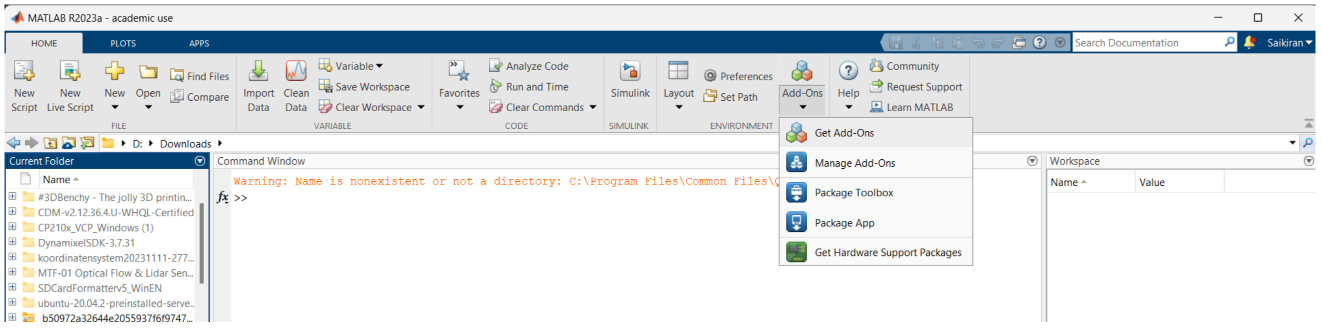
ME 5250 - Tutorial 1

In this tutorial, we introduce some basic functionalities of the MATLAB Robotics Systems Toolbox. This toolbox supports popular robotics standards such as URDF (unified robot description format), and it provides algorithms and functions for designing, simulating, testing, and deploying robotic applications. In this tutorial, you will learn how to create and visualize robots and obtain the homogeneous transformation matrices between the link frames in any given configuration. First, you need to have MATLAB and the Robotics Systems Toolbox installed:

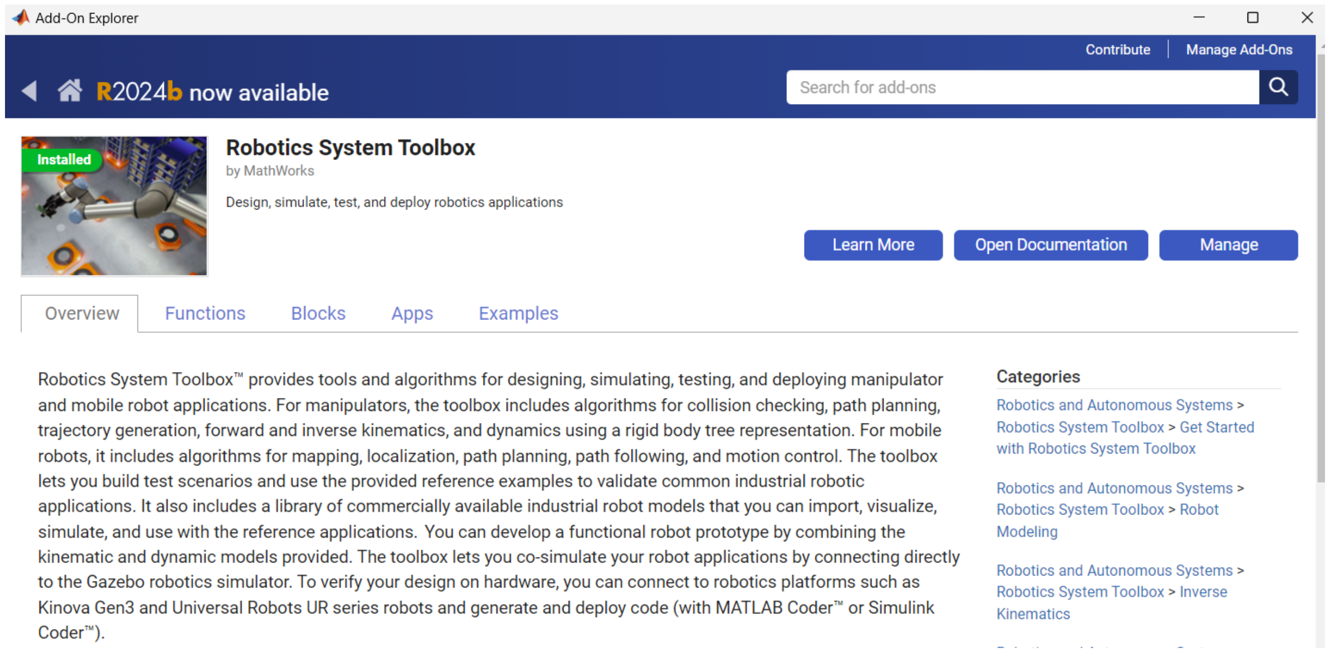
1. Install **MATLAB R2023A** (or the latest) version. Link

