

Navigation Toolbox™ Release Notes



MATLAB® & SIMULINK®



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

Navigation Toolbox™ Release Notes

© COPYRIGHT 2019–2020 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.

New Time Scope object: Visualize signals in the time domain	1-2
Scope Tab	1-2
Measurements Tab	1-2
Scale Axes	1-3
Scan Matching Using Line Features: Estimate pose and covariance based on line features in lidar scans	1-3
Trajectory Optimization Improvements: Specify longitudinal segments, deviation offsets, and additional waypoint parameters	1-3
Path Metrics Improvements: Specify validatorVehicleCostmap as a state validator	1-4
Ray Intersections for 3-D Maps: Calculate ray intersections, import, and export with a 3-D occupancy map	1-4
Code Generation for Monte Carlo Localization: Generate C/C++ code using the monteCarloLocalization object	1-4
Code Generation for Sampling-Based Planners: Generate C/C++ code using the plannerRRT, plannerRRTStar, and plannerHybridAStar objects	1-4
Code Generation for Trajectory Optimization: Generate C/C++ code using the trajectoryOptimalFrenet object	1-4
Access residuals and residual covariance of insfilters and ahrs10filter	1-4
Model inertial measurement unit using IMU Simulink block	1-4
Estimate device orientation using AHRS Simulink block	1-4
Calculate angular velocity from quaternions	1-5
Transform position and velocity between two frames to motion quantities in a third frame	1-5

Simultaneous Localization and Mapping (SLAM): Create 2-D and 3-D occupancy maps using SLAM algorithm and lidar scan data	2-2
SLAM Map Builder App: Interactively modify loop closures and adjust overall map using SLAM algorithm	2-2
Pose Estimation: Accurately estimate vehicle poses using IMU and GPS sensors and Monte Carlo Localization	2-2
Customizable Sampling-Based Path Planners: Plan a path from start to goal locations using RRT and RRT* algorithms	2-2
Path-Planning Metrics: Use metrics to check and compare the output of path planners	2-3
Sensor Models: Use simulated models for IMU, GPS, and range sensors	2-3
Trajectory and Waypoint Following Algorithms: Use built-in algorithms to generate trajectories and control commands for robots	2-3

R2020a

Version: 1.1

New Features

Bug Fixes

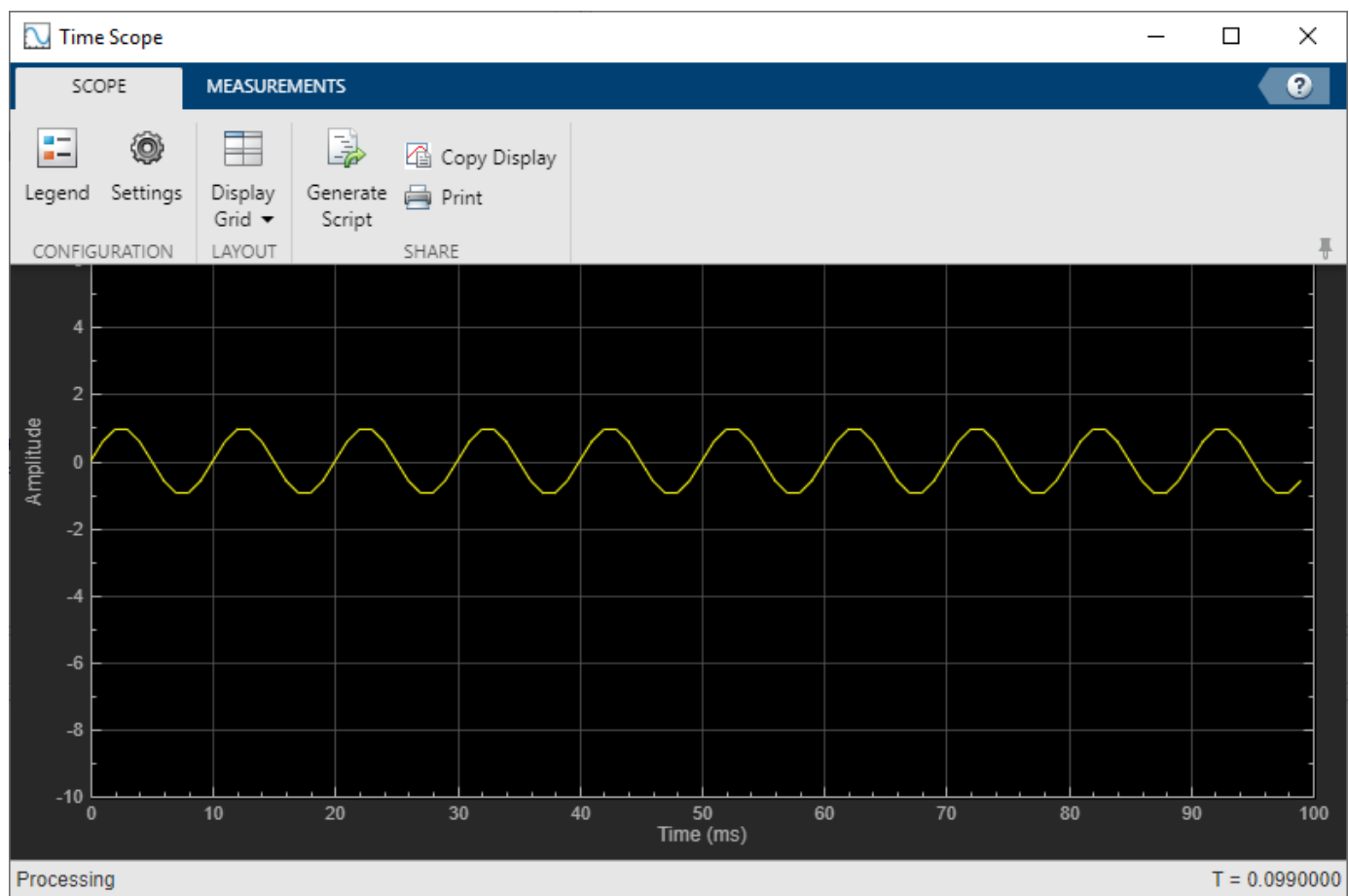
New Time Scope object: Visualize signals in the time domain

