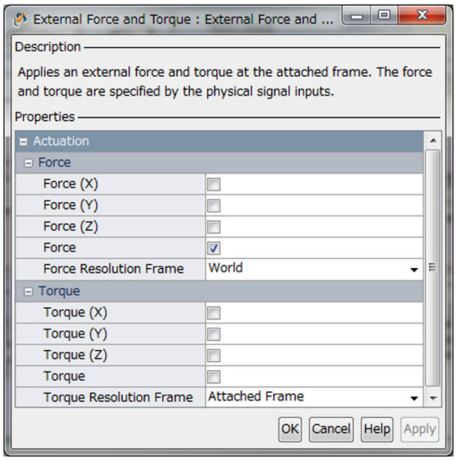


FIGURE 8.40 Property of the "External Force and Torque" block. In order to apply the force to the point, check the "Force" item and set the "Force Resolution Frame" item to "World" (since (8.1) and (8.2) are forces at the virtual point in the world frame).

TABLE 8.2 Offset parameters for the "Rigid Transform" blocks for the right sole. These offsets are measured as shown in the drawing in Fig. 8.34

Rigid Transform	Offset (mm)	Rigid Transform	Offset (mm)
FL1	[51.5 9.12 0]	FR1	[51.5 −30.12 0]
FL2	[46 20.5 0]	FR2	[46 −41.5 0]
RL1	[−46 20.5 0]	RR1	[−46 −41.5 0]
RL2	[−51.5 9.12 0]	RR2	[−51.5 −30.12 0]

Next, the sum of the floor reaction forces, given in the diagram shown in Fig. 8.36, is transmitted to the leg via the origin of the sole shown in Fig. 8.34. The force $\boldsymbol{f}_k$ and moment $\boldsymbol{m}_k$ at the origin of the sole in frame $\{k\}$ ($k \in \{F_r, F_l\}$) can be denoted as follows:

$$\boldsymbol{f}_k = \sum_{n=1}^{N} {}^k\boldsymbol{R}_W \boldsymbol{f}_{k_n}, \tag{8.3}$$

$$\boldsymbol{m}_k = \sum_{n=1}^{N} {}^k\boldsymbol{r}_{\overleftarrow{no}} \times {}^k\boldsymbol{R}_W \boldsymbol{f}_{k_n}, \tag{8.4}$$

where ${}^k\boldsymbol{R}_W \in \Re^{3\times3}$ denotes the rotation matrix that transforms vectors from the world frame to the foot frame $\{k\}$; $\boldsymbol{f}_{k_n} = \left[f_{k_{nx}}\ f_{k_{ny}}\ f_{k_{nz}}\right]^T$, N, and ${}^k\boldsymbol{r}_{\overleftarrow{no}}$ are the floor reaction force denoted in (8.1) and (8.2), the number of the contact points, and the position vector of the nth contact point in $\{k\}$, respectively. The diagram shown in Fig. 8.37 describes the geometrical relationships between the foot origin and the virtual contact points. The PMC port named "Conn1" is connected at the foot origin and the subsystem named "Floor Contact Model" is the library block shown in Fig. 8.36. The "Rigid Transform" block stands for ${}^k\boldsymbol{r}_{\overleftarrow{no}}$. The "Floor Contact Model" block outputs the tangential force components of $\boldsymbol{f}_k$ based on the spring constant "Kxy" and the damping coefficient "Dxy." In addition, the block outputs the nominal force of $\boldsymbol{f}_k$ based on the sprint constant "Kz," the damping coefficient "Dz," and the friction coefficient "mu_k." The values of the spring constants, the damping coefficients, and the friction coefficient are specified by the Simulink standard input ports. As shown in Fig. 8.41, the position of the virtual contact point can be set in the "Offset" item. The "Offset" values for all the points on the right sole are shown in Table 8.2. The offsets are measured as shown in the drawing in Fig. 8.34. Also, set the left sole to be symmetrical to the right one.

In order to connect the PMC port "Conn1" to the origin of the right foot frame, it is needed to make the right foot frame. Fig. 8.42 shows the geometrical relationship between the ankle and the foot frames. The origin of the foot frame is located at the projection of the origin of the ankle frame on the plantar surface. The directions of the x-, y-, and z-axes of the foot frame point forward, to the left, and upward.

Fig. 8.43 shows a diagram describing the kinematic relationships for the right sole. As shown in Fig. 8.28, the "MP_ANKLE2_R" body is at the end of the tree originally, and a PMC port named "Connection Port" is added to the "Reference Frame" of the "MP_ANKLE2_R" body. In order to attach the foot frames, the "Rigid Transform" block named "For rotation" is connected to the "Connection Port" that rotates from the reference frame of the "MP_ANKLE2_R" to the frame parallel to the foot frames. In the "Rigid Transform" block, the rotation

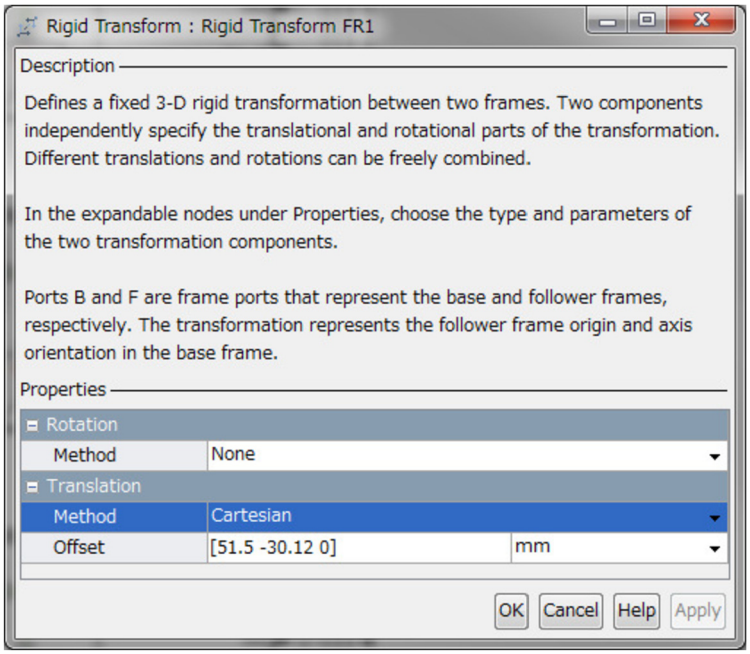


FIGURE 8.41 Property of the "Rigid Transform" block. The position of the virtual contact point can be set in the "Offset" item. The "Offset" values for all the points on the right sole are shown in Table 8.2.