2. Install Robotics Systems Toolbox from the packages in MATLAB

Open MATLAB -> click on Add-Ons -> Get Add-Ons -> Search for Robotics Systems Toolbox and Install it.



```
Warning: Name is nonexistent or not a directory: C:\Program Files\Common Files\
>>
```



Robotics System Toolbox

by MathWorks

Design, simulate, test, and deploy robotics applications

Robotics System Toolbox™ provides tools and algorithms for designing, simulating, testing, and deploying manipulator and mobile robot applications. For manipulators, the toolbox includes algorithms for collision checking, path planning, trajectory generation, forward and inverse kinematics, and dynamics using a rigid body tree representation. For mobile robots, it includes algorithms for mapping, localization, path planning, path following, and motion control. The toolbox lets you build test scenarios and use the provided reference examples to validate common industrial robotic applications. It also includes a library of commercially available industrial robot models that you can import, visualize, simulate, and use with the reference applications. You can develop a functional robot prototype by combining the kinematic and dynamic models provided. The toolbox lets you co-simulate your robot applications by connecting directly to the Gazebo robotics simulator. To verify your design on hardware, you can connect to robotics platforms such as Kinova Gen3 and Universal Robots UR series robots and generate and deploy code (with MATLAB Coder™ or Simulink Coder™).

Categories

Robotics and Autonomous Systems > Robotics System Toolbox > Get Started with Robotics System Toolbox

Robotics and Autonomous Systems > Robotics System Toolbox > Robot Modeling

Robotics and Autonomous Systems > Robotics System Toolbox > Inverse Kinematics

The documentation for the inbuilt functions of the toolbox is provided here.

Creating a Robot

There are multiple ways to create a robot in the MATLAB environment. In this tutorial, we cover two standard methods. The first method is by importing a URDF file when it is available. The second method is by defining a rigid body tree using the Denavit-Hartenberg (DH) parameters.

1. Importing a URDF file:

URDF (Unified Robot Description Format) is an XML format used to represent the physical and visual properties of a robot, like links, joints, and sensors. This format is widely used, and many standard robots have their URDFs and computer aided design (CAD) files available. For such robots, “importrobot” function can be used to create the robot model in the MATLAB environment.

There are some predefined robot models in MATLAB, which can be imported directly using their names.

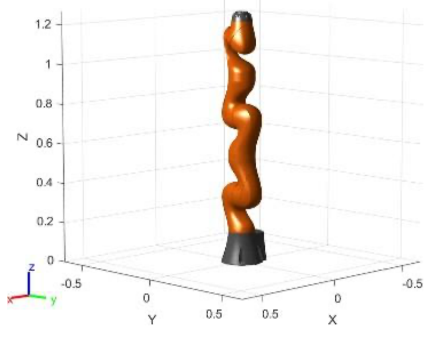
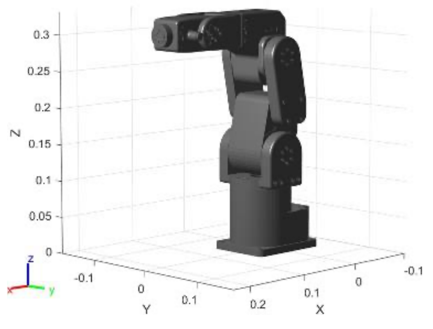
Example 1: We are importing the Puma560 robot model, which is predefined in MATLAB.

