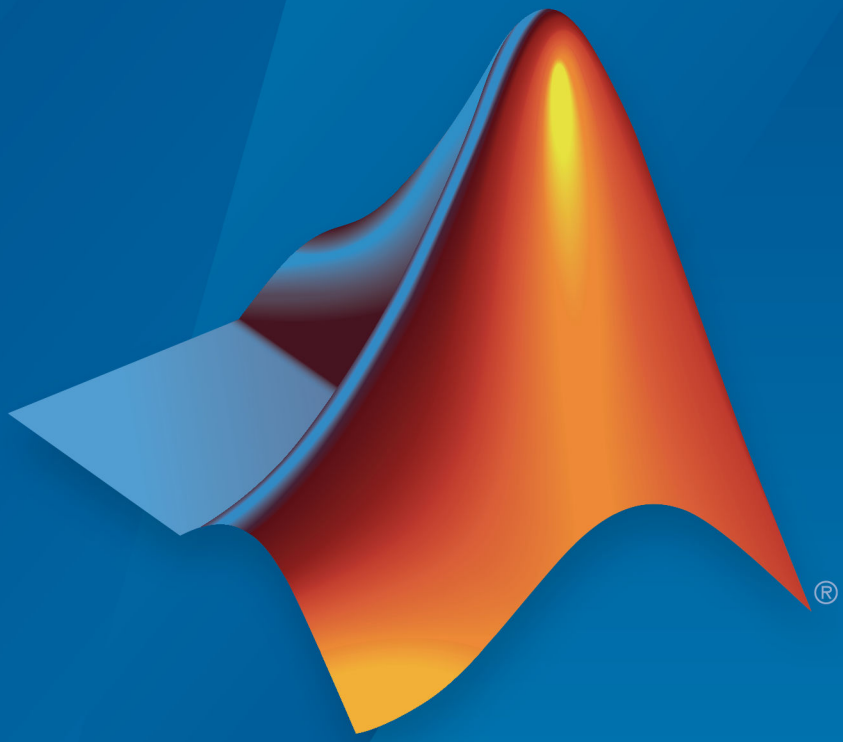
# Robotics System Toolbox™ Release Notes



MATLAB® & SIMULINK®

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 ™ Release Notes*

# Contents

## R2018a



## R2017b

## R2017a



## R2016b

## R2016a

## R2015aSP1

**Bug Fixes**

## R2015b



## R2015a

# R2018a

**Version: 2.0**

**New Features**

**Bug Fixes**

## Manipulator Algorithm Blocks: Compute rigid body tree kinematics and dynamics in Simulink

Simulink® now supports dynamics and kinematics functions for rigid body trees. The following blocks use an associated rigid body tree model, specified as a `RigidBodyTree` object, to compute the kinematic or dynamic values for the robot:

- Inverse Dynamics: Required joint torques for given motion
- Forward Dynamics: Joint accelerations given joint torques and states
- Get Transform: Get transform between body frames
- Get Jacobian: Geometric Jacobian for robot configuration
- Gravity Torque: Joint torques that compensate for gravity
- Joint Space Mass Matrix: Joint-space mass matrix
- Velocity Product Torque: Joint torques that cancel velocity-induced forces

## Lidar-Based SLAM: Localize robots and build map environments using lidar sensors

The `LidarSLAM` class uses lidar sensor data and robot poses to simultaneously localize the robot and build a map. The class uses an underlying `PoseGraph` class that contains pose estimates for lidar scan readings. Lidar scans are incrementally added to the `lidarScan` object. The object detects loop closures and optimizes pose graphs as scans are added to the map. A grid-based, scan-matching algorithm determines placement in the map and detects loop closures.

## Pose Graph Data Structure and Optimization: Represent and optimize 2-D and 3-D pose graphs

The `PoseGraph` and `PoseGraph3D` classes store pose graph data with pose estimates and information matrices to specify the uncertainty. The data is represented as nodes and edges connecting the different poses to draw out a robot trajectory. Nodes represent the pose estimates, and edges contain the relative pose differences between nodes and information matrices as edge constraints. Loop closure edges link existing nodes together as a relative pose difference.

The `optimizePoseGraph` function optimizes the entire graph using the edge constraints. The function attempts to balance the relative poses and their edge constraints across the whole graph. The option to ignore specific loop closures is also available.

