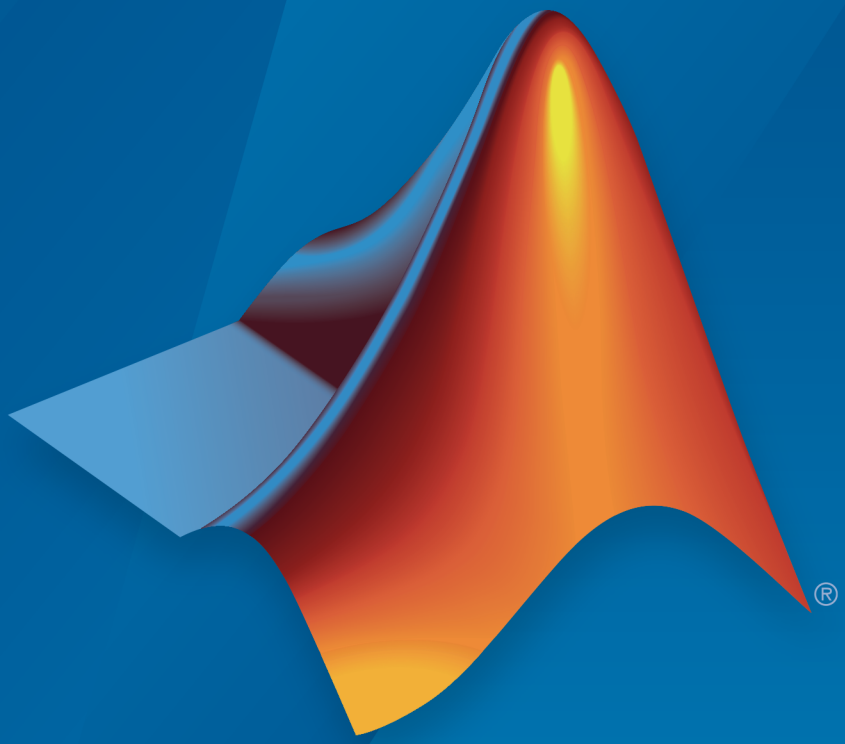
Robotics System Toolbox™

Getting Started Guide



MATLAB® & SIMULINK®

R2015b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

Robotics System Toolbox™ Getting Started Guide

Revision History

March 2015	Online only	New for Version 1.0 (R2015a)
September 2015	Online only	Revised for Version 1.1 (R2015b)

Contents

1 Product Overview

2 Coordinate System Transformations

3 Robot Operating System (ROS)

1

Product Overview

Robotics System Toolbox Product Description

Design and test algorithms for robotics applications

Robotics System Toolbox™ provides algorithms and hardware connectivity for developing autonomous mobile robotics applications. Toolbox algorithms include map representation, path planning, and path following for differential drive robots. You can design and prototype motor control, computer vision, and state machine applications in MATLAB® or Simulink® and integrate them with core algorithms in Robotics System Toolbox.

The system toolbox provides an interface between MATLAB and Simulink and the Robot Operating System (ROS) that enables you to test and verify applications on ROS-enabled robots and robot simulators such as Gazebo. It supports C++ code generation, enabling you to generate a ROS node from a Simulink model and deploy it to a ROS network.

Robotics System Toolbox includes examples showing how to work with virtual robots in Gazebo and actual ROS-enabled robots.

Key Features

- Path planning, path following, and map representation algorithms
- Functions for converting between different rotation and translation representations
- Bidirectional communication with live ROS-enabled robots
- Interface to Gazebo and other ROS-enabled simulators
- Data import from rosbag log files
- ROS node generation from Simulink models (with Embedded Coder®)

2

Coordinate System Transformations

Standard Units for Robotics System Toolbox

Robotics System Toolbox uses a fixed set of standards for units to ensure consistency across algorithms and applications. Unless specified otherwise, functions and classes in this toolbox represent all values in units based on the International System of Units (SI). The table below summarizes the relevant quantities and their SI derived units.

Quantity	Unit (abbrev.)
Length	meter (m)
Time	second (s)
Angle	radian (rad)
Velocity	meter/second (m/s)
Angular Velocity	radian/second (rad/s)
Acceleration	meter/second^2 (m/s^2)
Angular Acceleration	radian/second^2 (rad/s^2)
Mass	kilogram (kg)
Force	Newton (N)
Torque	Newton-meter (N-m)
Moment of Inertia	kilogram-meter^2 (kg-m^2)

Coordinate Transformations in Robotics

In this section...

In robotics applications, many different coordinate systems can be used to define where robots, sensors, and other objects are located. In general, the location of an object in 3-D space is defined by its position and orientation. There are multiple possible representations for these quantities, some of which are specific to certain applications. Translation and rotation are alternative terms for position and orientation. Robotics System Toolbox supports representations that are commonly used in robotics and allows you to convert between them. You can transform between coordinate systems when you apply these representations to 3-D points. These supported representations are detailed below with brief explanations of their usage and numeric equivalent in MATLAB. Each representation has an abbreviation for its name. This is used in the naming of arguments and conversion functions that are supported in this toolbox.

At the end of this section, you can find out about the conversion functions that we offer to convert between these representations.

Robotics System Toolbox assumes that positions and orientations are defined in a right-handed Cartesian coordinate system.

Axis-Angle

Abbreviation: axang

A rotation in 3-D space described by a scalar rotation around a fixed axis defined by a vector.