Use the `timescope` object to visualize real- and complex-valued floating-point and fixed-point signals in the time domain.

The Time Scope window has two toolstrip tabs:

Scope Tab

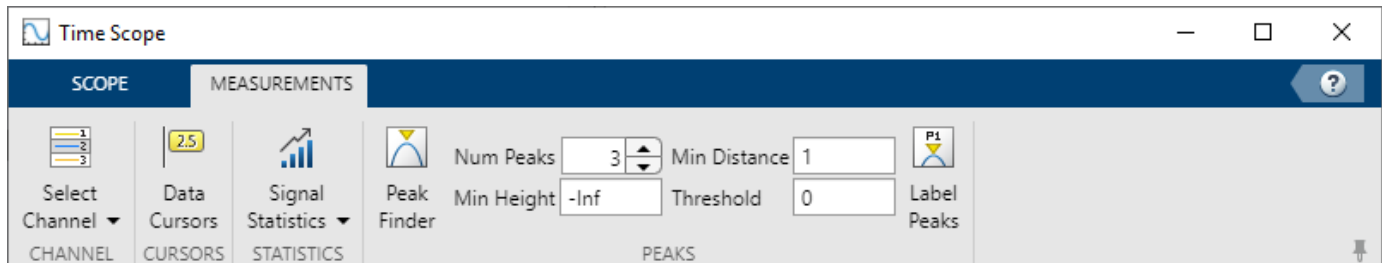
In the **Scope** tab, you can control the layout and configuration settings, and set the display settings of the Time Scope. You can also generate script to recreate your Time Scope with the same settings. When doing so, an editor window opens with the code required to recreate your `timescope` object.



Measurements Tab

In the **Measurements** tab, all measurements are made for a specified channel.





- **Data Cursors** -- Display the screen cursors.
- **Signal Statistics** -- Display the various statistics of the selected signal, such as maximum/minimum values, peak-to-peak values, mean, median, RMS.
- **Peak Finder** -- Display peak values for the selected signal.



Scale Axes

You can use the mouse to pan around the axes, and use the scroll button on your mouse to zoom in and out of the plot.

You can also use the buttons that appear when you hover over the plot window.

-  — Maximize the axes, hiding all labels and inseting the axes values.
-  — Zoom in on the plot.
-  — Pan around the axes.
-  — Autoscale the axes to fit the shown data.

For more details, see “Configure Time Scope MATLAB Object”.

Scan Matching Using Line Features: Estimate pose and covariance based on line features in lidar scans

The `matchScansLine` function calculates a relative pose and estimated covariance between lidar scan readings based on estimated linear features.

Trajectory Optimization Improvements: Specify longitudinal segments, deviation offsets, and additional waypoint parameters

The `trajectoryOptimalFrenet` contains two new properties: `NumSegments` and `DeviationOffset`. Increasing `NumSegments` divides the longitudinal terminal states into multiple segments to calculate more dynamic trajectories, but increases computational complexity. `DeviationOffset` specifies an offset on the cost calculation for trajectories to bias the optimal trajectory in a specific direction that deviated from the reference path.

You can also calculate trajectories based on a velocity-keeping behavior by specifying `NaN` for the `Longitudinal` field of the `TerminalStates` property.

The `plan` function no longer errors when a feasible trajectory is not found. The function now returns an empty trajectory vector and an exit flag is included in the output arguments.

Path Metrics Improvements: Specify `validatorVehicleCostmap` as a state validator

The `pathmetrics` function now supports the `validatorVehicleCostmap` as the state validator input to the function.

Ray Intersections for 3-D Maps: Calculate ray intersections, import, and export with a 3-D occupancy map

The `occupancyMap3D` object now supports the `rayIntersection` function for calculating the intersection of rays with obstacles in the environment. You can also import and export occupancy maps as a `.bt` or `.ot` octomap file.

