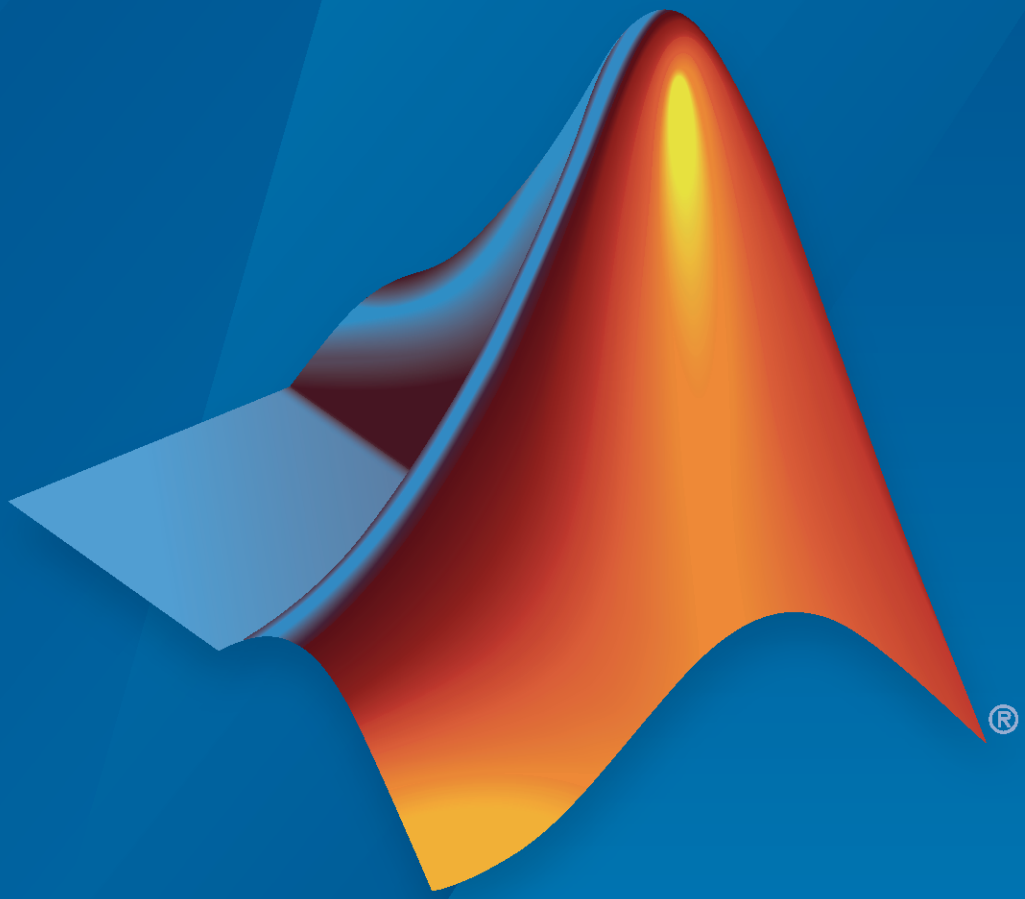
Robotics System Toolbox™ Release Notes



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.
1 Apple Hill Drive
Natick, MA 01760-2098

Robotics System Toolbox™ Release Notes

© COPYRIGHT 2015–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Contents

R2022b

Spatial Math: Use SE(2), SE(3), SO(2), and SO(3) transformations **1-2**

Custom Sensor Support in Robot Scenarios: Add custom sensor models to robots in simulated scenarios **1-2**

Time Optimal Trajectory: Create time optimal trajectory subject to kinematic constraints **1-2**

Capsule Collision Geometry: Use additional collision capsule geometry **1-2**

Collision Checking: Check collisions using collision capsules **1-3**

Capsule Approximation: Approximate rigid body trees using capsules . . **1-3**

Robot Models: Use additional manipulator and gripper rigid body tree robot models introduced to robot model library **1-4**

Self-Collision Checking: Alter rigid body tree self-collision checking behavior change, new default self-collision checking behavior **1-4**

Gazebo Co-Simulation: Gazebo co-simulation enhancements **1-5**

Gazebo Co-Simulation: Gazebo Read block sensor read performance improvements **1-6**

waypointTrajectory Reverse Motion: Specify wait and reverse motion for waypoint trajectory **1-6**

Support Package Updates **1-6**

New Examples **1-7**

R2022a

Robot Scenarios and Sensor Models: Author robot scenarios and simulate sensor readings for robotics applications **2-2**

Inverse Kinematics Designer: Design inverse kinematics solvers, configurations, and waypoints **2-2**

Convert Collision Mesh: Convert primitive collision objects to collision mesh **2-3**

Kinematic Constraints: Additional Generalized Inverse Kinematics Constraints **2-3**

INS Sensor Model: Simulate inertial navigation and GPS readings **2-3**

GPS Sensor Model: Simulate GPS receiver readings **2-3**

Trajectory and Waypoint Following Algorithm: Use built-in algorithm to generate trajectories and control commands for robots **2-3**

Transform Tree Object: Define coordinate frames and relative transformations **2-3**

Point Cloud Object: Store 3-D point clouds **2-3**

Trajectory Generation Blocks: Create minimum jerk and minimum snap polynomial trajectories with Simulink **2-4**

Commercial Robot Models: Universal Robots E-Series models introduced to the library of robot models **2-4**

Simulation 3-D Environment Upgrade: Gazebo 11 support **2-4**

New Examples **2-4**

R2021b

State Space and State Validation for Robot Manipulator Models **3-2**

Ignore Self Collisions for Manipulator RRT Path Planning **3-2**

Minimum Jerk and Snap Polynomial Trajectories **3-2**

Simulation Description Format (SDF) Support **3-2**

COLLADA Mesh Support **3-2**

Load Robot Function Update **3-2**

R2021a

Workspace Regions For Motion Planning: Specify a workspace region to sample end-effector poses for path planning **4-2**

Gazebo Co-Simulation in MATLAB: Access and modify Gazebo model parameters **4-2**

Educational Robot Model: Additional rigid body tree robot model for manipulator introduced to the library of robot models **4-2**

Rigid Body Tree Visualization Improvements **4-2**

Rigid Body Tree Function Generation **4-3**

R2020b

RRT Planner for Manipulators: Plan collision-free motion for rigid body tree robot models **5-2**

Collision Checking for Robot Meshes: Add collision meshes to rigid body tree models and check collisions for robot configurations **5-2**

Custom Messages with Gazebo: Publish and subscribe to custom message types in a Gazebo simulation **5-2**

Joint and Link States in Gazebo: Send and receive messages for robot joint and link states in a Gazebo Simulation **5-2**

Analytical Inverse Kinematics: Generate functions for inverse kinematics solutions using closed-form solutions **5-2**

Educational and Commercial Robot Models: Additional rigid body tree robot models for manipulators and mobile robots introduced to the library of robot models **5-3**

