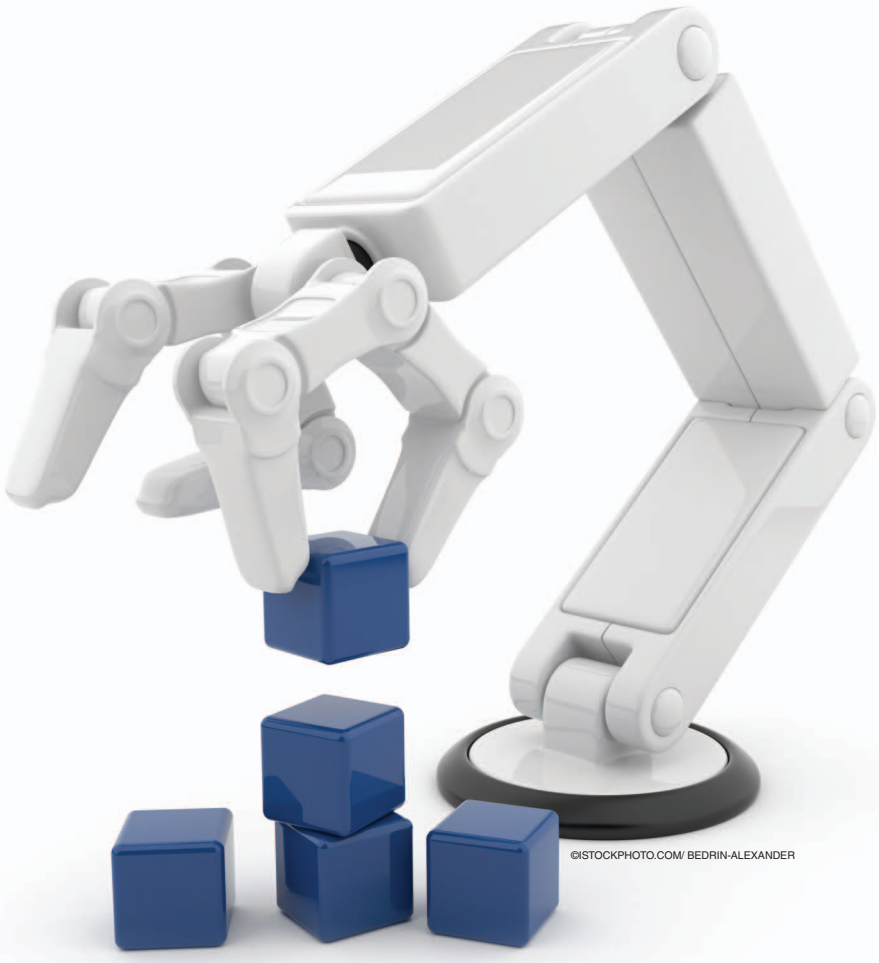
TUTORIAL

Robot Manipulator Capability in MATLAB

A Tutorial on Using the Robotics System Toolbox



©ISTOCKPHOTO.COM/ BEDRIN-ALEXANDER

By Peter Corke

The Robotics System Toolbox for MATLAB provides a wide and growing set of functionalities for creating robotic systems: Robot Operating System (ROS) integration, mobile robotics, and robot manipulator arms. This capability increases with each release and is targeted at industrial developers as well as

Digital Object Identifier 10.1109/MRA.2017.2718418
Date of publication: 4 August 2017

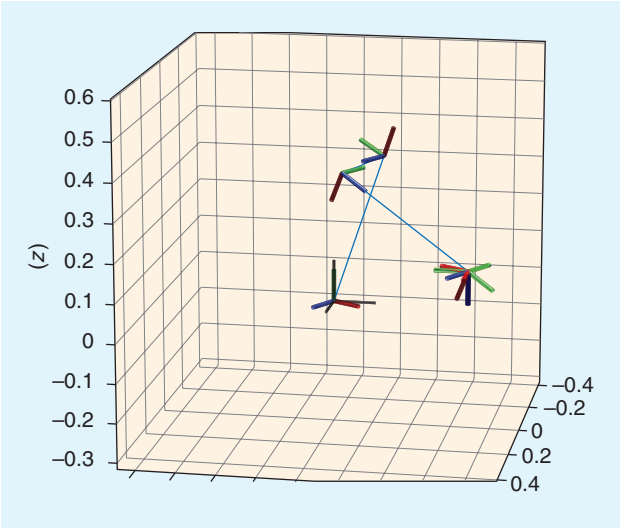


Figure 1. A display of the PUMA 560 robot coordinate frames, produced by the show method of the RigidBodyTree class.

academic teaching and research. This tutorial is concerned with the robot manipulator kinematic functionality that has been available since MATLAB release 2016b.

The Toolbox comes with several robot models, which can be loaded by

```
>> load exampleRobots
```

and includes PUMA 560, KUKA LBR, and Baxter robots, each represented as a tree of rigid bodies by objects of the `RigidBodyTree` class. This class allows general branched-mechanism representation, such as multiarm robots.

The model `puma1` is a standard Denavit–Hartenberg representation of the classical PUMA 560 serial-link manipulator. The joint angles for the home configuration are given by

```
>> q = puma1.homeConfiguration;
```

and the forward kinematics is simply

```
>> puma1.getTransform(q, 'L6')
ans =
    1.0000         0         0    0.4521
    0         1.0000         0   -0.1500
    0              0    1.0000    0.4318
    0              0         0    1.0000,
```

where we specify by name the frame whose pose we wish to compute, and the result is a 4×4 homogeneous transformation matrix. To solve the inverse kinematics, we first create an inverse kinematics solver object:

```
ik = robotics.InverseKinematics
     ('RigidBodyTree', puma1);
```

Next, we define an arbitrary pose for our PUMA robot with its end-effector pointing downward:

```
T = trvec2tform([0.4 0.4 0.2]) *
    eul2tform([0 0 pi], 'ZYX'),
```

and then solve for the joint angles:

```
>> [q, info] = ik('L6', T, ones(1,6),
                  puma1.homeConfiguration);
```

where the passed arguments are the name of the robot frame to solve for, the pose of that frame, the weights applied to the pose error (three components for the orientation error and three for the position error), and the initial solution. The return values are the joint angles and the status of the solution. We can display a graphical representation of the PUMA robot in this configuration by

```
>> puma1.show(q),
```

and the result is shown in Figure 1, where the individual link coordinate frames can be clearly seen.

This short tutorial has barely scratched the surface of the capabilities, which include, from MATLAB release 2017a and onwards: Universal Robotic Description Format (URDF) import, rigid body tree dynamics, and multiconstraint inverse kinematics. More detail can be found in the extended version of this article, including code examples, which accompanies this tutorial in IEEE *Xplore*.

More detail can be found in the extended version of this article, including code examples, which accompanies this tutorial in IEEE *Xplore*.

Peter Corke, Queensland University of Technology, Brisbane, Australia. Acts as a consultant to The MathWorks Inc. E-mail: peter.corke@qut.edu.au.