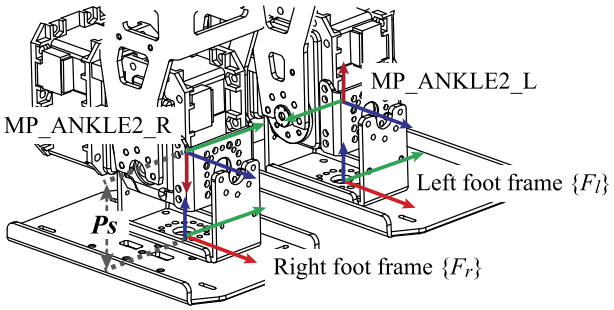


FIGURE 8.42 Geometrical relationship between the ankle and the foot frames. The origin of the foot frame is located at the projection of the origin of the ankle frame on the plantar surface. The directions of the x-, y-, and z-axes of the foot frame point forward, to the left, and upward.

can be applied by choosing a rotation method from the "Aligned Axes," the "Standard Axes," the "Arbitrary Axis," the "Rotation Sequence," and the "Rotation Matrix" methods. In this case, using the "Aligned Axes," it is easy to specify the rotation between the two frames. As shown in Fig. 8.42, the direction along the z-axis of the reference frame of the "MP_ANKLE2_R" is the same as the direction along the x-axis, and the directions along the z-axis of the two frames are the same. Fig. 8.44 shows the property of the "Rigid Transform" block named "For rotation" based on this geometrical relationship. The distance $\boldsymbol{P}_s$ between the

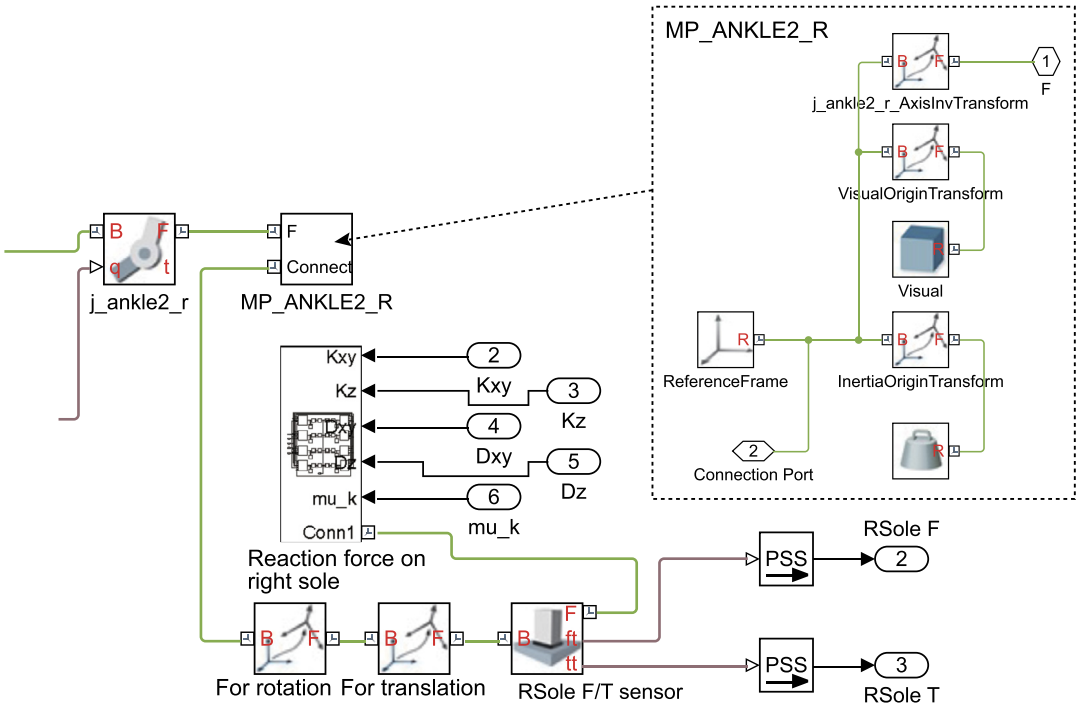


FIGURE 8.43 Making the right foot frame and an F/T (force and torque) sensor. A PMC port named "Connection Port" is added to "Reference Frame" of the "MP_ANKLE2_R" body. In order to attach the foot frame, the "Rigid Transform" block named "For rotation" is connected to "Connection Port" that rotates from the reference frame of "MP_ANKLE2_R" to the frame parallel to the foot frame. The distance between the reference frame of "MP_ANKLE2_R" and the right foot frame is applied by the "Rigid Transform" block named "For translation." In order to measure the force and the moment applied to the sole, a virtual F/T sensor is mounted at the origin of the foot frame. This virtual sensor can be implemented with the "Weld Joint" block that is a 0-DoF joint.

reference frame of the "MP_ANKLE2_R" and the right foot frame is $[0\ 0\ -33.5]^T$ (mm) and can be applied to the "Rigid Transform" block named "For translation" block in the same way as in Fig. 8.41.

In order to measure the force and the moment applied to the sole, a virtual F/T (force and torque) sensor is mounted on the origin of the foot frame. The measured force and moment will be used in Section 8.4.5 for computing the zero-moment point (ZMP). The virtual sensor can be implemented by the "Weld Joint" block, which is a 0-DoF joint. By checking the "Total Force" and the "Total Torque" in the property of the "Weld Joint," as shown in Fig. 8.45, the block measures these values. The measured values are connected to the Simulink standard output port via the "PS-Simulink Converter" block. The follower port of the "Weld Joint" is connected to the PMC port named "Conn1" of the subsystem named "Reaction force on right sole," which is explained in Fig. 8.37. Set these configurations to the left sole as well. The "Constant" blocks used for setting the parameters of the contact force model are placed at the top level of the hierarchical diagram shown in Fig. 8.46. By clicking on these blocks, the parameters of all the contact points can be set easily at the same time. In addition, a solid model

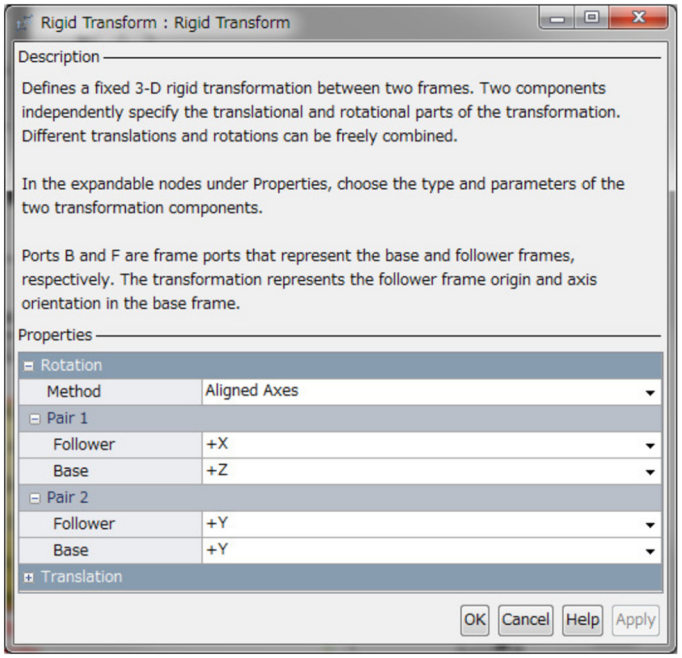


FIGURE 8.44 Property of the "Rigid Transform" block named "For rotation." In the "Rigid Transform" block, the rotation can be applied by choosing a rotation method from the "Aligned Axes," the "Standard Axes," the "Arbitrary Axis," the "Rotation Sequence," and the "Rotation Matrix" methods. In this case, using "Aligned Axes," it is easy to specify the rotation between the two frames.

is added to the top level diagram in the hierarchy for visualizing the floor. The "Solid" block named "Floor" in Fig. 8.46 is a large thin brick connected to the world fame. The property of the "Solid" block named "Floor" is shown in Fig. 8.47.