The `LidarSLAM` class uses the 2-D `PoseGraph` class for simultaneous localization and mapping based on lidar scan data.

## 3-D Occupancy Maps: Map 3-D environments using efficient octree data structure

The `OccupancyMap3D` class supports the mapping of 3-D environments using probabilities to represent occupancy of locations. The class stores the occupancy map using an efficient octree structure to minimize data storage and dynamically prunes the tree appropriately. Sensor observations are added as point clouds using `insertPointCloud` to incrementally build a map that you can `show` in a figure. Also, you can `inflate` the map for obstacle avoidance and navigation.

## Enhanced Performance for rosbag Logfiles: Load rosbags faster and extract message data as structures

Performance improvements for reading rosbags enable faster load times using the `rosbag` function. ROS messages can now be returned as a cell array of structures instead of ROS message objects using `readMessages`, which allows for easier access of fields in the message and direct access to custom message data:

```
bag = rosbag('ros_turtlesim.bag');
msgStructs = readMessages(bSel,'DataFormat','struct');
```

The `getTransform` and `canTransform` functions now support accessing transformations from rosbags. You can also query statistics about the rosbag using `rosbag info` *`fileName`*.

# R2017b

**Version: 1.5**

**New Features**

**Bug Fixes**

## RigidBodyTree Visualization Improvements: Attach mesh files and inspect individual bodies in a MATLAB figure

The `robotics.RigidBodyTree` class's `show` method can now display visual meshes in a figure window. `addVisual` can assign an individual mesh file (`.stl`) to a rigid body a mesh file, or you can use `importrobot` with a Unified Robotics Description Format (URDF) file that has mesh files associated with bodies.

Other improvements to the visualization include inspection of individual body properties and toggling of individual visual elements of the rigid body tree using mouse interaction.

## Coordinate Transformation Conversion Block: Convert between coordinate system representations in Simulink

You can now convert between different coordinate system representations using the Coordinate Transformation Conversion block. The supported representations are:

- Axis-Angle (`AxAng`) – `[x y z theta]`
- Euler Angles (`Eul`) – `[z y x]`, `[z y z]`, or `[x y z]`
- Homogeneous Transformation (`TForm`) – 4-by-4 matrix
- Quaternion (`Quat`) – `[w x y z]`
- Rotation Matrix (`RotM`) – 3-by-3 matrix
- Translation Vector (`TrVec`) – `[x y z]`

For more information about the different coordinate transformation representations and the equivalent MATLAB® functions, see Coordinate System Transformations.

## ROS Image and Point Cloud Blocks: Convert ROS messages to nonbus signals in Simulink

You can now use Robotics System Toolbox to convert Robot Operating System (ROS) `Image`, `CompressedImage`, and `PointCloud2` messages to nonbus signals in Simulink. The image or point cloud data are output as an array. Subscribe to a ROS message using Subscribe and feed the output bus to the Read Image or Read Point Cloud block to convert the message to an array signal. You can configure the block from a topic on a live ROS network or specify message parameters manually.

## Lidar Sensor Object: Store and use lidar scan data

The `lidarScan` object can store data from a lidar (light detection and ranging) scan. A lidar scan, also called a laser scan, contains ranges and angles from a sensor to measure and map your environment. This object contains sensor information and the data collected from an individual scan. You can use this object with other Robotics System Toolbox functionality that previously used `ranges` and `angles` as inputs:

- `matchScans`
- `transformScan`
- `robotics.MonteCarloLocalization`
- `robotics.VectorFieldHistogram`
- `robotics.OccupancyGrid.insertRay`

You can also convert `LaserScan` ROS messages to the `lidarScan` object.

## Scan Matching: New trust-region solver and code generation support

You can now use the `'trust-region'` solver for the `matchScans` function. This solver does not require an Optimization Toolbox™ license and replaces the `'fminunc'` solver as the default. Code generation for the `'trust-region'` solver with MATLAB Coder™ is now available as well.

To use a specific algorithm, specify the `'SolverAlgorithm'` name-value pair:

```
pose = matchScans( ___ ,'SolverAlgorithm','trust-region')
```

# R2017a

**Version: 1.4**

**New Features**

**Bug Fixes**

## External Mode Support: Tune parameters and view signal values of deployed ROS nodes over TCP/IP (with Simulink Coder)

You can now use external mode with your deployed ROS nodes. External mode enables you to tune parameters and log signals while code is running on the target hardware. You must have Simulink Coder installed.