Robotics System Toolbox UAV Library add-on is now the UAV Toolbox . . . **5-3**

Robotics System Toolbox Support Package for Manipulators Add-on **5-3**

R2020a

Interactive Robot Visualization: Manipulate rigid body tree models with visual meshes and perform inverse kinematics for target bodies **6-2**

Commercial Robot Models: Load additional rigidBodyTree robot models for manipulators and mobile robots added to our library of existing models **6-2**

Code Generation for Collision Checking: Generate C/C++ code using the checkCollision function and collision geometries **6-2**

Simscape Multibody Model Parameters: Import models with initial position and joint limits **6-2**

R2019b

Gazebo Co-simulation: Perform time-synchronized simulation of Gazebo with Simulink **7-2**

Robot Motion Modeling and Simulation: Simulate mobile robot kinematics and closed-loop manipulator dynamics **7-2**

Collision Checking: Define collision shapes and detect collisions between mesh geometries **7-2**

Commercial Robot Models: Use a provided library of rigid body robot models to quickly model your robot applications **7-3**

Robot Application Examples: Get started with reference examples for pick-and-place robots and warehouse mobile robots **7-3**

Robotics System Toolbox Support Package for Turtlebot-Based Robots functionality has moved **7-3**

Robotics System Toolbox has transitioned into Robotics System Toolbox, Navigation Toolbox, and ROS Toolbox **7-3**

R2019a

SLAM Map Builder Sessions: Save and load app sessions **8-2**

MAVLink Protocol Support: Communicate with UAVs using MAVLink messages and load log files **8-2**

Trajectory Generation: Create piecewise polynomials, trapezoidal velocity profiles, B-splines, and task-space interpolation **8-2**

Model Reference Support for ROS: Use model reference in models with ROS Blocks **8-2**

Orbit Follower for UAVs: Follow a circular path around a point of interest **8-2**

Code Generation for SLAM: Generate code using LidarSLAM, PoseGraph, and PoseGraph3D objects **8-2**

R2018b

SLAM Map Builder App: Build and tune a 2-D grid map with lidar-based SLAM **9-2**

UAV Algorithms: Create UAV guidance models and 3-D path following for fixed-wing and multirotor UAVs **9-2**

Read Data Block: Play back data from a rosbag logfile in Simulink **9-2**

Inverse Kinematics Block: Calculate joint configurations for a desired end-effector pose in Simulink **9-2**

ROS Service and Current Time Blocks: Call ROS services and get the current ROS time in Simulink **9-2**

Simscape Multibody Data Exchange: Use importrobot to import Simscape Multibody models to a RigidBodyTree object. **9-3**

Ground Vehicle Motion Primitives: Generate paths using Dubins, Reeds-Shepp, and straight-line connections **9-3**

R2018a

Manipulator Algorithm Blocks: Compute rigid body tree kinematics and dynamics in Simulink **10-2**

Lidar-Based SLAM: Localize robots and build map environments using lidar sensors **10-2**

Pose Graph Data Structure and Optimization: Represent and optimize 2-D and 3-D pose graphs **10-2**

3-D Occupancy Maps: Map 3-D environments using efficient octree data structure **10-2**

Enhanced Performance for rosbag Logfiles: Load rosbags faster and extract message data as structures **10-3**

R2017b

RigidBodyTree Visualization Improvements: Attach mesh files and inspect individual bodies in a MATLAB figure **11-2**

Coordinate Transformation Conversion Block: Convert between coordinate system representations in Simulink **11-2**

ROS Image and Point Cloud Blocks: Convert ROS messages to nonbus signals in Simulink **11-2**

Lidar Sensor Object: Store and use lidar scan data **11-2**

Scan Matching: New trust-region solver and code generation support **11-3**

R2017a

External Mode Support: Tune parameters and view signal values of deployed ROS nodes over TCP/IP (with Simulink Coder) **12-2**

Dynamics for Robot Manipulators: Solve inverse and forward dynamics for RigidBodyTree objects **12-2**

Generalized Inverse Kinematics: Solve multiconstrained inverse kinematics for robot manipulators **12-2**

URDF File Importer: Import URDF robot descriptions as a RigidBodyTree object **12-3**

Scan Matching: Calculate pose difference between laser scans **12-3**

Code Generation for RigidBodyTree objects: Generate code with robot manipulator data structures **12-3**

rosparam Simplified Commands: Modify ROS parameters using a simplified interface without creating a ParameterTree object **12-3**

R2016b

Robotic Manipulator Algorithms: Represent robot manipulators using a rigid body tree and calculate forward and inverse kinematics **13-2**

Automated Deployment of ROS Nodes: Automatically deploy ROS nodes to target hardware using Simulink Coder **13-2**

Occupancy Grid Class: Build a robot environment using a 2-D occupancy map with probabilistic values **13-2**

Mobile Robot Algorithm Blocks: Perform obstacle avoidance and path following in Simulink **13-2**

ROS Action Client: Send action goals via a ROS network and get feedback on their execution **13-2**

Buffered ROS tf2 Transformations: Access time-buffered transformations from the ROS transformation tree **13-2**

Odometry Motion Model Class: Predict poses for a differential drive robot **13-3**

ROS Time and Duration: Use mathematical operations on ROS time and duration objects **13-3**

Code Generation for Robotics Algorithms: Generate code for select algorithms **13-4**

R2016a

Monte Carlo Localization Algorithm: Estimate robot location in a known map **14-2**

Particle Filter Algorithm: Estimate state for nonlinear systems **14-2**

Fixed-Rate Execution: Run MATLAB code at a constant rate **14-2**

Robotics System Toolbox Support Package for TurtleBot based Robots: Connect to TurtleBot hardware **14-2**

String support for ROS parameters in Simulink **14-2**

String array support for ROS messages in Simulink **14-2**

Code generation from Simulink using Simulink Coder **14-3**

roboticsSupportPackages function replaced with roboticsAddons **14-3**

R2015aSP1

Bug Fixes

R2015b

Vector Field Histogram Plus (VFH+) obstacle avoidance algorithm **16-2**

Access to ROS parameters from Simulink **16-2**

Code generation for coordinate transforms and select robotics algorithms **16-2**

R2015a

Path planning, path following, and map representation algorithms . . . **17-2**

Functions for converting between different rotation and translation representations **17-2**

Bidirectional communication with live ROS-enabled robots **17-2**

Interface to Gazebo and other ROS-enabled simulators **17-2**

Data import from rosbag log files **17-2**

ROS node generation from Simulink models (with Embedded Coder) **17-2**

R2022b

Version: 4.1

New Features

Bug Fixes

Compatibility Considerations

Spatial Math: Use SE(2), SE(3), SO(2), and SO(3) transformations

Use the `se3` and `so3` objects to represent 3-D transformation matrices and 3-D rotation matrices, respectively. These objects make it easier to perform common operations, such as transforming 3-D points, normalizing the rotation, or interpolating between positions and orientations. To represent multiple transformations and rotations, you can arrange these objects into arbitrarily shaped arrays.