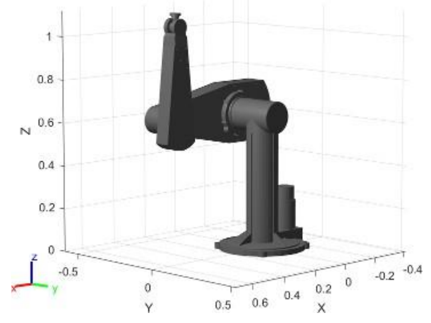
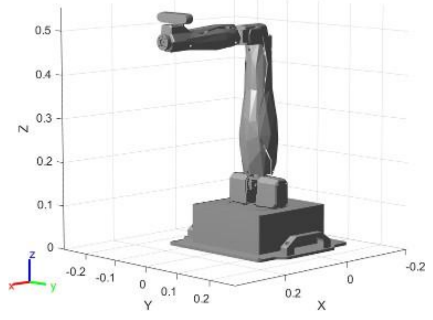
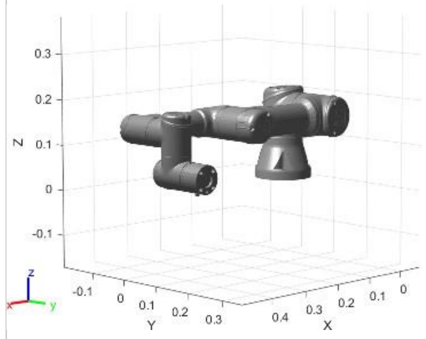
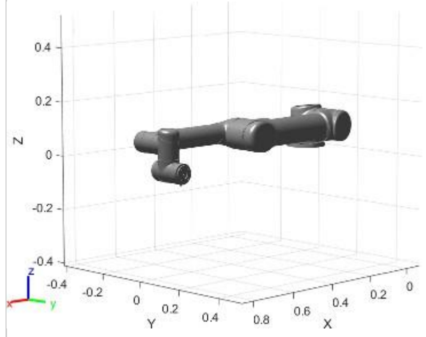
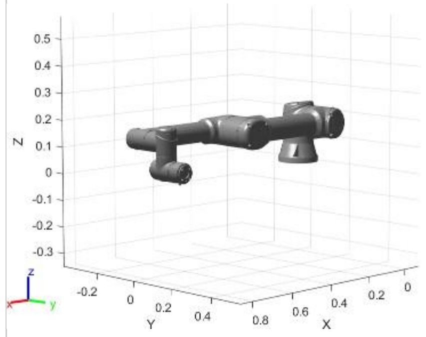
```
>> robot = importrobot('puma560.urdf')
```

Predefined robots can also be imported using the “loadrobot” function. For example, Puma 560 can also be imported using the following command:

```
>> robot = loadrobot('puma560')
```