8.4.5 Computing the ZMP

The ZMP of a multilink system can be computed from the time derivative of the linear and angular momenta of the robot, as described in Section 4.4.5. When the robot is in contact with the floor surface only at its soles, the ZMP can be obtained based on the law of action and reaction from the force and moment measured by the F/T sensors mounted on the soles as follows:

$$r_{p_x} = -\frac{m_y}{f_z} = -\frac{m_{ry} + m_{ly}}{f_{rz} + f_{lz}}, \tag{8.5}$$

$$r_{p_y} = \frac{m_x}{f_z} = \frac{m_{rx} + m_{lx}}{f_{rz} + f_{lz}}, \tag{8.6}$$

where r_{p_t}, f_z, and m_j $(t \in \{x, y\})$ denote the position of the ZMP, the total force in the z-direction, and the moment around the j-axis in the world frame, respectively; f_{jz} and m_{jt} $(t \in \{x, y\},\ j \in \{r, l\})$ are respectively the force in the z-direction and the moment around the

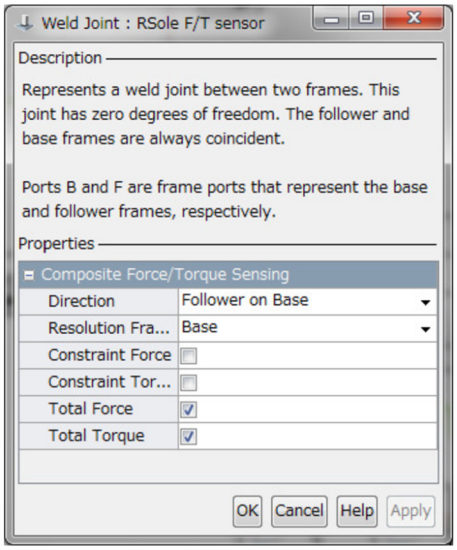


FIGURE 8.45 Property of the "Weld Joint" block used as a virtual F/T (force and torque) sensor. The virtual sensor can be implemented with the "Weld Joint" block which is a 0-DoF joint. By checking "Total Force" and "Total Torque" in the property of "Weld Joint," as shown in this figure, the block outputs these values to the Simulink standard output ports named "RSole F" and "RSole T."

t-axis in the world frame measured by each sensor. We have

$$^{w}\boldsymbol{f}_k = {}^{w}\boldsymbol{R}_k \boldsymbol{f}_k, \tag{8.7}$$

$$^{w}\boldsymbol{m}_k = {}^{w}\boldsymbol{R}_k \boldsymbol{m}_k + \boldsymbol{r}_k \times {}^{w}\boldsymbol{f}_k, \tag{8.8}$$

where $^{w}\boldsymbol{f}_k$ and $^{w}\boldsymbol{m}_k$ are the measured force and moment in the world frame, respectively; $\boldsymbol{r}_k$ and $^{w}\boldsymbol{R}_k$ denote respectively the position vector and the rotation matrix of the foot frame in the world frame.

Fig. 8.48 shows the Simulink implementation of (8.7) and (8.8) for the sensor mounted on the right sole. In order to obtain $\boldsymbol{r}_k$ and $^{w}\boldsymbol{R}_k$, the "Transform Sensor" block is connected to the virtual sensor block named "RSole F/T sensor." Although (8.7) and (8.8) can be implemented with a Simulink diagram, the "MATLAB Function" block is used for describing them simply. Input the measured force $\boldsymbol{f}_k$, the moment $\boldsymbol{m}_k$, $\boldsymbol{r}_k$, and $^{w}\boldsymbol{R}_k$ into the "MATLAB Function" block. The MATLAB code for this block is as follows.

Listing 8.2: MATLAB function for computing the forces and moments in the world frame.

```
function ft = fcn(mf,mt,wRm,wp)
f=wRm*mf;
t=wRm*mt+cross(wp,f);
ft=[f',t']';
end
```

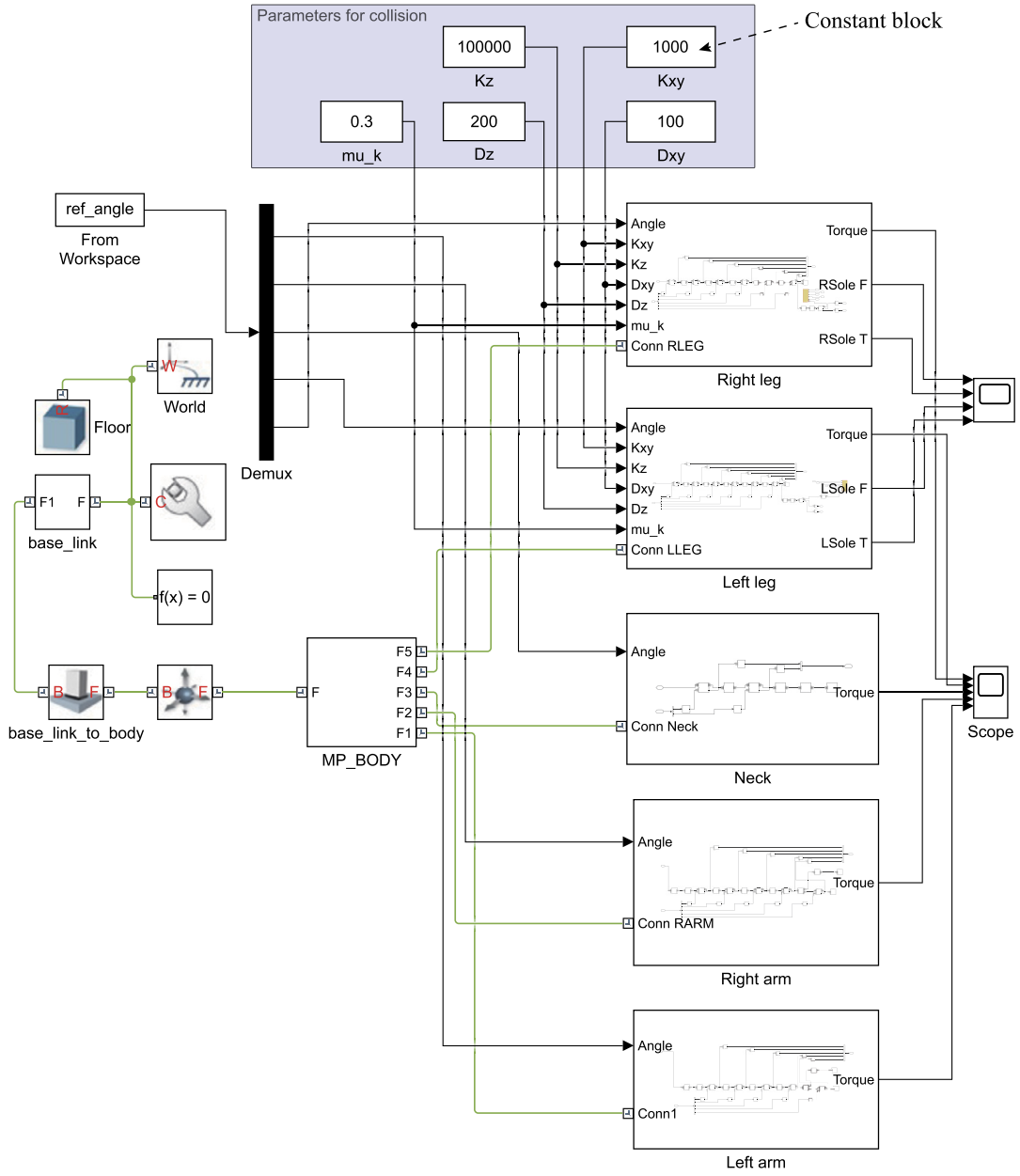


FIGURE 8.46 The top level in the hierarchical diagram for dynamics-based simulation with a humanoid robot. The "Constant" blocks for setting the parameters of the contact force model are placed at the top level of the hierarchical diagram. By clicking on these blocks, the parameters of all the contact points can be set easily at the same time. The "Solid" block named "Floor" is a large thin brick for visualizing the floor.

