

# Navigation Toolbox™ Release Notes



# MATLAB® & SIMULINK®



## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://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](http://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](http://www.mathworks.com/patents) for more information.

<b>Grid-Based A* Path Planning: Plan path from start to goal location using the A* algorithm</b> .....	<b>1-2</b>
<b>Trajectory Optimal Frenet Utilities: More control in generating optimal trajectory in Frenet space</b> .....	<b>1-3</b>
<b>Dynamic Capsule-Based Obstacle: Model and simulate ego bodies and obstacles using collision primitive objects in environment</b> .....	<b>1-3</b>
<b>SLAM Map Builder Update on Data Import: Import lidar scans and odometry data from MATLAB workspace</b> .....	<b>1-3</b>
<b>Pose Graph Optimization Updates: Additional optimization functionality for pose graphs</b> .....	<b>1-3</b>
<b>Wheel Encoder Sensor Models: Simulate wheel encoder sensor readings</b> .....	<b>1-3</b>
<b>Wheel Encoder Odometry: Compute vehicle odometry using wheel encoder sensor readings</b> .....	<b>1-4</b>
<b>INS Sensor Model: Simulate inertial navigation and GPS readings</b> .....	<b>1-4</b>
<b>GNSS Sensor Model: Simulate GNSS receiver readings</b> .....	<b>1-4</b>
<b>Code Generation for 3-D Occupancy Map: Generate C/C++ code using occupancyMap3D object</b> .....	<b>1-4</b>
<b>State Space and State Validator for 3-D Occupancy Map</b> .....	<b>1-4</b>
<b>Generate Random Grid Maps for Planner Benchmarking</b> .....	<b>1-4</b>
<b>Adjust Inertial Sensor Fusion Performance Using Filter Tuner</b> .....	<b>1-5</b>
<b>GPS Device Object: Connect to GPS receiver from host computer</b> .....	<b>1-6</b>
<b>NMEA Parser Object: Parse data from standard NMEA sentences sent from GNSS receivers</b> .....	<b>1-6</b>
<b>Time Scope object: Bilevel measurements, triggers, and compiler support</b> .....	<b>1-6</b>
<b>New examples</b> .....	<b>1-6</b>

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

<b>Simultaneous Localization and Mapping (SLAM): Create 2-D and 3-D occupancy maps using SLAM algorithm and lidar scan data</b> .....	3-2
<b>SLAM Map Builder App: Interactively modify loop closures and adjust overall map using SLAM algorithm</b> .....	3-2

<b>Pose Estimation: Accurately estimate vehicle poses using IMU and GPS sensors and Monte Carlo Localization</b> .....	<b>3-2</b>
<b>Customizable Sampling-Based Path Planners: Plan a path from start to goal locations using RRT and RRT* algorithms</b> .....	<b>3-2</b>
<b>Path-Planning Metrics: Use metrics to check and compare the output of path planners</b> .....	<b>3-3</b>
<b>Sensor Models: Use simulated models for IMU, GPS, and range sensors</b> .....	<b>3-3</b>
<b>Trajectory and Waypoint Following Algorithms: Use built-in algorithms to generate trajectories and control commands for robots</b> .....	<b>3-3</b>