The `se2` and `so2` objects represent 2-D transformation matrices and 2-D rotation matrices, respectively.

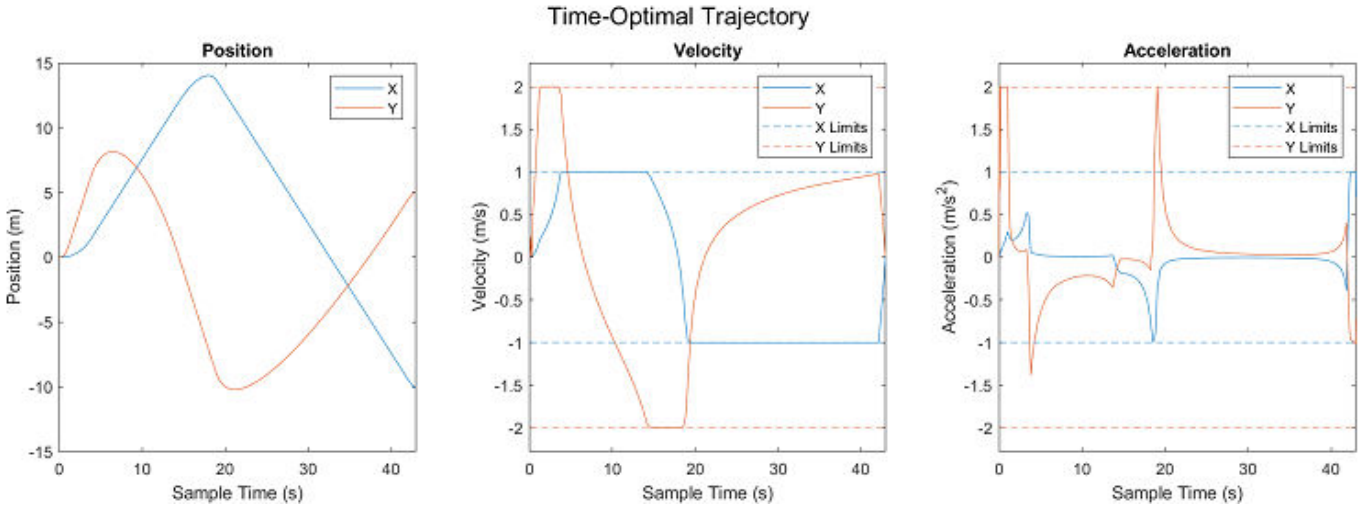
Use `plotTransforms` to visualize the transformations and rotations in a figure. This can be helpful to see the underlying positions and orientations and to see interpolation results.

Custom Sensor Support in Robot Scenarios: Add custom sensor models to robots in simulated scenarios

Specify custom sensor models and define their behavior in simulation using the `robotics.SensorAdaptor` class. To generate a template for implementing the class, use the `createCustomRobotSensorTemplate` function.

Time Optimal Trajectory: Create time optimal trajectory subject to kinematic constraints

Use the `contopptraj` function to generate a time optimal trajectory subject to velocity and acceleration limits. This is also useful for retiming existing trajectories such that they are also within acceleration and velocity limits. For more information, see “Generate Time-Optimal Trajectories with Constraints Using TOPP-RA Solver”.

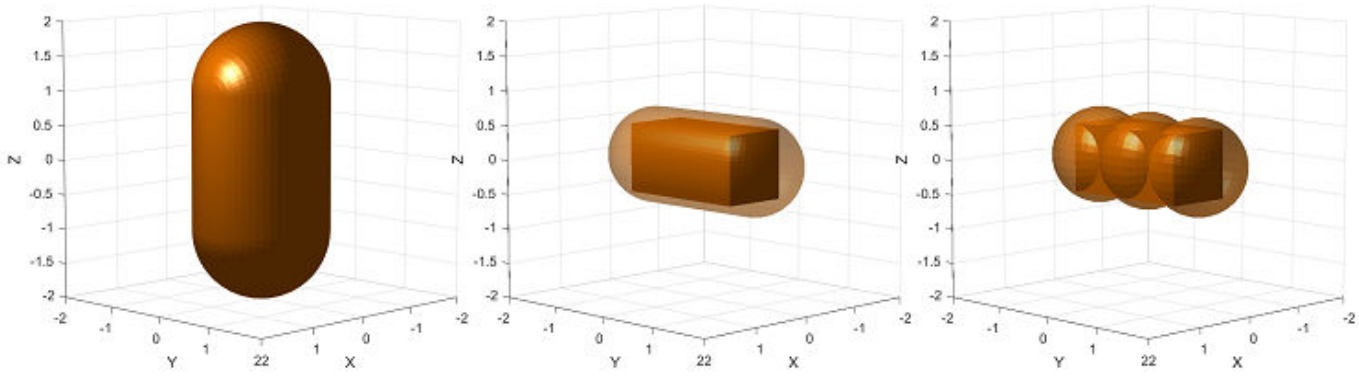


Capsule Collision Geometry: Use additional collision capsule geometry

Create collision capsules as `collisionCapsule` objects and use the `fitCollisionCapsule` function to fit a collision capsule over a `collisionBox`, `collisionCylinder`, `collisionSphere`, or `collisionMesh` object. This is useful for reducing motion planning times by approximating

collision meshes with collision capsules. See “Reduce Motion Planning Times Using Capsule Approximation” for more information.

Use the `genspheres` object function to decompose a collision capsule into spheres for faster collision-checking performance.

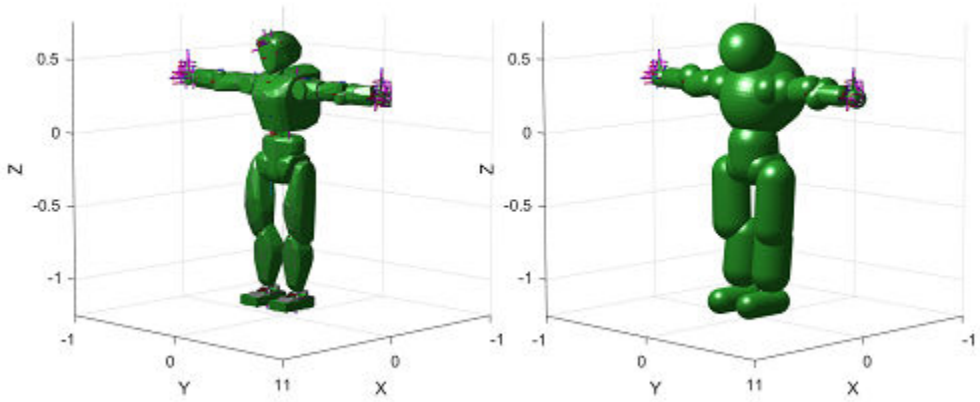


Collision Checking: Check collisions using collision capsules

The `checkCollision` function now supports collision checking between `collisionCapsule` objects.