To deploy a standalone ROS node, see Generate a Standalone ROS Node from Simulink.

To use external mode, see Enable External Mode for Robotics System Toolbox Models

## Dynamics for Robot Manipulators: Solve inverse and forward dynamics for RigidBodyTree objects

The `RigidBodyTree` class provides dynamics information for robot manipulators. For each `RigidBody` object, you can specify the following properties:

- `Mass` — Total mass of rigid body
- `CenterOfMass` — Location of body's center of mass in the body frame
- `Intertia` — Independent elements of the inertia tensor in the body frame

You can also specify the `Gravity` property for the entire `RigidBodyTree` object.

New object functions are available for solving inverse and forward dynamics and for calculating other relevant values for the robot model:

- `forwardDynamics` — Compute the resulting joint accelerations for given joint torques, joint positions, and velocities. You can also specify external forces to the robot model by using `externalForce`.
- `inverseDyanmics` — Compute the required joint torques for given joint positions, velocities, and accelerations (robot motion). You can also specify external forces on the robot model.
- `externalForce` — Create external forces to apply to a robot model. This function creates a matrix that `forwardDyanmics` and `inverseDynamics` use as an input.
- `massMatrix` — Compute the joint-space mass matrix for a certain robot configuration.
- `velocityProduct` — Compute the joint torques that compensate for Coriolis and centrifugal terms for given joint positions and velocities.

- `gravityTorque` — Compute the joint torques required to compensate for gravity for a certain robot configuration.
- `centerOfMass` — Compute the center of mass position and center of mass Jacobian for a certain robot configuration in the base frame.

## Generalized Inverse Kinematics: Solve multiconstrained inverse kinematics for robot manipulators

Find a robot configuration on a `RigidBodyTree` model given one or more constraints. The `InverseKinematics` class previously supported a single end-effector pose constraint. The `GeneralizedInverseKinematics` class supports multiple constraints with different types.

Specify constraint types when creating the object, and specify the constraint parameters when calling the object. You can create constraint inputs from these classes:

- `AimingConstraint`
- `CartesianBounds`
- `JointPositionBounds`
- `OrientationTarget`
- `PoseTarget`
- `PositionTarget`

## URDF File Importer: Import URDF robot descriptions as a RigidBodyTree object

You can now import rigid body tree models from the Unified Robot Description Format (URDF) robot description using `importrobot`. The function parses the URDF information and returns a `RigidBodytree` object.

## Scan Matching: Calculate pose difference between laser scans

Use the `matchScans` function to calculate the pose difference between two laser scans. This pose difference, given as `[x y theta]`, is used to correlate scans together and transform them into the same coordinate frame. To transform laser scans based on this pose difference, use the `transformScan` function.

## Code Generation for RigidBodyTree objects: Generate code with robot manipulator data structures

Code generation with MATLAB Coder is now available for `RigidBodyTree` objects. You can generate code for all inverse and forward dynamics algorithms, but not for the `show` and `showdetails` methods.

## rosparam Simplified Commands: Modify ROS parameters using a simplified interface without creating a ParameterTree object

You can now set, get, list, and delete ROS parameters directly using the `rosparam` function. Previously, you had to create a ROS ParameterTree object to modify parameter values. `rosparam` has simplified commands that mimic ROS behavior. For example, to set the `'/param_value'` ROS parameter to the value, `42.15`, use:

```
rosparam set /param_value 42.15
```

# R2016b

## Robotic Manipulator Algorithms: Represent robot manipulators using a rigid body tree and calculate forward and inverse kinematics

The `robotics.RigidBodyTree` class enables you to build kinematic chains or trees using rigid bodies to represent physical robots. You can add or modify bodies on a structure, specify joint limits, and replace bodies or joints. In addition, you can use forward kinematics to get transformations between two body frames and compute geometric Jacobians for specified end effectors for a given robot configuration.

Inverse kinematics is available in the `robotics.InverseKinematics` class. Use inverse kinematics to calculate corresponding joint angles for desired end-effector positions.

## Automated Deployment of ROS Nodes: Automatically deploy ROS nodes to target hardware using Simulink Coder

You can now automatically deploy and run ROS nodes using Simulink Coder. Create a Simulink model using Robotics System Toolbox blocks and deploy it to your target Linux device that has ROS installed. You can use the `rosdevice` object to connect to the target device and run or stop the deployed ROS nodes.