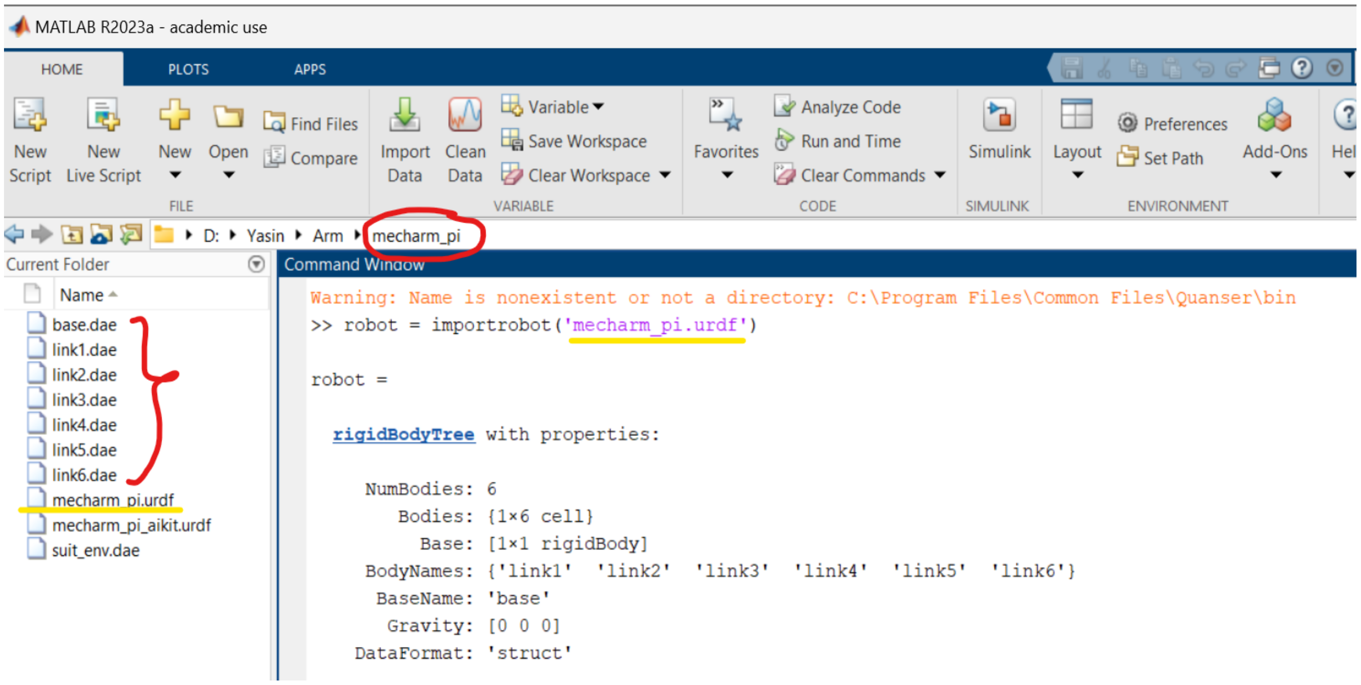
Below are some of the other predefined models available in MATLAB, which can be imported similarly.

Robot Model	Mesh Visualization	Description
"kukaIiwa7.urdf"		KUKA LBR iiwa 7 R800 7-axis robot
"meca500r3.urdf"		Mecademic Meca500 R3 6-axis robot

"puma560.urdf"		PUMA 560 6-axis robot
"quanserQArm.urdf"		Quanser QArm 4 DOF robot
"universalUR3e.urdf"		Universal Robots UR3e 6-axis robot
"universalUR5.urdf"		Universal Robots UR5 6-axis robot
"universalUR5e.urdf"		Universal Robots UR5e 6-axis robot

When importing robots that are not predefined in the Robotics Systems Toolbox, you need to have the corresponding URDF and CAD files (.stl/.dae) added to your working folder (or provide the path to those files).

Example 2: We are importing Mecharm Pi, which is not a predefined robot model in MATLAB. We first add its URDF and CAD files to our working folder. All of these files are provided in the respective folder on the Canvas page under the section “Files/MATLAB”. We can now use the “importrobot” function.



```
Warning: Name is nonexistent or not a directory: C:\Program Files\Common Files\Quanser\bin
>> robot = importrobot('mecharm_pi.urdf')

robot = 

  rigidBodyTree with properties:

     NumBodies: 6
        Bodies: {1×6 cell}
          Base: [1×1 rigidBody]
     BodyNames: {'link1'  'link2'  'link3'  'link4'  'link5'  'link6'}
      BaseName: 'base'
       Gravity: [0 0 0]
    DataFormat: 'struct'
```

2. Defining a Rigid Body Tree:

When the URDF and CAD files are not available, a simplified robot model can still be created in the MATLAB environment by using the DH parameters and defining the robot as a rigid body tree, which is a representation of rigid bodies connected by joints.

Example 3: We are creating the Puma560 robot model as a rigid body tree using its DH parameters. We first define the DH table as a matrix. Each row *i*, corresponds to joint *i.* Using the modified DH convention we have seen in class, the first column of this matrix corresponds to a_{i-1}, the second column is α_{i-1}, the third column is d_i, and the fourth column is θ_i. The values of the joint variables are not important when creating the robot, so we set them to zero here. Once the robot is created, we can change the joint variables and visualize the robot at different configurations.

```
mdhparams = [0          0       0         0;
             0        -pi/2     0          0;
             0.4318     0      0.15005   0;
             0.0203    -pi/2   0.4318    0;
             0          pi/2   0         0;
             0         -pi/2   0         0];
```