Capsule Approximation: Approximate rigid body trees using capsules

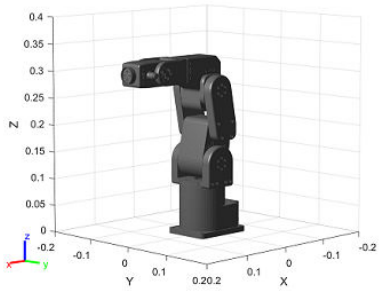
Use the `capsuleApproximation` object to create an approximation of a rigid body tree robot by using `collisionCapsule` objects, for more efficient collision checking. By creating approximations of collision meshes of the rigid body tree, you can reduce motion planning times. See “Reduce Motion Planning Times Using Capsule Approximation” for more information.



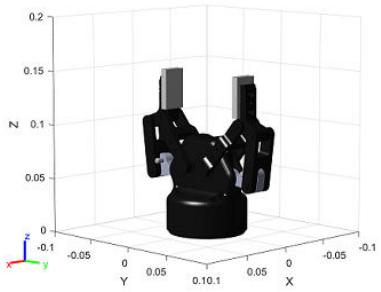
Robot Models: Use additional manipulator and gripper rigid body tree robot models introduced to robot model library

You can now retrieve these additional robots from the robot model library using the `loadrobot` function:

- `"meca500r3"` — Mecademic Meca500 R3 six-axis robot arm



- `"robotiq2F85"` — Robotiq 2F-85 two-finger gripper



Self-Collision Checking: Alter rigid body tree self-collision checking behavior change, new default self-collision checking behavior

You can now specify self-collision checking behavior for a rigid body tree robot model by using the `SkippedSelfCollisions` name-value argument of the `checkCollision` and by setting the `SkippedSelfCollisions` property of the `manipulatorCollisionBodyValidator` and `manipulatorRRT` objects. Specify the `SkippedSelfCollisions` argument as `"parent"` or `"adjacent"`.

- `"parent"` — Collision checking ignores self-collisions between parent and child rigid bodies.
- `"adjacent"` — Collision checking ignores self -collisions between rigid bodies of adjacent indices.

As of R2022b, the default behavior of collision checking is to ignore self-collisions between parent and child rigid bodies. In previous releases, the default behavior of self-collision checking was to ignore self-collisions between rigid bodies at adjacent indices. To instead ignore self-collisions between rigid bodies of adjacent indices, specify the `SkippedSelfCollisions` argument as `"adjacent"`.

Gazebo Co-Simulation: Gazebo co-simulation enhancements

The Gazebo co-simulation features have been updated with these enhancements:

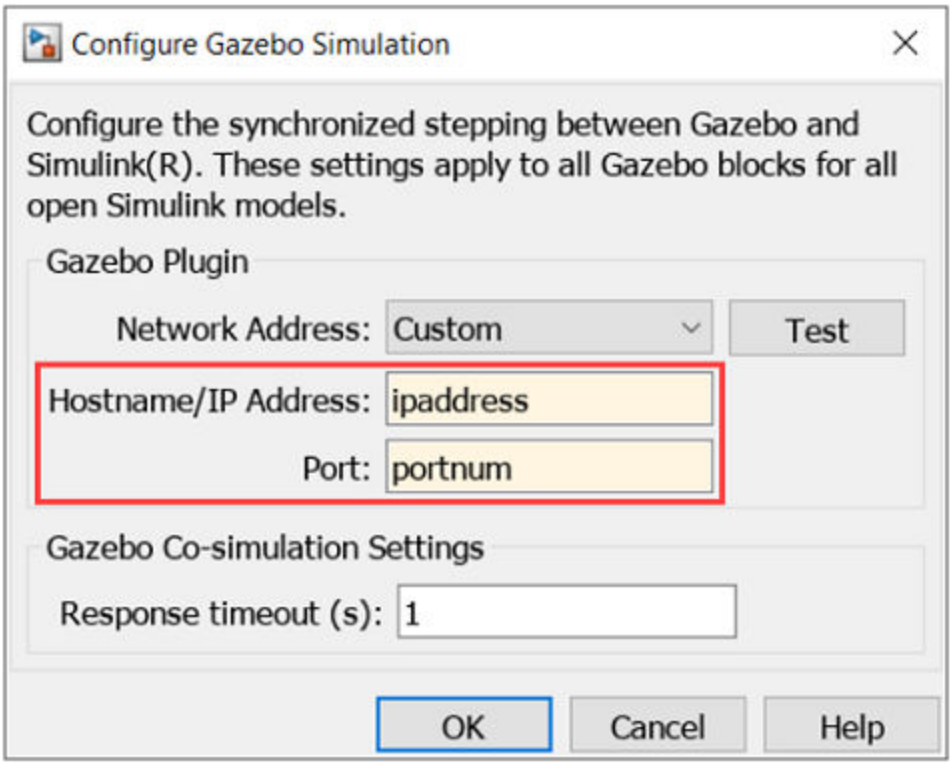
- You can now connect to multiple Gazebo simulations from one or more machines. You can now specify a cell array of IP addresses and a cell array of port numbers in the MATLAB®, workspace and then specify their variable names to the **Hostname/IP Address** and **Port** boxes, respectively, of the Configure Gazebo Simulation dialog box.

  To connect to a single Gazebo session from MATLAB, specify the port number and IP address of the computer running the Gazebo simulator.

  ```
  portnum = 14580;
  ipaddress = '172.18.250.125';
  ```

  To connect to multiple Gazebo sessions from MATLAB, specify the port numbers and IP addresses of the computers running the Gazebo simulator.

  ```
  portnum = {14580,14581};
  ipaddress = {'172.18.250.125','172.18.250.125'};
  ```



- The Gazebo Read block now automatically suggests to change the dimension of the bus to match the signals from Gazebo. You can now toggle the joint-axis-related bus signal to variable or fixed dimension.

Gazebo Co-Simulation: Gazebo Read block sensor read performance improvements

The Gazebo Read block now has improved sensor read performance. The performance is measured on two different machine set ups while keeping the **Real Time Factor** in the Gazebo simulator at 1 for all the conditions. All the sensors are running at 30Hz.

The tables show the time taken to complete a 10 second simulation to read messages from each of the sensors. The sensors with read time that matches the simulation time of 10 seconds are considered to be at real time.

Gazebo and MATLAB running on the same machine

Release Version	IMU	Lidar	Camera
R2022a	769 sec	12.82 sec	66.66 sec
R2022b	10 sec	10 sec	30.3 sec

Gazebo and MATLAB running on different machines

Release Version	IMU	Lidar	Camera
R2022a	909 sec	33.33 sec	100 sec
R2022b	25 sec	25 sec	50 sec

These simulations were timed in MATLAB running on a Ubuntu® Bionic 20.04, Intel® Xeon® W-2133 CPU @ 3.60 GHz, with 12 threads and 64 GB of RAM.