For more information, see Generate a Standalone ROS Node from Simulink®.

## Occupancy Grid Class: Build a robot environment using a 2-D occupancy map with probabilistic values

The `robotics.OccupancyGrid` class enables you to create 2-D occupancy maps using probabilistic values. You can incorporate probabilistic sensor information using Bayes' rule. Also, you can use the occupancy grid with the `robotics.PRM` and `robotics.MonteCarloLocalization` classes for path planning and localization.

## Mobile Robot Algorithm Blocks: Perform obstacle avoidance and path following in Simulink

You can now use the Vector Field Histogram and Pure Pursuit algorithms with Simulink. The Pure Pursuit block outputs a target direction, which you can feed directly into the Vector Field Histogram block to perform obstacle avoidance with path following.

## ROS Action Client: Send action goals via a ROS network and get feedback on their execution

By setting up a simple action client using the `rosactionclient` function, you can now perform predefined actions that are available on the ROS network. Once an action is triggered, the client receives asynchronous feedback about a specified goal and can preempt the execution of goals on the server.

## Buffered ROS tf2 Transformations: Access time-buffered transformations from the ROS transformation tree

The ROS transformation tree now supports time-buffered transformation. By default, the `TransformationTree` object has a time buffer of 10 seconds. After creating a transformation tree using `rostf`, transformations are saved based on the buffer time. You can call `getTransform` or `transform` to access and apply the transformations at a specified source time. A new function, `canTransform`, enables you to check if the transformation is available.

## Compatibility Considerations

`waitForTransform` will be removed in a future release. Use `getTransform` with a specified `timeout` instead. To wait indefinitely, specify `timeout` as `inf`.

The behavior of `getTransform` will change in a future release. The function will no longer return an empty transform when the transform is unavailable and no `sourcetime` is specified. If `getTransform` waits for the specified timeout period and the transform is still not available, the function returns an error. The timeout period is 0 by default.

## Odometry Motion Model Class: Predict poses for a differential drive robot

The `robotics.OdometryMotionModel` class contains the equations of motion that govern a differential drive robot. The odometry motion model predicts the motion of a robot based on previous poses and noise parameters. You can tune the `Noise` property and see the effect on particle distributions using the `showNoiseDistribution` function. You can also use this motion model with `robotics.MonteCarloLocalization` to localize robots in a known environment.

## ROS Time and Duration: Use mathematical operations on ROS time and duration objects

In the `rostime` function, you can now specify second and nanosecond scalar inputs when creating a ROS `Time` message object. You can also use the new `rosduration` function to create a ROS `Duration` message object. Both message types support mathematical operations and comparisons. For example:

Create a ROS `Time` and `Duration` object and add them together. Compare the two `Time` objects.

```
time = rostime(5.54);
duration = rosduration(2);
time2 = time + duration

time2 =

  ROS Time with properties:

     Sec: 7
    Nsec: 540000000

time2 <= time

ans =

   0
```

## Code Generation for Robotics Algorithms: Generate code for select algorithms

Code generation with MATLAB Coder is now available for the following algorithms:

- `robotics.BinaryOccupancyGrid`
- `robotics.OccupancyGrid`
- `robotics.OdometryMotionModel`
- `robotics.PRM` — The `map` input must be specified on creation of the `PRM` object.
- `robotics.PurePursuit`

For a full list of code generation support for Robotics System Toolbox, see Code Generation.

# R2016a

**Version: 1.2**

**New Features**

**Compatibility Considerations**

## Monte Carlo Localization Algorithm: Estimate robot location in a known map

Monte Carlo Localization utilizes a particle filter to localize a robot in a known environment. You can supply a `BinaryOccupancyGrid` object of your map and range sensor data from the robot to the `robotics.MonteCarloLocalization` object to estimate the pose (location and orientation) of the robot. You have the option of using global localization or specifying an initial pose to improve performance. As sensor data is supplied to the algorithm, particles converge on the best estimate of the robot location.

## Particle Filter Algorithm: Estimate state for nonlinear systems

The `robotics.ParticleFilter` class enables you to create a particle filter for state estimation. The algorithm uses particles and sensor data to try to match the posterior distribution of the current state. It first predicts the current state based on a given system model and then corrects the estimate based on sensor data inputs. You can specify a fixed number of particles to use, number of state variables to estimate, and your method for final estimation based on the particle weights. You can customize your particle filter by giving a state transition function and measurement likelihood model to match your system.