Next, we create the robot as a rigid body tree object.

```
robot = rigidBodyTree;
```

We create a cell array for the rigid body objects, and another cell array for the joint objects. We do this by iterating through the DH parameters and, for each *i=1,2,...,6,* creating a rigidBody object and a rigidBodyJoint with unique names. We use the “addBody” function to attach each body to the rigid body tree.

```
robotMDH = rigidBodyTree;
bodiesMDH = cell(6,1);
jointsMDH = cell(6,1);
for i = 1:6
   bodiesMDH{i} = rigidBody(['body' num2str(i)]);
   jointsMDH{i} = rigidBodyJoint(['jnt' num2str(i)],"revolute");

   % Apply the MDH parameters to each joint
   setFixedTransform(jointsMDH{i}, mdhparams(i,:),"mdh");

   bodiesMDH{i}.Joint = jointsMDH{i};

   if i == 1
       % Add first body to base
       addBody(robotMDH, bodiesMDH{i}, "base");
   else
       % Add current body to the previous body
       addBody(robotMDH, bodiesMDH{i}, bodiesMDH{i-1}.Name);
   end
end
```

When using the function “rigidBodyJoint”, we used “revolute” as the last argument (highlighted above) in the case of the Puma560 robot. For others with prismatic joints, the last argument of would be “prismatic”.

We used the modified DH convention in Example 3, but you can also use the classical DH to define a rigidBodytree. When using the classical convention, the last argument of the “setFixedTransform” function would be “dh” instead of “mdh”.

Once the robot is created, you can verify that it has been built properly by using the “showdetails” function. The showdetails function lists all the bodies of the robot in the MATLAB® command window.

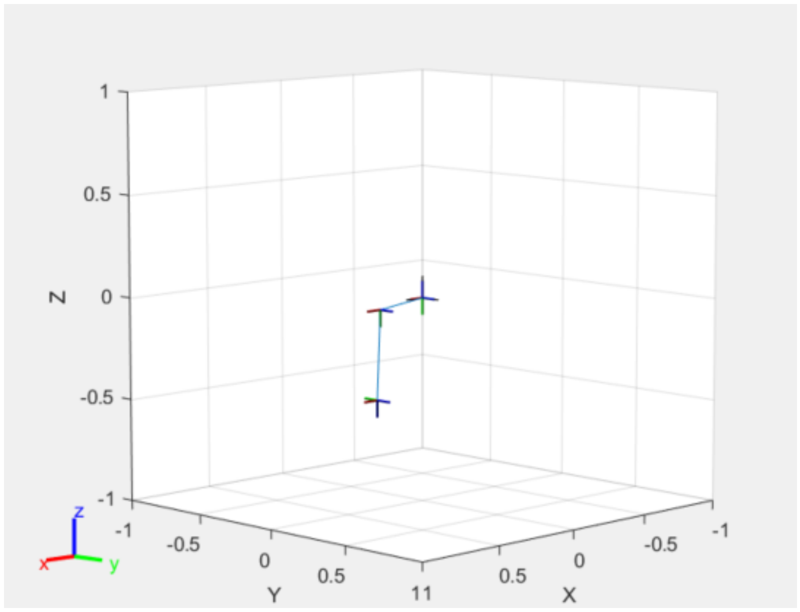
```
>> showdetails(robotMDH)
--------------------
Robot: (6 bodies)

 Idx    Body Name   Joint Name   Joint Type    Parent Name(Idx)   Children Name(s)
 ---    ---------   ----------   ----------    ----------------   ----------------
   1        body1         jnt1     revolute             base(0)   body2(2)
   2        body2         jnt2     revolute            body1(1)   body3(3)
   3        body3         jnt3     revolute            body2(2)   body4(4)
   4        body4         jnt4     revolute            body3(3)   body5(5)
   5        body5         jnt5     revolute            body4(4)   body6(6)
   6        body6         jnt6     revolute            body5(5)
--------------------
```

Visualizing a Robot

Once a robot is created, we can visualize it using the “show” function. This function, “show(robot,configuration)” is used to display the robot at a specified configuration, which is a vector of joint angles (revolute) or displacements (prismatic). If we use this function without the configuration argument, i.e., as “show(robot)”, it shows the robot at its home configuration. The joint variables that correspond to the home configuration of a robot can be obtained by using the function “homeConfiguration(robot)”.