The Gazebo simulator is running on a Ubuntu Bionic 20.04 virtual machine, Intel Xeon W-2133 CPU @ 3.60 GHz, with 8 threads and 8 GB of RAM.

waypointTrajectory Reverse Motion: Specify wait and reverse motion for waypoint trajectory

You can now specify wait and reverse motion using the `waypointTrajectory` System object™.

- To let the trajectory wait at a specific waypoint, simply repeat the waypoint coordinate in two consecutive rows when specifying the Waypoints property.
- To render reverse motion, separate positive (forward) and negative (backward) groundspeed values by a zero value in the GroundSpeed property.

Support Package Updates

The Robotics System Toolbox Support Package for Manipulators has been split into these support packages:

- Robotics System Toolbox Support Package for KINOVA® Gen3 Manipulators
- Robotics System Toolbox Support Package for Universal Robots UR Series Manipulators

For more information, see *Release Notes for Robotics System Toolbox Support Package for KINOVA Gen3 Manipulators*.

New Examples

This release contains several new examples:

- “Reduce Motion Planning Times Using Capsule Approximation”
- “Motion Planning for Backhoe Using RRT”
- “Multi-Robot Control with Resource Allocation and Conflict Management”
- “Generate Time-Optimal Trajectories with Constraints Using TOPP-RA Solver”
- “Pick-and-Place Workflow in Unity 3D Using ROS”
- “Sign Following Robot with Time Synchronization Using ROS and Gazebo Co-Simulation”
- “Simulate Ultrasonic Sensors Mounted on Mobile Robots”

R2022a

Version: 4.0

New Features

Bug Fixes

Compatibility Considerations

Robot Scenarios and Sensor Models: Author robot scenarios and simulate sensor readings for robotics applications

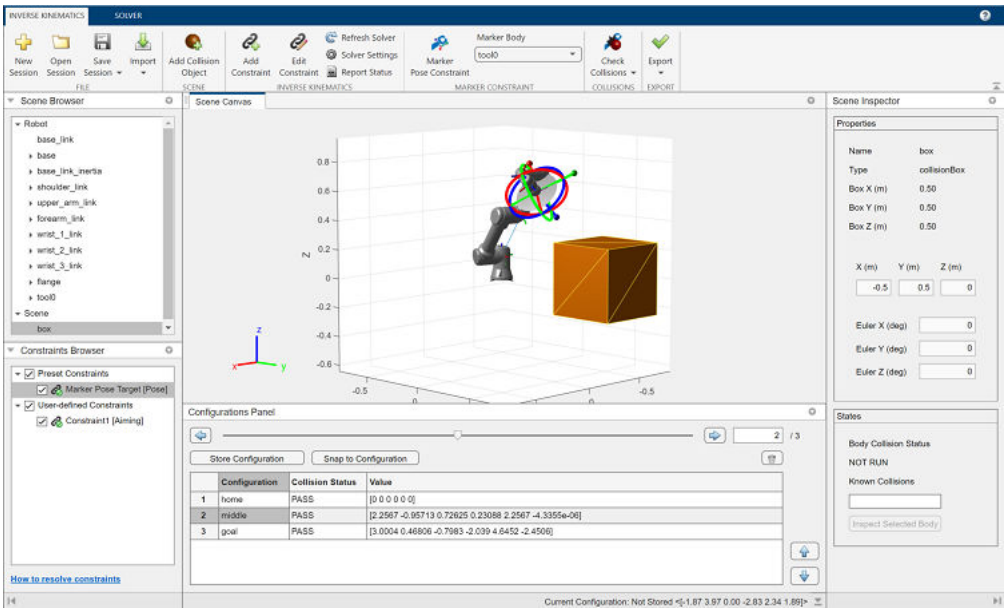
The `robotScenario` object generates a simulation scenario consisting of static meshes, robot platforms, and sensors in a 3-D environment. Add `robotPlatform` objects to the scenario and attach different sensor models with the `robotSensor` object. To specify complex mesh objects as triangulated vertices and faces, use the `extendedObjectMesh` object. Add the mesh to robotic scenarios using the `addMesh` function.

The `robotPlatform` object represents a robot platform in a given robot scenario. Use the platform to define and track the trajectory of the base motion of a robot platform in the scenario. To simulate sensor readings for the platform, mount sensors like the `gpsSensor`, `insSensor`, or `robotLidarPointCloudGenerator` objects using the `robotSensor` object. Add a body mesh for visualization using the `updateMesh` function.

The `robotSensor` object creates a sensor in a robot scenario. The sensor is rigidly attached to a robot platform, specified as a `robotPlatform` object. You can specify different mounting positions and orientations. Configure this object to automatically generate readings from a sensor specified as a `gpsSensor`, `insSensor`, or `robotLidarPointCloudGenerator` object.

Inverse Kinematics Designer: Design inverse kinematics solvers, configurations, and waypoints

Load `rigidBodyTree` objects into the **Inverse Kinematics Designer** app to visualize and tune an inverse kinematics solver with or without kinematic constraints. Create, modify, and export joint configurations as well as the solver by exporting to the MATLAB workspace.



Convert Collision Mesh: Convert primitive collision objects to collision mesh

Use the `convertToCollisionMesh` function to convert `collisionBox`, `collisionSphere`, or `collisionCylinder` objects into a `collisionMesh` object.

Kinematic Constraints: Additional Generalized Inverse Kinematics Constraints

Use the joint constraint objects to constrain one rigid body using another rigid body on the same `rigidBodyTree` object:

- `constraintFixedJoint` — Create a fixed joint constraint to fix the position and orientation between two bodies.
- `constraintPrismaticJoint` — Create a prismatic joint constraint between two bodies to simulate prismatic motion.
- `constraintRevoluteJoint` — Create a revolute joint constraint between two bodies to simulate revolute motion.

Use the `constraintDistanceBounds` constraint object to constrain an end effector within a specified minimum and maximum distance of another body on the same `rigidBodyTree`.

INS Sensor Model: Simulate inertial navigation and GPS readings

Use the `insSensor` object to simulate inertial navigation and GPS readings.

GPS Sensor Model: Simulate GPS receiver readings

Use the `gpsSensor` object to simulate GPS receiver readings based on position and velocity inputs.

Trajectory and Waypoint Following Algorithm: Use built-in algorithm to generate trajectories and control commands for robots

Use the `waypointTrajectory` object to generate trajectories for sensors or robots and control commands to send to your robots.

Transform Tree Object: Define coordinate frames and relative transformations

The `transformTree` object contains an organized tree structure for defining coordinate frames and their relative transformations over time.