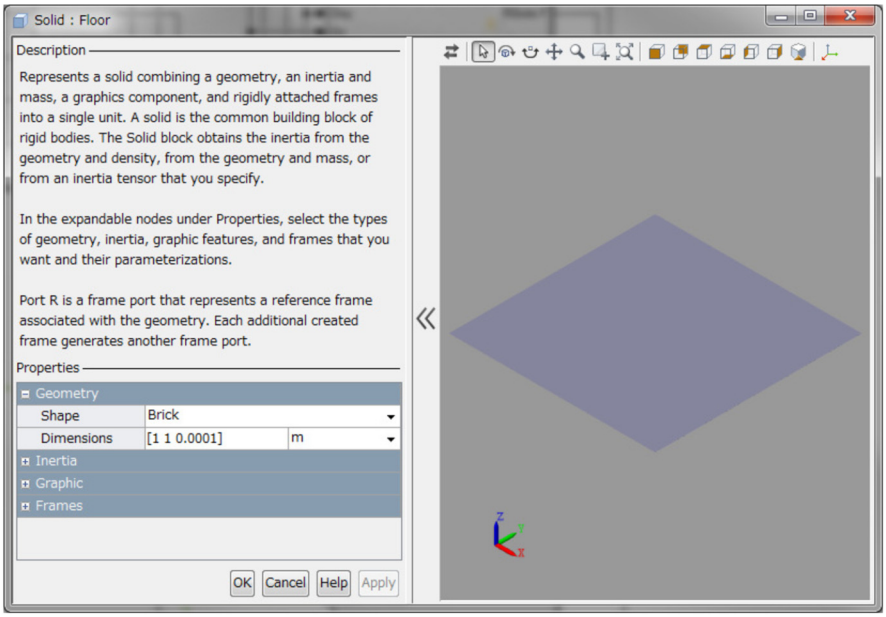


FIGURE 8.47 Property of the "Solid" block named "Floor." This solid model is added to the top level of the hierarchical diagram for visualizing the floor.

The output of the block is a vector which consists of forces and moments computed in the world frame. This output is connected to the Simulink standard output port named "FTw." Likewise, the force and moment in the world frame are also obtained from the sensor mounted on the left sole. For computing the ZMP, input the two vectors into the "MATLAB Function" block as shown in Fig. 8.49. The MATLAB code for this block based on (8.5) and (8.6) is as follows.

Listing 8.3: MATLAB function for computing the position of the ZMP.

```
function pzmp = fcn(FTr,FTl)
pzmp=zeros(3,1);
pzmp(1)=-(FTr(5)+FTl(5))/(FTr(3)+FTl(3));
pzmp(2)=(FTr(4)+FTl(4))/(FTr(3)+FTl(3));
pzmp(3)=0;
end
```

The computed position of the ZMP is sent to a subsystem named "ZMP marker" which displays the position by a sphere in Mechanics Explore. At the time of writing, since there is no function to draw a sphere as a computer graphics feature in Mechanics Explore, the sphere is handled as an object and moved by the "Cartesian joint" as shown in Fig. 8.50. This joint has three translational DoFs represented by three prismatic primitives. The "Cartesian joint" moves the sphere based on a reference position input from the Simulink standard port named "ZMPxyz." Set the items "Force" to the "Automatically Computed" and the items "Motion"

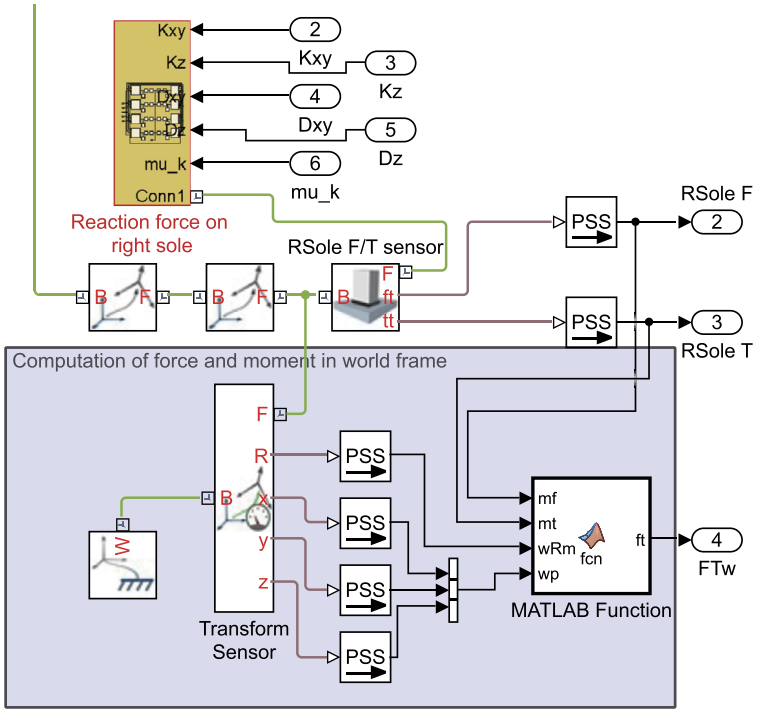


FIGURE 8.48 Computation of the force and moment in the world frame from the measured force and moment in the foot frame. In order to obtain $\boldsymbol{r}_k$ and ${}^w\boldsymbol{R}_k$, the "Transform Sensor" block is connected to the virtual sensor block named "RSole F/T sensor." The measured force $\boldsymbol{f}_k$ and moment $\boldsymbol{m}_k$, as well as $\boldsymbol{r}_k$ and ${}^w\boldsymbol{R}_k$, are connected to the inputs of the "MATLAB Function" block. The output of the block is a vector which consists of forces and moments computed in the world frame.

to the "Provided by Input" in the "X," the "Y," and the "Z" as shown in Fig. 8.51. Fig. 8.52 shows the property of the "Solid" block named "ZMP Marker." Its mass property can have any value.

Since the "MATLAB Function" block in Fig. 8.49 outputs NaN (Not a Number) when the sole has no contact with the floor, the "MATLAB Function" block in Fig. 8.50 handles the error by setting the position of the marker far from the origin as follows.

Listing 8.4: MATLAB function for error handling.

```
function y = fcn(u)
if sum(isnan(u))>0
    y=zeros(3,1);
    y(3)=-1000;
else
    y=u;
end
end
```