Numeric Representation: 1-by-3 unit vector and a scalar angle combined as a 1-by-4 vector

For example, a rotation of `pi/2` radians around the *y*-axis would be:

```
axang = [0 1 0 pi/2]
```

Euler Angles
Abbreviation: eul

Euler angles are three angles that describe the orientation of a rigid body. Each angle is a scalar rotation around a given coordinate frame axis. The Robotics System Toolbox supports two rotation orders. The `'ZYZ'` axis order is commonly used for robotics applications. We also support the `'ZYX'` axis order which is also denoted as "Roll Pitch Yaw (rpy)." Knowing which axis order you use is important for apply the rotation to points and in converting to other representations.

Numeric Representation: 1-by-3 vector of scalar angles

For example, a rotation around the *y* -axis of pi would be expressed as:

```
eul = [0 pi 0]
```

Note: The axis order is not stored in the transformation, so you must be aware of what rotation order is to be applied.

Homogeneous Transformation Matrix
Abbreviation: tform

A homogeneous transformation matrix combines a translation and rotation into one matrix.

Numeric Representation: 4-by-4 matrix

For example, a rotation of angle α around the *y* -axis and a translation of 4 units along the *y* -axis would be expressed as:

```
tform =
 cos α  0      sin α  0
 0      1      0      4
-sin α  0      cos α  0
 0      0      0      1
```

You should **pre-multiply** your transformation matrix with your homogeneous coordinates, which are represented as a matrix of row vectors (*n*-by-4 matrix of points). For example:

```
points = rand(100,4);
tformPoints = tform*points;
```

Quaternion
Abbreviation: quat

A quaternion is a four-element vector with a scalar rotation and 3-element vector. Quaternions are advantageous because they avoid singularity issues that are inherent in other representations. The first element, *w*, is a scalar to normalize the vector with the three other values, *[x y z]* defining the axis of rotation.

Numeric Representation: 1-by-4 vector

For example, a rotation of `pi/2` around the *y* -axis would be expressed as:

```
quat = [0.7071 0 0.7071 0]
```

Rotation Matrix
Abbreviation: rotm

A rotation matrix describes a rotation in 3-D space. It is a square, orthonormal matrix with a determinant of 1.

Numeric Representation: 3-by-3 matrix

For example, a rotation of **α** degrees around the *x*-axis would be:

```
rotm =

     1     0         0
     0     cos α     -sin α
     0     sin α     cos α
```

You should **pre-multiply** your rotation matrix with your coordinates, which are represented as a matrix of row vectors (*n*-by-3 matrix of points). For example:

```
points = rand(100,3);
rotPoints = rotm*points;
```

Translation Vector
Abbreviation: trvec

A translation vector is represented in 3-D Euclidean space as Cartesian coordinates. It only involves coordinate translation applied equally to all points. There is no rotation involved.

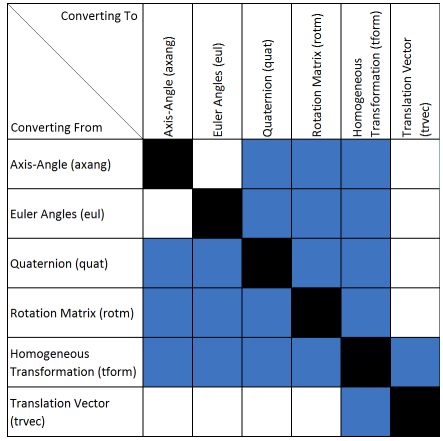
Numeric Representation: 1-by-3 vector

For example, a translation by 3 units along the x -axis and 2.5 units along the z -axis would be expressed as:

```
trvec = [3 0 2.5]
```

Conversion Functions and Transformations

Robotics System Toolbox provides conversion functions for the previously mentioned transformation representations. Not all conversions are supported by a dedicated function. Below is a table showing which conversions are supported (in blue). The abbreviations for the rotation and translation representations are shown as well.



The names of all the conversion functions follow a standard format. They follow the form `alpha2beta` where `alpha` is the abbreviation for what you are converting from and `beta` is what you are converting to as an abbreviation. For example, converting from Euler angles to quaternion would be `eul2quat`.

All the functions expect valid inputs. If you specify invalid inputs, the outputs will be undefined.

There are other conversion functions for converting between radians and degrees, Cartesian and homogeneous coordinates, and for calculating wrapped angle differences. For a full list of conversions, see “Coordinate System Transformations”.

3

Robot Operating System (ROS)

Robot Operating System (ROS)

Robot Operating System (ROS) is a framework of tools, libraries, and software to aid in robot software development. It is a flexible system for programming robots and controlling robotic platforms. ROS was developed by an open-source collaborative community to help grow the world of robotics. Applications for working with hardware, robotic simulation models, path planning, localization and mapping, and many other algorithms are available. For an introduction to ROS, see the ROS Introduction on their website.

For more information about ROS and its functionality, see the ROS Website and the ROS Wiki. The wiki contains documentation and tutorials for ROS, software packages, core libraries, and supported robots and hardware.

Robotics System Toolbox allows you to access ROS functionality in MATLAB. Use MATLAB to communicate with a ROS network, interactively explore robot capabilities, and visualize sensor data. You can develop robotics applications by exchanging data with ROS-enabled robots and robot simulators such as Gazebo. You can also create Simulink models that exchange messages with a ROS network. Verify your model within the Simulink environment by receiving messages from, and sending messages to, ROS-enabled robots and robot simulators. From your model, you can also generate C++ code for a standalone ROS application.

The first thing to do when working with ROS is to set up or connect to a ROS network. Here is a link to an explanation of the ROS network setup and some examples to get started using ROS in MATLAB and Simulink:

MATLAB

- “ROS Network Setup”
- “Getting Started with ROS”
- “Connecting to a ROS Network”

Simulink

- “Getting Started with ROS in Simulink®”
- “Configuring ROS Network Addresses”
- “Connecting to a ROS-enabled Robot from Simulink®”