Point Cloud Object: Store 3-D point clouds

The `pointCloud` object creates point cloud data from a set of points in a 3-D coordinate system. You can retrieve, select, and remove desired points from the point cloud data.

Trajectory Generation Blocks: Create minimum jerk and minimum snap polynomial trajectories with Simulink

Use the Minimum Jerk Polynomial Trajectory block to generate multi-axis, minimum jerk, polynomial trajectories.

Use the Minimum Snap Polynomial Trajectory block to generate multi-axis, minimum snap, polynomial trajectories.

Commercial Robot Models: Universal Robots E-Series models introduced to the library of robot models

You can now retrieve these additional commercially available robots from the robot model library using the `loadrobot` function:

- `"universalUR3e"`
- `"universalUR5e"`
- `"universalUR10e"`
- `"universalUR16e"`

The meshes for these robots are also available in the Robotics System Toolbox Robot Library Data support package. See Get and Manage Add-Ons or Robotics System Toolbox Robot Library Data in File Exchange.

Simulation 3-D Environment Upgrade: Gazebo 11 support

Gazebo co-simulation framework now supports Gazebo 11. In prior releases, the framework supported only Gazebo 9 and Gazebo 10.

New Examples

This release contains several new examples:

- Plan Manipulator Path for Dispensing Task Using Inverse Kinematics Designer
- Create Constrained Inverse Kinematics Solver Using Inverse Kinematics Designer
- Perform Path Planning Simulation with Mobile Robot
- Perform Obstacle Avoidance in Warehouse Scenario with Mobile Robots
- Solve Inverse Kinematics for Closed Loop Linkages
- Generate Code for Manipulator Motion Planning in Perceived Environment
- Design Position Controlled Manipulator Using Simscape
- Perform Trajectory Tracking and Compute Joint Torque for Manipulator Using Simscape

R2021b

Version: 3.4

New Features

Bug Fixes

State Space and State Validation for Robot Manipulator Models

The `manipulatorStateSpace` object represents the joint configuration state space of a rigid body tree robot model. For a given `rigidBodyTree` object, the nonfixed joints in the rigid body tree model form the state space. The most common use of the manipulator state space is with sampling-based path planners like the `plannerRRT` (Navigation Toolbox) and `plannerBiRRT` (Navigation Toolbox) objects.

Use these features for customization of planning beyond what the `manipulatorRRT` object allows.

To sample and validate paths for manipulators, combine the state space with a state validator by using the `manipulatorCollisionBodyValidator` object.

Ignore Self Collisions for Manipulator RRT Path Planning

The `IgnoreSelfCollision` property of the `manipulatorRRT` object determines whether or not to check for self collisions when planning paths. If this property is set to `true`, the `plan` function skips checking for collisions between bodies and only compares the bodies to the environment. Skipping checking for self collisions can improve the speed of the planning phase.

Minimum Jerk and Snap Polynomial Trajectories

Use the `minjerkpolytraj` function to generate a minimum jerk polynomial trajectory. Specify waypoints, time points, and number of samples to get a series of positions, velocities, accelerations and jerks.

Use the `minsnappolytraj` function to generate a minimum snap polynomial trajectory. Specify waypoints, time points, and number of samples to get a series of positions, velocities, accelerations, jerks, and snaps.

Simulation Description Format (SDF) Support

The `gzmodel` function now returns the SDF of the specified Gazebo model as a string.

The `importrobot` function now imports an SDF model from an SDF file or an SDF text to MATLAB as a `rigidBodyTree` object.

COLLADA Mesh Support

The `rigidBodyTree` object now supports COLLADA™ (`*.dae`) mesh files for collision checking and visualizing a robot. In addition to existing STL 3-D mesh format, `importrobot` with URDF and SDF models now supports the COLLADA DAE 3-D mesh format for mesh import.

Load Robot Function Update

The `loadrobot` function now supports multiple URDF versions for a robot model by specifying the `Version` property. `loadrobot("kinovaGen3","Version",2)` will load the second URDF version of the Kinova Gen 3.

R2021a

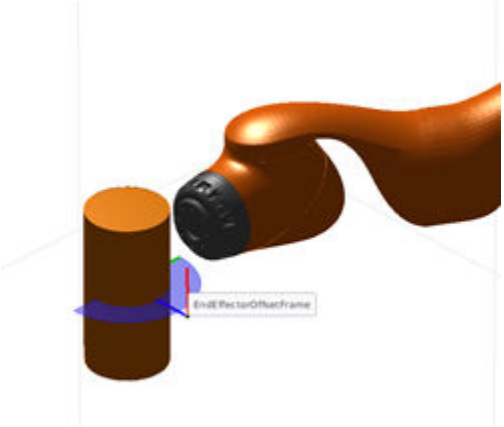
Version: 3.3

New Features

Bug Fixes

Workspace Regions For Motion Planning: Specify a workspace region to sample end-effector poses for path planning

The `workspaceGoalRegion` object defines a region for valid end-effector goal poses. To sample poses within the bounds of the goal region, use the `sample` function. You can also visualize the bounds you define using the `show` function.



The object is typically used with rapidly exploring random tree (RRT) planners like the `manipulatorRRT` object. Specify your workspace goal region as the third input to the `plan` function. Using the `workspaceGoalRegion` object, the planner can plan to multiple reachable goal end-effector poses during planning and give collision-free paths.

Gazebo Co-Simulation in MATLAB: Access and modify Gazebo model parameters

Use the following Gazebo co-simulation MATLAB functions to access and modify Gazebo model parameters:

- `gzinit` — Initialize connection settings for Gazebo co-simulation in MATLAB
- `gzjoint` — Assign and retrieve Gazebo model joint information
- `gzlink` — Assign and retrieve Gazebo model link information
- `gzmodel` — Assign and retrieve Gazebo model information
- `gzworld` — Interact with Gazebo world

Educational Robot Model: Additional rigid body tree robot model for manipulator introduced to the library of robot models

Load the `"quanserQArm"` robot model using the `loadrobot` function, which returns a `rigidBodyTree` object of the manipulator.

Rigid Body Tree Visualization Improvements

The `show` function of the `rigidBodyTree` object supports faster update of the rigid body tree robot model in the figure in a loop for fast animations. Enabling the '`FastUpdate`' name-value argument

increases the rate at which the robot model updates in the figure. You can loop the updates for a sped up animation.

Rigid Body Tree Function Generation

Use the `writeAsFunction` function to generate a function file that creates the `rigidBodyTree` object of the specified robot model. The generated function supports code generation.

R2020b

Version: 3.2

New Features

Bug Fixes

RRT Planner for Manipulators: Plan collision-free motion for rigid body tree robot models

The `manipulatorRRT` object enables you to plan paths for a `rigidBodyTree` robot model using the bidirectional rapidly-exploring random tree (RRT) algorithm. Specify the collision geometries for the `rigidBodyTree` object, a start and goal configuration, and any obstacles in the environment using collision meshes. Use the `plan` function to generate a collision-free path from start to goal.