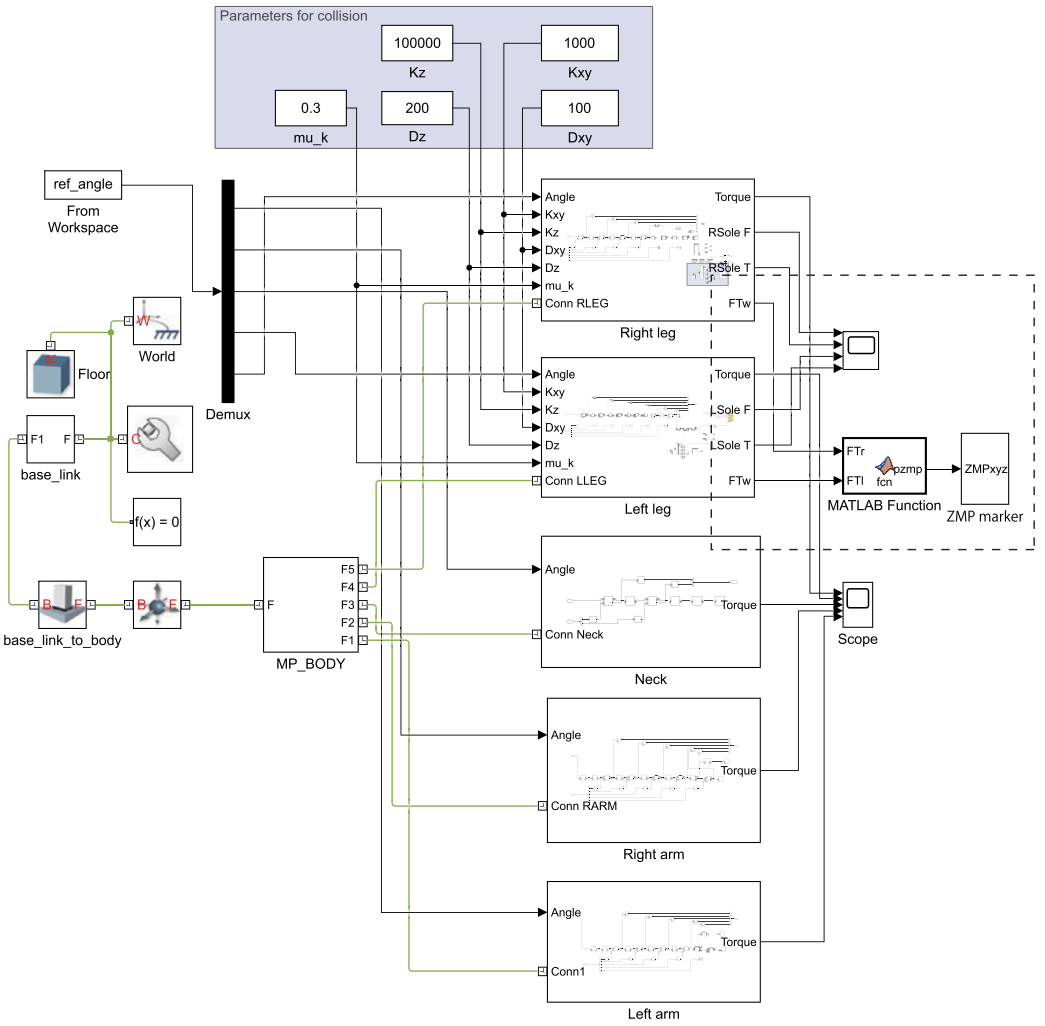


FIGURE 8.49 The top level in the hierarchical diagram with the ZMP marker. The "MATLAB Function" block shown in Fig. 8.48 outputs a vector which consists of forces and moments in the world frame. Such quantities are also obtained from the sensor mounted on the left sole. For computing the ZMP, input the two vectors into the "MATLAB Function" block as shown in this figure.

8.4.6 Motion Design

In this section, it is explained how to design an off-line motion with the MATLAB script utilizing Robotics System Toolbox. The reference trajectory of the left gripper of ROBOTIS-OP2 in the world frame is generated with a fifth-order polynomial interpolation and the time-series reference joint angles are computed by inverse kinematics. The MATLAB script is shown in Fig. 8.53 and is saved as op2_motion.m in the same folder with the Simulink files as shown in Fig. 8.17.

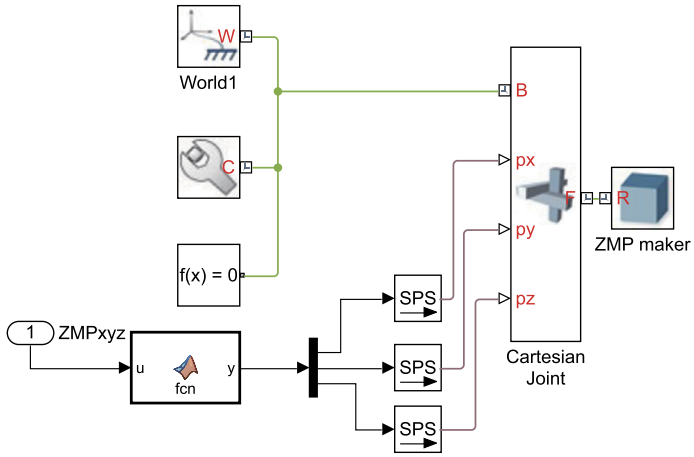


FIGURE 8.50 Subsystem for the ZMP marker. The computed position of the ZMP is sent to this subsystem which displays the position by a sphere in Mechanics Explore. The sphere is handled as an object and moved by the "Cartesian joint" as shown in this figure.

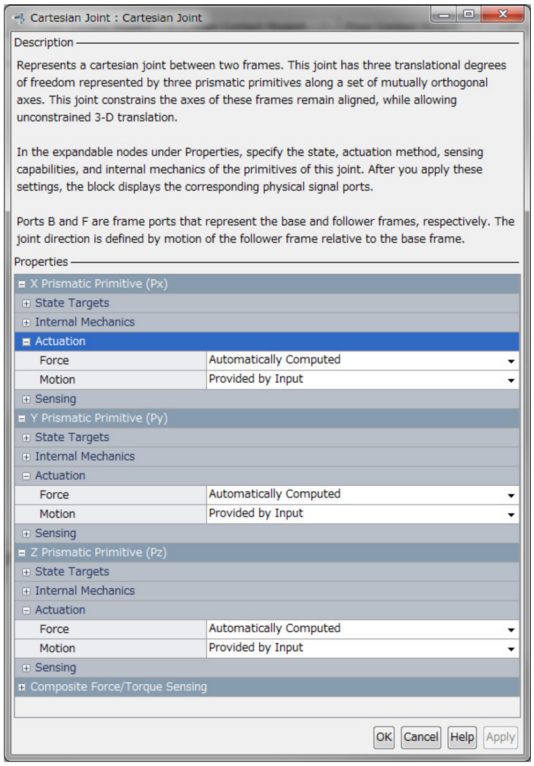


FIGURE 8.51 Property of the "Cartesian joint" block. This joint has three translational DoFs represented by three prismatic primitives.

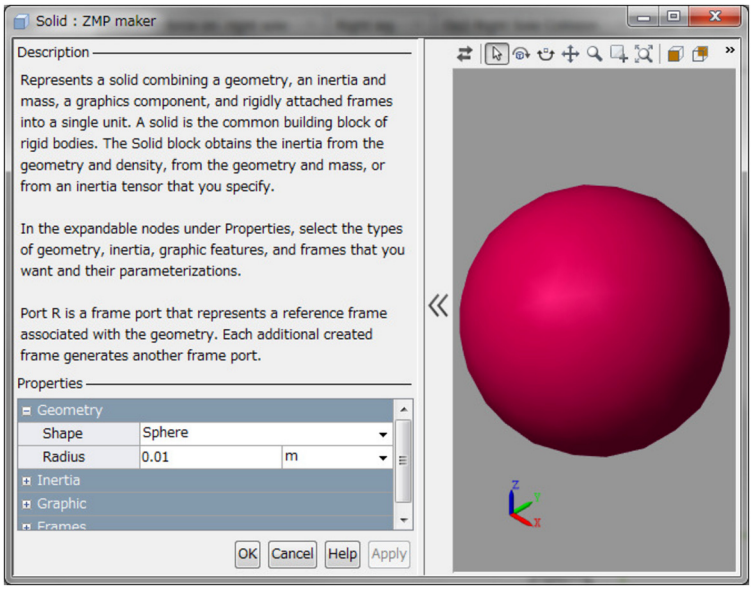


FIGURE 8.52 Property of the "Solid" block named "ZMP Marker."