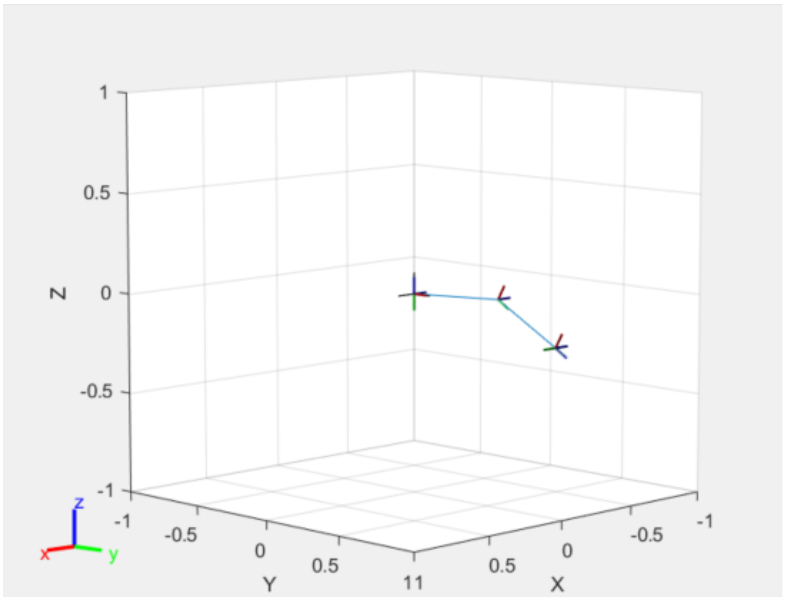
<u>Example 4:</u> We visualize the robot in Example 3 at its home configuration.

```
>> show(robotMDH)
```



Example 5: We visualize the robot in Example 3 at joint configuration [pi/2 0 -pi/3 0 0   0].

```
% Define the new configuration
newconfig = [pi/2; 0; -pi/3; 0; 0; 0];
% Create a configuration structure for the
robot
config = homeConfiguration(robotMDH);
% Assign the new joint positions to the
configuration
for i = 1:numel(newconfig)
   config(i).JointPosition = newconfig(i);
end
% Display the robot with the new configuration
figure(Name="PUMA 560 Robot Model with New
Configuration");
show(robotMDH, config);  % Show the robot with the
updated joint positions
```



Obtaining the Transformations Between Frames

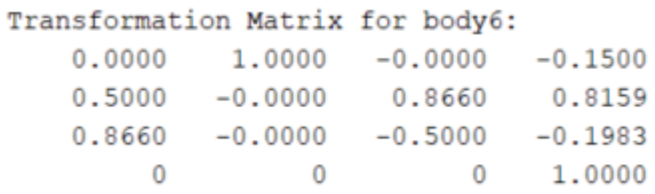
The 4x4 homogeneous transformation matrix, T between any two link-frames can be obtained by using the "getTransform" function:

transform = getTransform(robot,configuration,bodyname) computes the transform that converts points in the bodyname frame to the robot base frame, using the specified robot configuration.

transform = getTransform(robot,configuration,sourcebody,targetbody) computes the transform that converts points from the source body frame to the target body frame, using the specified robot configuration.

Example 6: We obtain the transformation matrix between body6 frame and the base frame for the robot in Example 5 at the following configuration: [pi/2 0 -pi/3 0 0   0].

```
% Get the transformation matrix for the last body ('body6')
newconfig = [pi/2;0;-pi/3;0;0;0];
transform = getTransform(robotMDH, newconfig, 'body6');
disp('Transformation Matrix for body6:');
disp(transform);
```



```
Transformation Matrix for body6:
    0.0000    1.0000   -0.0000   -0.1500
    0.5000   -0.0000    0.8660    0.8159
    0.8660   -0.0000   -0.5000   -0.1983
         0         0         0    1.0000
```