To find direct connections between the bidirectional trees, enable the `ConnectHeuristic` property of the `manipulatorRRT` object, which ignores the maximum connection distance for extending the tree.

To shorten the path length, use the `shortenPath` function, which finds random edges to trim from the path.

Collision Checking for Robot Meshes: Add collision meshes to rigid body tree models and check collisions for robot configurations

The `rigidBodyTree` object now supports adding collision meshes along with visual meshes by using the `addCollision` object function. The `checkCollision` object function checks for self collisions and collisions with the environment based on robot configurations and obstacle positions.

Custom Messages with Gazebo: Publish and subscribe to custom message types in a Gazebo simulation

Use the Gazebo Publish and Gazebo Subscribe blocks to send and receive custom messages on specific topics in a Gazebo simulation. Specify messages as a bus with the appropriate elements based on your message type.

Joint and Link States in Gazebo: Send and receive messages for robot joint and link states in a Gazebo Simulation

The Gazebo Blank Message block now generates commands for setting the position and velocity of different joint and link elements of a Gazebo simulation as a bus. Select the message type in the block mask, and assign the appropriate elements of the bus for your command. To send the commands, use the Gazebo Apply Command block. To read states, use a Gazebo Read block configured for the topic of the desired element.

Analytical Inverse Kinematics: Generate functions for inverse kinematics solutions using closed-form solutions

The `analyticalInverseKinematics` object is an analytical inverse kinematics solver for a specific `rigidBodyTree` robot model. The object supports kinematic group types for six or seven degree-of-freedom (DoF) robots. Using the object, generate a function for your robot model and query solutions based on a desired end-effector pose.

Educational and Commercial Robot Models: Additional rigid body tree robot models for manipulators and mobile robots introduced to the library of robot models

Load additional robot models using the `loadrobot` function, which returns a `rigidBodyTree` object. To access these robot models, specify the associated string name for the `robotname` argument of the `loadrobot` function

- `"kukaIiwa14"`
- `"quanserQBot2e"`
- `"quanserQCar"`
- `"rethinkSawyer"`

Robotics System Toolbox UAV Library add-on is now the UAV Toolbox

The Robotics System Toolbox UAV Library add-on is now available as the UAV Toolbox.

Robotics System Toolbox Support Package for Manipulators Add-on

The Robotics System Toolbox Support Package for Manipulators add-on is new in this release for connecting to KINOVA manipulator hardware.

To see available add-ons for Robotics System Toolbox, use the `roboticsAddons` function.

R2020a

Version: 3.1

New Features

Bug Fixes

Interactive Robot Visualization: Manipulate rigid body tree models with visual meshes and perform inverse kinematics for target bodies

Create an `interactiveRigidBodyTree` object to launch a window displaying a `rigidBodyTree` object and the attached visual meshes. You can directly modify the robot position and configuration using interactive markers and see the results in the figure. Also, you can specify additional constraints and inverse kinematics parameters in the object to further restrict the robot motion when solving for configurations.

Commercial Robot Models: Load additional rigidBodyTree robot models for manipulators and mobile robots added to our library of existing models

Use commercially available robot models using the `loadrobot` function. Robot models are returned as a `rigidBodyTree` object. Starting this release, new robot model options available include:

- `"abbIrb1600"`
- `"abbYuMi"`
- `"atlas"`
- `"fanucLRMate200ib"`
- `"fanucM16ib"`
- `"frankaEmikaPanda"`
- `"robotisOP2"`
- `"robotisOpenManipulator"`
- `"universalUR10"`
- `"universalUR3"`
- `"universalUR5"`
- `"valkyrie"`
- `"yaskawaMotomanMH5"`

Code Generation for Collision Checking: Generate C/C++ code using the checkCollision function and collision geometries

You can now generate code when using `checkCollision` and the collision geometry objects `collisionBox`, `collisionCylinder`, `collisionMesh`, and `collisionSphere`.

Simscape Multibody Model Parameters: Import models with initial position and joint limits

When you import Simscape™ Multibody™ models using `importrobot`, the assembly initial position is set to the home configuration of your `rigidBodyTree` model. Also, the joint limits are set in the `PositionLimits` property of each `rigidBodyJoint` object.

R2019b

Version: 3.0

New Features

Bug Fixes

Compatibility Considerations

Gazebo Co-simulation: Perform time-synchronized simulation of Gazebo with Simulink

Gazebo is a physics-based simulator for testing and simulating robotics applications. Co-simulation means synchronized time-stepping of your Simulink® model with the Gazebo simulation. Use the Gazebo Pacer to control the pace of your model. To select entities, receive and send messages, or apply commands, use the other provided Gazebo blocks:

- Gazebo Apply Command
- Gazebo Blank Message
- Gazebo Read
- Gazebo Select Entity

For examples using Gazebo co-simulation, see

- Perform Co-Simulation between Simulink and Gazebo
- Control A Differential-Drive Robot in Gazebo With Simulink

Robot Motion Modeling and Simulation: Simulate mobile robot kinematics and closed-loop manipulator dynamics

Use provided motion models to simulate mobile robots and manipulator robot motion. Mobile robot models include:

- `ackermannKinematics` object or Ackermann Kinematic Model block
- `bicycleKinematics` object or Bicycle Kinematic Model block
- `differentialDriveKinematics` object or Differential Drive Kinematic Model block
- `unicycleKinematics` object or Unicycle Kinematic Model block

Manipulator models support any rigidBodyTree model for both joint- and task-space kinematics:

- `taskSpaceMotionModel` object or Task Space Motion Model block
- `jointSpaceMotionModel` object or Joint Space Motion Model block

Collision Checking: Define collision shapes and detect collisions between mesh geometries

Create collision meshes using either primitive shapes or custom mesh definitions with:

- `collisionBox`
- `collisionCylinder`
- `collisionSphere`
- `collisionMesh`

Check for collisions and calculate distances between meshes using the `checkCollision` function. For examples using collision detection, see:

- Check for Manipulator Self Collisions using Collision Meshes

- Check for Environmental Collisions with Manipulators

Commercial Robot Models: Use a provided library of rigid body robot models to quickly model your robot applications

Use commercially available robot models using the `loadrobot` function. Robot models are returned as a `rigidBodyTree` object. Robot models available include:

- KINOVA® Gen3
- KINOVA JACO®
- KINOVA MICO®
- ABB IRB 120
- Rethink Robotics Baxter
- Willow Garage PR2

For an example using the Gen3 robot, see Plan and Execute Collision-Free Trajectories using KINOVA Gen3 Manipulator.