In line 3, the URDF files of ROBOTIS-OP2 are parsed and stored in the Workspace memory. In line 11, the coordinate frame of the left gripper named "MP_ARM_GRIPPER_FIX_DUMMY_L" (cf. Fig. 8.14) is specified as the end effector. In order to generate the reference trajectory of the end effector with a fifth-order polynomial interpolation, the coefficients for the interpolation are prepared with the code from lines 15 to 29. The solver for the inverse kinematics is initialized in line 32. The first three elements of the variable "weight" in line 33 correspond to the weights on the tolerances for the orientation for the desired pose. The last three elements correspond to the weights on the tolerances for the x-y-z-position of the gripper for the desired pose. Form lines 35 to 46, for each sampling time, the trajectory is calculated and the inverse kinematics solver computes the joint angles. From lines 49 to 52, the stationary motion is attached before the motion generated in the section from lines 35 to 46. In line 55, the reference joint angles are stored in the Workspace as a time-series matrix, and they will be available in Simulink.

8.4.7 Simulation

After executing the MATLAB script explained in Section 8.4.6, the Simulink diagram (op2_simulation.slx) is ready to be executed. Fig. 8.54 shows an example of a simulation condition. The type of the solver and its time step should be carefully chosen based on the intended situation. In this example, the solver is the Runge–Kutta method and the step size is 1 ms. By clicking on the play icon (▷) on the Simulink window, the simulation starts. Figs. 8.55 and 8.56 show snapshots of the simulation and the result of the simulation, respectively. In the animation of the simulation, the ZMP marker moves in accordance with the designed

```
clearvars;
addpath('.\RobotModels\robotis_op_description\meshes');
op2=importrobot('.\RobotModels\robotis_op_description\robots\robotis_op.urdf');
q0=homeConfiguration(op2);

%% Generating the end-effector trajectory
SamplingTime=0.01;
tf=2;
t = (0:SamplingTime:tf)'; % Time
count = length(t);
endEffector = 'MP_ARM_GRIPPER_FIX_DUMMY_L';
T=op2.getTransform(q0,endEffector);
p=T(1:3,4);
%coefficients for 5th order polynomial interpolation
xs=0;
dxs=0;
ddxs=0;
xf=0.03;
dxf=0;
ddxf=0;
a0 = xs;
a1 = dxs;
a2 = ddxs/2.0;
a3 = (20.0*xf - 20.0*xs - (8.0*dxf + 12.0*dxs)*tf - ...
    (3.0*ddxs - ddxf)*tf.^2.0)/(2.0*tf.^3.0);
a4 = (30.0*xs - 30.0*xf + (14.0*dxf + 16.0*dxs)*tf + ...
    (3.0*ddxs - 2.0*ddxf)*tf.^2.0)/(2.0*tf.^4.0);
a5 = (12.0*xf - 12.0*xs - (6.0*dxf + 6.0*dxs)*tf - ...
    (ddxs - ddxf)*tf.^2.0)/(2.0*tf.^5.0);

%% Inverse kinematics
ik = robotics.InverseKinematics('RigidBodyTree', op2);
weights = [0 0 0 1 1 1];
qInitial=q0;
for i = 1:count
    ti=SamplingTime*(i-1);
    %5th order polynomial interpolation
    pi=a0+a1*ti+a2*ti^2+a3*ti^3+a4*ti^4+a5*ti^5;
    T(1:3,4)=  [p(1)+pi,p(2)-pi,p(3)]';
    % Solving inverse kinematics
    [qSol,solInfo] = ik(endEffector,T,weights,qInitial);
    % Store the configuration
    ref_q(i,:) = [qSol.JointPosition];
    % Start from prior solution
    qInitial = qSol;
end

%% Generating motion to be stationary for half second.
t_stop=0.5;
t_init = (0:SamplingTime:(t_stop-SamplingTime))'; % Time
len = length(t_init);
ref_q_init(1:len,:)=ones(len,1)*ref_q(1,:);

%% Making timeseries matrix for Simulink
ref_angle=timeseries([ref_q_init;ref_q],[t_init;(t+t_stop)]);
```

FIGURE 8.53 Matlab program code for moving the left arm. The reference trajectory of the gripper (MP_ARM_GRIPPER_FIX_DUMMY_L) is generated in the world frame with fifth order polynomial interpolation. The time-series reference joint angles are computed with the inverse kinematics.

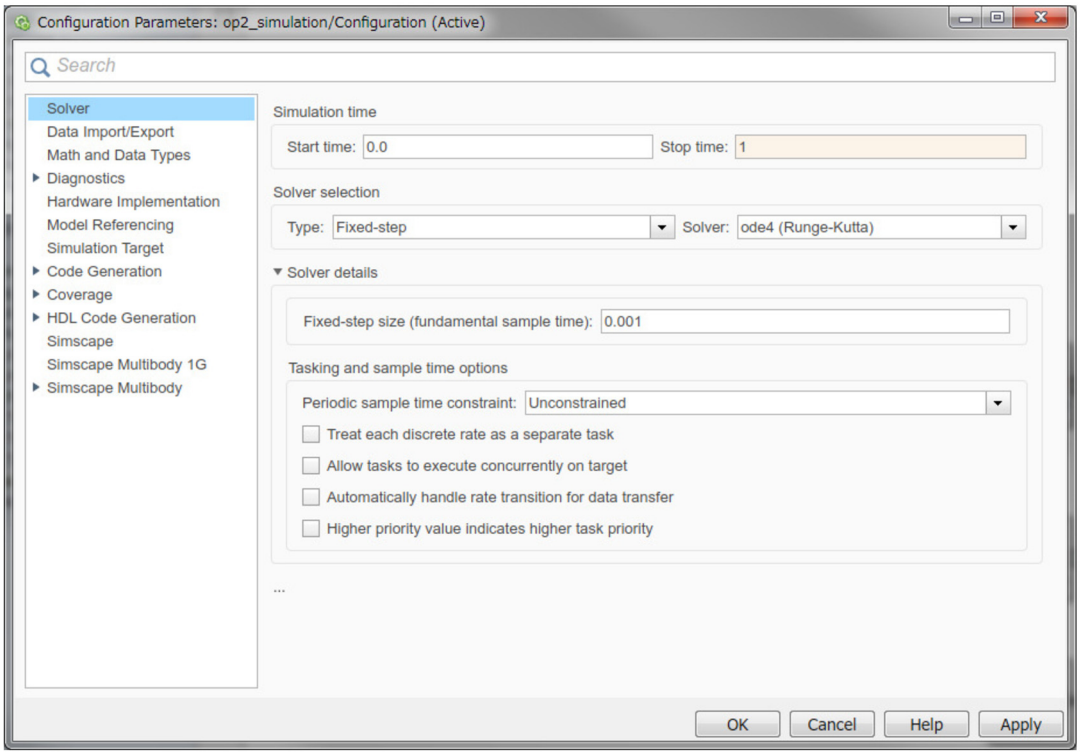
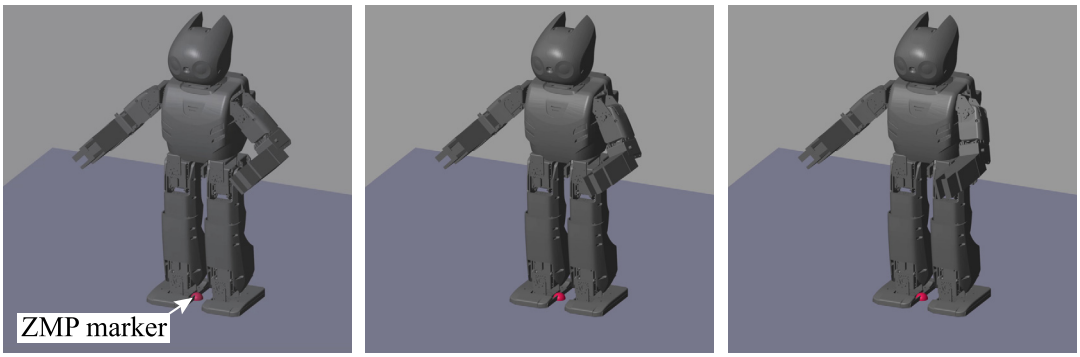
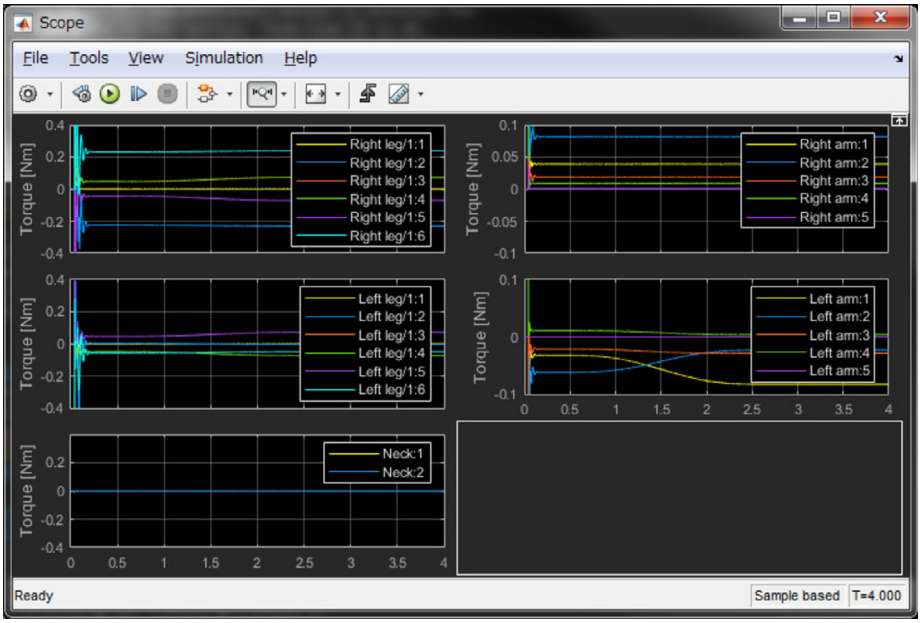