Code Generation for Monte Carlo Localization: Generate C/C++ code using the `monteCarloLocalization` object

You can now generate code when using the `monteCarloLocalization` object.

Code Generation for Sampling-Based Planners: Generate C/C++ code using the `plannerRRT`, `plannerRRTStar`, and `plannerHybridAStar` objects

You can now generate code when using the `plannerRRT`, `plannerRRTStar`, and `plannerHybridAStar` objects.

Code Generation for Trajectory Optimization: Generate C/C++ code using the `trajectoryOptimalFrenet` object

You can now generate code when using the `trajectoryOptimalFrenet` object.

Access residuals and residual covariance of insfilters and `ahrs10filter`

You can access the residuals and residual covariance information of insfilters (`insfilterMARG`, `insfilterAsync`, `insfilterErrorState`, and `insfilterNonholonomic`) and `ahrs10filter` through their object functions such as `fusegps`, `fusegyro`, `residual`, and `residualgps`.

Model inertial measurement unit using IMU Simulink block

Use the IMU Simulink block to model an inertial measurement unit (IMU) composed of accelerometer, gyroscope, and magnetometer sensors.

Estimate device orientation using AHRS Simulink block

Use the AHRS Simulink block to estimate the orientation of a device from its accelerometer, magnetometer, and gyroscope sensor readings.

Calculate angular velocity from quaternions

Use `angvel` to calculate angular velocity from an array of quaternions.

Transform position and velocity between two frames to motion quantities in a third frame

Use `transformMotion` to transform position and velocity between two coordinate frames to motion quantities in a third coordinate frame.

R2019b

Version: 1.0

New Features

Simultaneous Localization and Mapping (SLAM): Create 2-D and 3-D occupancy maps using SLAM algorithm and lidar scan data

Use the SLAM algorithm to tune parameters for scan matching and loop-closure detection. The `LidarSLAM` object takes lidar scan data and builds a map as your vehicle moves through it. The algorithm generates a `poseGraph` and continuously optimizes edge-constraints based on detected loop closures. As more loop closures are detected, you can continuously build a map of your environment and adjust for odometry drift.

For an example using 2-D lidar scans, see [Implement Online Simultaneous Localization And Mapping \(SLAM\) with Lidar Scans](#).

For an example using 3-D lidar point clouds, see [Perform SLAM Using 3-D Lidar Point Clouds](#).

For more information, see [SLAM](#).

SLAM Map Builder App: Interactively modify loop closures and adjust overall map using SLAM algorithm

Use the **SLAM Map Builder** app to load and filter lidar scans and estimated poses from a log file or data in the workspace. Tune and run the SLAM algorithm to automatically build the map. Pause at any time to modify relative poses between scans. Modify or delete loop closures from the pose graph to improve the overall map. After you are done with the entire data set, output the map as an occupancy grid to use with path planning or other navigation algorithms.

Pose Estimation: Accurately estimate vehicle poses using IMU and GPS sensors and Monte Carlo Localization

Use localization and pose estimation algorithms to orient your vehicle in your environment. Sensor pose estimation uses filters to improve and combine sensor readings for IMU, GPS, and other sensors. Localization algorithms, like Monte Carlo localization and scan matching, estimate your pose in a known map using range sensor or lidar readings. Pose graphs track your estimated poses and can be optimized based on edge constraints and loop closures.

For more information, see [Localization and Pose Estimation](#)

Customizable Sampling-Based Path Planners: Plan a path from start to goal locations using RRT and RRT* algorithms

Plan paths through a 2-D environment using provided path planning algorithms:

- `plannerRRT`
- `plannerRRTStar`
- `plannerHybridAStar`

Specify parameters for provided 2-D state-space representations:

- `stateSpaceSE2`
- `stateSpaceDubins`

-
- `stateSpaceReedsShepp`

Validate your planned paths using occupancy maps or vehicle cost maps:

- `validatorOccupancyMap`
- `validatorVehicleCostmap`

Write your own custom state space or state validator using class interfaces:

- `nav.StateSpace`
- `nav.StateValidator`

Path-Planning Metrics: Use metrics to check and compare the output of path planners

Calculate path metrics to evaluate planned paths using the `pathmetrics` object. Check the clearance and smoothness based on your path constraints.

Sensor Models: Use simulated models for IMU, GPS, and range sensors

Perform sensor modeling and simulation for accelerometers, magnetometers, gyroscopes, altimeters, GPS, IMU, and range sensors. Analyze sensor readings, sensor noise, environmental conditions, and other configuration parameters. Generate trajectories to emulate these sensors traveling through a world, and calibrate the performance of your sensors.

Sensor models include:

- `gpsSensor`
- `imuSensor`
- `rangeSensor`

For other sensors and more information, see [Sensor Models](#).

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

Use the `waypointTrajectory` and `kinematicTrajectory` objects to generate trajectories for sensors or vehicles and control commands to send to your vehicle