Robot Application Examples: Get started with reference examples for pick-and-place robots and warehouse mobile robots

Reference examples for complete mobile robot and manipulator workflows are provided:

Pick-and-Place Manipulator Robots

- Plan and Execute Task- and Joint-space Trajectories using KINOVA Gen3 Manipulator
- Plan and Execute Collision-Free Trajectories using KINOVA Gen3 Manipulator
- Pick-and-Place Workflow using Stateflow for MATLAB

Warehouse Mobile Robots

- Plan Path for a Differential Drive Robot in Simulink
- Execute Tasks for a Warehouse Robot
- Control A Differential-Drive Robot in Gazebo With Simulink
- Simulate Different Kinematic Models for Mobile Robots

Robotics System Toolbox Support Package for Turtlebot-Based Robots functionality has moved

Starting in R2019b, the Robotics System Toolbox Support Package for Turtlebot-Based Robots will no longer be available for download. This functionality has been moved to ROS Toolbox Support Package for TurtleBot -Based Robots.

Robotics System Toolbox has transitioned into Robotics System Toolbox, Navigation Toolbox, and ROS Toolbox

Starting in R2019b, the Robotics System Toolbox has been transitioned to 3 products:

- Robotics System Toolbox
- Navigation Toolbox
- ROS Toolbox

Explore product capabilities for designing robotics applications, planning and navigation algorithms, and ROS-based applications.

R2019a

Version: 2.2

New Features

Bug Fixes

SLAM Map Builder Sessions: Save and load app sessions

The **SLAM Map Builder** app now allows you to save and load app sessions. While building a map and tuning SLAM parameters, your progress can be saved to a file. Load the session to return to the same point in your map-building process.

MAVLink Protocol Support: Communicate with UAVs using MAVLink messages and load log files

Use the MAVLink communication protocols in MATLAB and load specific MAVLink dialects. To load a dialect, use `mavlinkdialect`. Connect to MAVLink clients and send and receive messages using `mavlinkio`. You can also load telemetry logs (`.tlogs`) and use the data in MATLAB.

Trajectory Generation: Create piecewise polynomials, trapezoidal velocity profiles, B-splines, and task-space interpolation

You can now generate multi-axis trajectories using new functions or blocks. Specify waypoints and time-scaling vectors to get a series of accelerations, velocities, and positions using third-order (cubic) polynomials, fifth-order (quintic) polynomials, B-splines, or trapezoidal velocity profiles. You can also interpolate between rotations or transformations with a specified time-scaling vector.

Model Reference Support for ROS: Use model reference in models with ROS Blocks

All ROS (Robot Operating System) blocks can now be used with model reference in Simulink. Simulink models are still limited to one ROS node, so referenced models all contribute to a single node at the top-level model. Execution modes that require host code generation (Accelerator and Rapid Accelerator modes) do not support model reference.

Orbit Follower for UAVs: Follow a circular path around a point of interest

The Orbit Follower block and `uavOrbitFollower` object allow you to follow a circular path around a central point. Specify the center location and radius of the circular path to follow. The block uses the current pose and look ahead distance and generates the desired heading ad yaw to achieve the circular path.

Code Generation for SLAM: Generate code using LidarSLAM, PoseGraph, and PoseGraph3D objects

You can now generate code when using `robotics.LidarSLAM`, `robotics.PoseGraph`, and `robotics.PoseGraph3D` objects. The `optimizePoseGraph` function also supports code generation.

R2018b

Version: 2.1

New Features

Bug Fixes

SLAM Map Builder App: Build and tune a 2-D grid map with lidar-based SLAM

Use the **SLAM Map Builder** app to import lidar scan and odometry data from a rosbag logfile or the MATLAB workspace to build an occupancy grid map. The app utilizes the lidar-based SLAM (simultaneous localization and mapping) algorithm to map the environment and localize the robot. To improve map quality, the app also provides an easy user interface to tune the SLAM algorithm parameters and manually modify individual incremental scans and loop-closure matches.

UAV Algorithms: Create UAV guidance models and 3-D path following for fixed-wing and multirotor UAVs

Use a `fixedwing` or `multirotor` object to generate a reduced-order guidance model for fixed-wing and multirotor UAVs (unmanned aerial vehicles). Use these functions with the guidance model:

- `control` — Control commands for UAVs
- `derivative` — Time derivative of UAV states
- `environment` — Environment inputs for UAVs
- `state` — UAV states for position, velocity, attitude, and thrust

The UAV Guidance Model block also contains these same guidance models for both UAV types. Specify the UAV initial state, control commands, and the environmental conditions to get the simulated UAV states from the block.

Use the Waypoint Follower block or the `uavWaypointFollower` System object to navigate a set of waypoints using a lookahead point. You can specify your UAV type, a fixed or variable transition radius, and control yaw at the waypoints.

Read Data Block: Play back data from a rosbag logfile in Simulink

The Read Data block loads rosbag logfile data into Simulink for playback in simulation. Messages are played back at the simulation time corresponding to the recorded rosbag time. Specify the logfile, time settings, and the desired topic to output.

Inverse Kinematics Block: Calculate joint configurations for a desired end-effector pose in Simulink

Use the Inverse Kinematics block to calculate the joint configuration for a desired end-effector pose in Simulink. Specify a `robotics.RigidBodyTree` object for your manipulator robot in MATLAB. The block uses this model to generate valid joint positions and angles to achieve the desired end-effector pose.

ROS Service and Current Time Blocks: Call ROS services and get the current ROS time in Simulink

Use the Call Service block to call a ROS service in Simulink. Specify a service request message and get a response from the service server. A valid ROS service server must be running on your ROS network. You can also now generate blank service request and response messages using the Blank Message block.

Use the Current Time block to get the current ROS or system time based on your ROS network connection.

You can now run a deployed model based on the ROS time. Select the **Enable ROS time model stepping** in the **Model Configuration Parameters**. Under **Hardware Implementation**, set **Hardware board** to `Robot Operating System (ROS)`, and select **Enable ROS time model stepping** under **Target Hardware resources** > **ROS time**.

Simscape Multibody Data Exchange: Use importrobot to import Simscape Multibody models to a RigidBodyTree object.

Use `importrobot` to import a Simscape Multibody model to MATLAB as a `robotics.RigidBodyTree` object. The `RigidBodyTree` object supports rigid bodies with revolute, prismatic, and fixed joints. Supported blocks are mapped to these components.

Ground Vehicle Motion Primitives: Generate paths using Dubins, Reeds-Shepp, and straight-line connections

Use motion models for Dubins, Reeds-Shepp, and straight-line connections to generate path segments. Specify the properties for the different motion models in the connection objects:

- `robotics.DubinsConnection`
- `robotics.ReedsSheppConnection`

Use the `connectconnect` function to create valid path segments between poses:

- `robotics.DubinsPathSegment`
- `robotics.ReedsSheppPathSegment`

You can then `interpolate` poses along the path segment or visualize it using `showshow`.

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

Version: 1.3

New Features

Bug Fixes

Compatibility Considerations

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

ROS 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.