FIGURE 8.54 Configuration parameters for a dynamics-based simulation. The type of the solver and its time step should be carefully chosen based on the intended situation. In this example, the solver is the Runge–Kutta method and the step size is 1 ms.

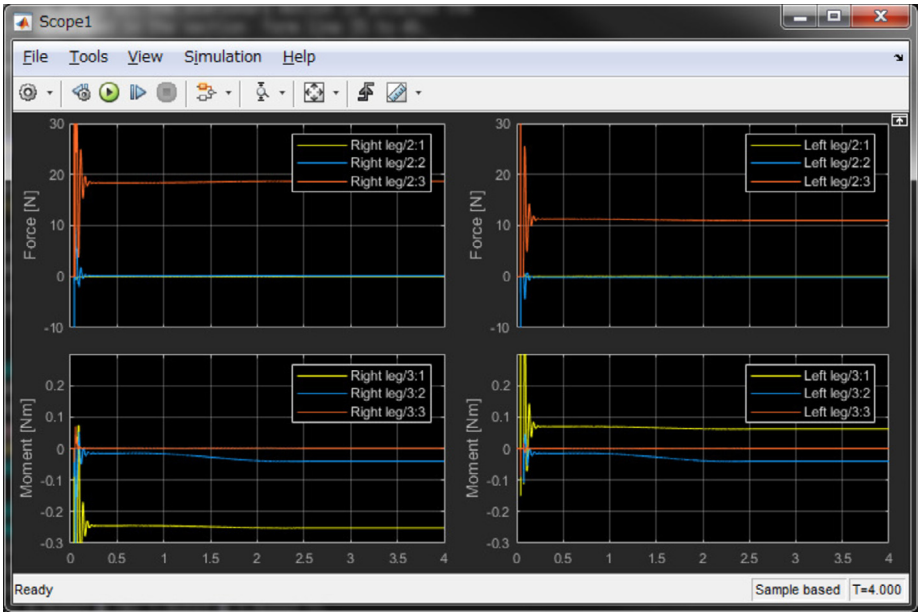


(A) 0.5 [s] (B) 1.5 [s] (C) 2.5 [s]

FIGURE 8.55 Visualization of dynamics-based simulation. As the left gripper moves to the right front, the ZMP marker also moves to the right front.



(A)



(B)

FIGURE 8.56 Simulation results plotted by the "Scope" blocks. (A) Above: Relationship between the time and the joint torques. (B) Below: Relationship between the time and the force/moment. The measured values oscillate in all the plots immediately after the start of the simulation. The robot is dropped from a certain height at the beginning of the simulation. That is why the values oscillate.

motion. As the left gripper moves to the right front, the ZMP marker also moves to the right front. As shown in Fig. 8.56, the measured values oscillate in all the plots immediately after the start of the simulation. The robot is dropped from a certain height at the beginning of the simulation. That is why the values oscillate. As shown in the plot "Left arm" in Fig. 8.56A, the motor torques are changing from 0.5 to 2.5 s.

References

[1] About ROS, http://www.ros.org/about-ros/.
[2] C.E. Agüero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J.L. Rivero, J. Manzo, E. Krotkov, G. Pratt, Inside the virtual robotics challenge: simulating real-time robotic disaster response, IEEE Transactions on Automation Science and Engineering 12 (2015) 494–506.
[3] Atlas, Boston dynamics, https://www.bostondynamics.com/atlas.
[4] BIOROID GP, http://www.robotis.us/robotis-gp/.
[5] Body model, Choreonoid 1.6 documentation, http://choreonoid.org/en/manuals/1.6/handling-models/bodymodel.html.
[6] Brushless DC Motor, Simscape documentation, https://jp.mathworks.com/help/physmod/elec/examples/brushless-dc-motor.html.
[7] Bullet, http://bulletphysics.org/.
[8] S. Chitta, I. Sucan, S. Cousins, MoveIt! [ROS topics], IEEE Robotics & Automation Magazine 19 (2012) 18–19.
[9] Coin3D, https://bitbucket.org/Coin3D/coin/wiki/Home.
[10] D. Cook, A. Vardy, R. Lewis, A survey of AUV and robot simulators for multi-vehicle operations, in: IEEE/OES Autonomous Underwater Vehicles, 2015, pp. 1–8.
[11] DART, http://dartsim.github.io/.
[12] J. Englsberger, A. Werner, C. Ott, B. Henze, M.A. Roa, G. Garofalo, R. Burger, A. Beyer, O. Eiberger, K. Schmid, A. Albu-Schaffer, Overview of the torque-controlled humanoid robot TORO, in: IEEE-RAS International Conference on Humanoid Robots, 2014, pp. 916–924.
[13] C. Ericson, Real-Time Collision Detection, CRC Press, 2004.
[14] R. Featherstone, A divide-and-conquer articulated-body algorithm for parallel O(log(n)) calculation of rigid-body dynamics. Part 1: basic algorithm, The International Journal of Robotics Research 18 (1999) 876–892.
[15] R. Featherstone, Rigid Body Dynamics Algorithms, Springer, 2008.
[16] P. Flores, H.M. Lankarani, Contact Force Models for Multibody Dynamics, Springer, 2016.
[17] FROST: fast robot optimization and simulation toolkit on GitHub, https://github.com/ayonga/frost-dev.
[18] Gazebo, http://gazebosim.org/.
[19] GLFW, http://www.glfw.org/.
[20] GLUT– The OpenGL Utility Toolkit, https://www.opengl.org/resources/libraries/glut/.
[21] GNU Octave, https://www.gnu.org/software/octave/.
[22] Google Maps, https://www.google.com/maps/.
[23] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, B. Maisonnier, Mechatronic design of NAO humanoid, in: IEEE International Conference on Robotics and Automation, 2009, pp. 769–774.
[24] graspPlugin, http://www.hlab.sys.es.osaka-u.ac.jp/grasp/en.
[25] I. Ha, Y. Tamura, H. Asama, J. Han, D.W. Hong, Development of open humanoid platform DARwIn-OP, in: SICE Annual Conference 2011, 2011, pp. 2178–2181.
[26] A. Hereid, A.D. Ames, FROST *: fast robot optimization and simulation toolkit, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2017, pp. 719–726.
[27] HOAP-2, Fujitsu, https://en.wikipedia.org/wiki/HOAP.
[28] S.H. Hyon, D. Suewaka, Y. Torii, N. Oku, H. Ishida, Development of a fast torque-controlled hydraulic humanoid robot that can balance compliantly, in: IEEE-RAS International Conference on Humanoid Robots, 2015, pp. 576–581.
[29] Install the Simscape Multibody Link plug-in, https://jp.mathworks.com/help/physmod/sm/ug/installing-and-linking-simmechanics-link-software.html?lang=en.