## Fixed-Rate Execution: Run MATLAB code at a constant rate

Execute loops at a constant rate based off either your system time or ROS time. By creating a `robotics.Rate` object, you can call `waitfor` to pause a loop until the next time step. This feature ensures that loops are run at a fixed rate when accurate timing of commands is required.

You can also use `rosrate` to base timing off the current time published in a ROS network. Therefore, messages and control commands can be published at a fixed rate to a ROS-enabled system.

## Robotics System Toolbox Support Package for TurtleBot based Robots: Connect to TurtleBot hardware

Robotics System Toolbox Support Package for TurtleBot® Based Robots allows robotics researchers to acquire sensor data from TurtleBot-based robots (either simulated or

physical robots). You can use the data for visualization and analysis, and send commands to control the robots.

## String support for ROS parameters in Simulink

Support for using strings as ROS parameters is now available in Simulink. When using strings, they must be cast as a `uint8` array of ASCII values. See ROS String Parameters for more information.

## String array support for ROS messages in Simulink

You can now use an array of strings when using the Publish, Subscribe, and Blank Message blocks to create, send, and receive messages using a ROS network in Simulink.

## Code generation from Simulink using Simulink Coder

You can now generate standalone ROS nodes from Simulink models with just Simulink Coder. If you have Embedded Coder®, you can customize the generated code with additional optimization, readability, and code configuration options.

## roboticsSupportPackages function replaced with roboticsAddons

The `roboticsSupportPackages` function is no longer available. Instead, use `roboticsAddOns` to access Add-ons for Robotics System Toolbox.

# R2015aSP1

**Version: 1.0.1**

**Bug Fixes**

# R2015b

**Version: 1.1**

**New Features**

## Vector Field Histogram Plus (VFH+) obstacle avoidance algorithm

The VFH+ obstacle avoidance algorithm is a reactive algorithm that calculates obstacle-free robot movements using range sensor information. You can use this algorithm to have your robot avoid unknown obstacles while driving through dynamic or partially known environments. See `robotics.VectorFieldHistogram` for more information.

## Access to ROS parameters from Simulink

Simulink workflows now support ROS parameters. You can get and set parameter values using the new Get Parameter and Set Parameter blocks.

## Code generation for coordinate transforms and select robotics algorithms

For select Robotics System Toolbox algorithms, you can now generate C/C++ code using MATLAB Coder. You can create MEX-files and shared libraries from your MATLAB application. These code generation workflows are supported for the coordinate transformation functions (Coordinate System Transformations), the VFH+ obstacle avoidance algorithm, and the Pure Pursuit controller algorithm (`robotics.PurePursuit`). See Code Generation for more information.

# R2015a

**Version: 1.0**

**New Features**

## Path planning, path following, and map representation algorithms

The Robotics System Toolbox provides algorithms for path planning, path following, and map representations. The support in this release includes classes for Binary Occupancy Grids, Probabilistic Roadmaps (PRM), and a Pure Pursuit controller.

## Functions for converting between different rotation and translation representations

Coordinate system transformations are provided as functions for converting between many different representations including quaternions, rotation matrices, homogeneous transformation matrices, and Euler angles. Other functions are available for converting between radians and degrees and for angle calculations. For more information, see Coordinate System Transformations.

## Bidirectional communication with live ROS-enabled robots

Communication with ROS using publishers and subscribers is available in MATLAB and Simulink. Many message types are readily supported. Robotics System Toolbox can also access ROS services, the parameter server, and the tf transformation tree in MATLAB.

## Interface to Gazebo and other ROS-enabled simulators

ROS-enabled simulators allow prototyping of algorithms and testing systems developed in MATLAB. Connection to a Gazebo simulator is supported with an example interacting with the simulator shown here: Reading Model and Simulation Properties from Gazebo.

## Data import from rosbag log files

This release of the Robotics System Toolbox includes the ability to access rosbags, which are logfiles from ROS. You can access whole log files or portions and manipulate the data as desired (see Working with rosbag Logfiles).

## ROS node generation from Simulink models (with Embedded Coder)

This release includes ROS node generation using Simulink. You can use Simulink to create models that exchange messages with a ROS network. Using Embedded Coder, you can generate C++ code for standalone ROS nodes from these models.