[30] S. Ivaldi, J. Peters, V. Padois, F. Nori, Tools for simulating humanoid robot dynamics: a survey based on user feedback, in: IEEE-RAS International Conference on Humanoid Robots, 2014, pp. 842–849.
[31] S. Ivaldi, B. Ugurlu, Free simulation software and library, in: Humanoid Robotics: A Reference, Springer, 2018, Chapter 35.
[32] F. Kanehiro, H. Hirukawa, S. Kajita, OpenHRP: open architecture humanoid robotics platform, The International Journal of Robotics Research 23 (2004) 155–165.
[33] K. Kaneko, F. Kanehiro, M. Morisawa, K. Akachi, G. Miyamori, A. Hayashi, N. Kanehira, Humanoid robot HRP-4 – humanoid robotics platform with lightweight and slim body, in: IEEE International Conference on Intelligent Robots and Systems, 2011, pp. 4400–4407.
[34] C.W. Kennedy, J.P. Desai, Modeling and control of the Mitsubishi PA-10 robot arm harmonic drive system, IEEE/ASME Transactions on Mechatronics 10 (2005) 263–274.
[35] N. Koenig, A. Howard, Design and use paradigms for Gazebo, an open-source multi-robot simulator, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004, pp. 2149–2154.
[36] K. Kojima, T. Karasawa, T. Kozuki, E. Kuroiwa, S. Yukizaki, S. Iwaishi, T. Ishikawa, R. Koyama, S. Noda, F. Sugai, S. Nozawa, Y. Kakiuchi, K. Okada, M. Inaba, Development of life-sized high-power humanoid robot JAXON for real-world use, in: IEEE-RAS International Conference on Humanoid Robots, 2015, pp. 838–843.
[37] J.L. Lima, J.A. Gonçalves, P.G. Costa, A.P. Moreira, Humanoid realistic simulator: the servomotor joint modeling, in: International Conference on Informatics in Control, Automation and Robotics, 2009, pp. 396–400.
[38] Make a model, Gazebo tutorial, http://gazebosim.org/tutorials?tut=build_model&cat=build_robot.
[39] Mates and joints, https://jp.mathworks.com/help/physmod/smlink/ref/mates-and-joints.html.
[40] Mathematica, https://www.wolfram.com/mathematica/index.ja.html?footer=lang.
[41] MATLAB, https://jp.mathworks.com/products/matlab.html.
[42] I. Millington, Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for Your Game, second edition, CRC Press, 2010.
[43] MoveIt!, https://moveit.ros.org/.
[44] S. Nakaoka, Choreonoid: extensible virtual robot environment built on an integrated GUI framework, in: IEEE/SICE International Symposium on System Integration, 2012, pp. 79–85.
[45] S. Nakaoka, S. Hattori, F. Kanehiro, S. Kajita, H. Hirukawa, Constraint-based dynamics simulator for humanoid robots with shock absorbing mechanisms, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007, pp. 3641–3647.
[46] J. Neider, T. Davis, M. Woo, OpenGL Programming Guide, Addison Wesley, 1993.
[47] Newton dynamics, http://newtondynamics.com/forum/newton.php.
[48] ODE (open dynamics engine), http://www.ode.org/.
[49] OpenStreetMap, http://www.openstreetmap.org/.
[50] I.W. Park, J.Y. Kim, J. Lee, J.H. Oh, Mechanical design of humanoid robot platform KHR-3 (KAIST humanoid robot – 3: HUBO), in: IEEE-RAS International Conference on Humanoid Robots, 2005, pp. 321–326.
[51] PhysX, https://developer.nvidia.com/gameworks-physx-overview.
[52] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Mg, ROS: an open-source robot operating system, in: IEEE International Conference on Robotics and Automation Workshop on Open Source Robotics, 2009.
[53] E. Rohmer, S.P. Singh, M. Freese, V-REP: a versatile and scalable robot simulation framework, in: 2013 IEEE International Conference on Intelligent Robots and Systems, 2013, pp. 1321–1326.
[54] ROS control, http://gazebosim.org/tutorials/?tut=ros_control.
[55] S. Shirata, A. Konno, M. Uchiyama, Development of a light-weight biped humanoid robot, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004, pp. 148–153.
[56] Simbody, https://simtk.org/home/simbody/.
[57] Simscape Multibody, https://jp.mathworks.com/products/simmechanics.html.
[58] Simscape Multibody Link plug-in, https://jp.mathworks.com/help/physmod/smlink/index.html.
[59] Simulink, https://jp.mathworks.com/products/simulink.html.
[60] Toyota unveils third generation humanoid robot T-HR3, http://corporatenews.pressroom.toyota.com/releases/toyota+unveils+third+generation+humanoid+robot+thr3.htm.
[61] T. Tsujita, O. Altangerel, S. Abiko, A. Konno, Analysis of drop test using a one-legged robot toward parachute landing by a humanoid robot, in: IEEE International Conference on Robotics and Biomimetics, 2017, pp. 221–226.
[62] Tunnel disaster response and recovery challenge, http://worldrobotsummit.org/en/wrc2018/disaster/.

[63] A URDF file and Mesh files of ROBOTIS-OP2 on GitHub, https://github.com/ROBOTIS-OP/robotis_op_common/tree/master/robotis_op_description.
[64] URDF in Gazebo, Gazebo tutorial, http://gazebosim.org/tutorials/?tut=ros_urdf.
[65] V-Rep, http://www.coppeliarobotics.com/index.html.
[66] G. Van Den Bergen, Collision Detection in Interactive 3D Environments, Elsevier, 2003.
[67] Vortex Studio, https://www.cm-labs.com/vortex-studio/.
[68] Webots, https://www.cyberbotics.com/#webots.
[69] J. Wernecke, The Inventor Mentor, Addison Wesley, 1994.
[70] K. Yamane, Simulating and Generating Motions of Human Figures, Springer-Verlag, Berlin, Heidelberg, 2004.