# R2020b

---

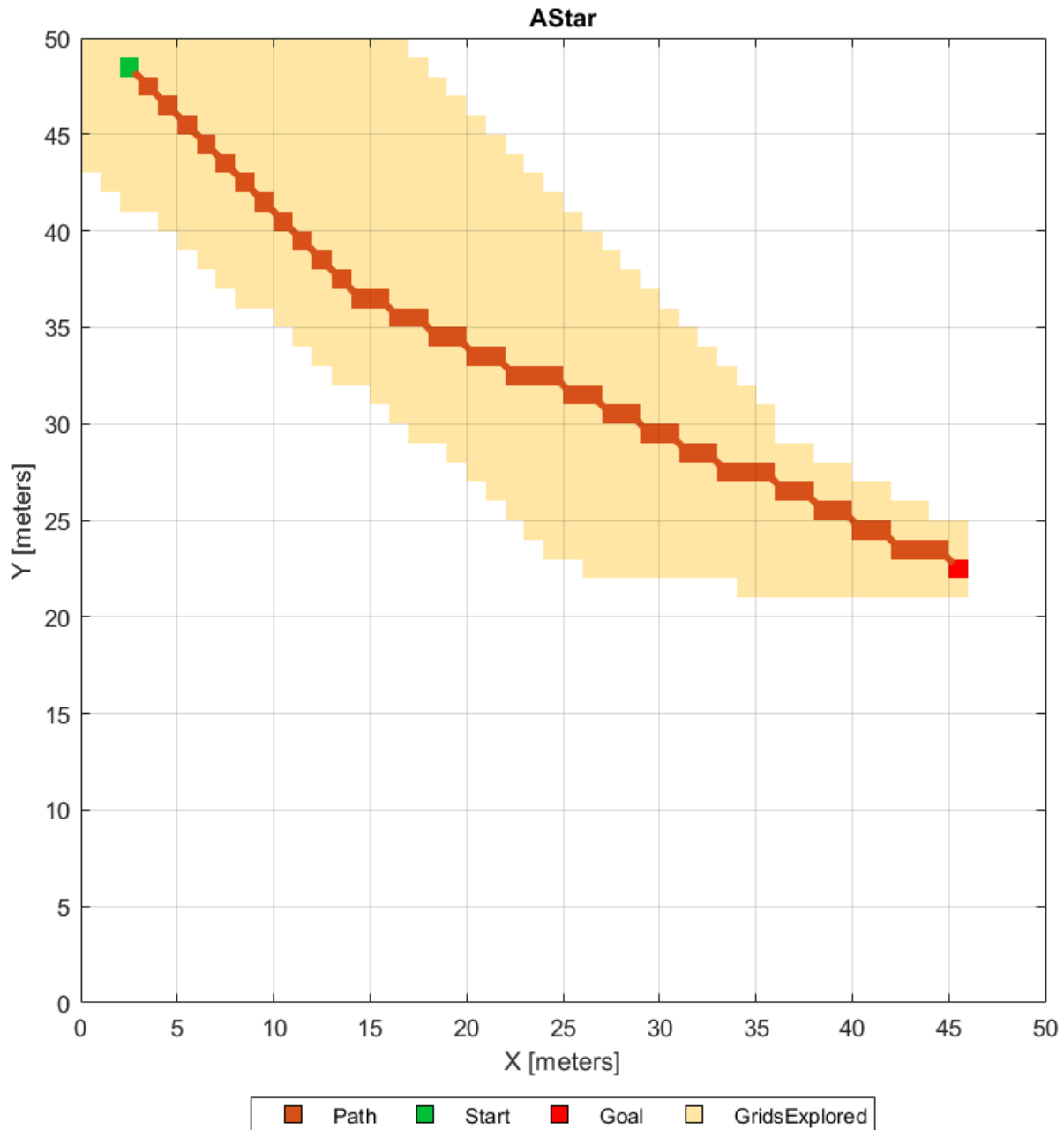
**Version: 1.2**

**New Features**

**Bug Fixes**

## Grid-Based A\* Path Planning: Plan path from start to goal location using the A\* algorithm

Plan a path on a 2-D grid map using the `plannerAStarGrid` object.





---

## Trajectory Optimal Frenet Utilities: More control in generating optimal trajectory in Frenet space

The `referencePathFrenet` object fits a smooth, piecewise continuous curve to the provided waypoints. Use the object functions to convert trajectories between global and Frenet coordinate systems, interpolate states along the path based on arc length, and query the closest point on the path from a global state.

The `trajectoryGeneratorFrenet` object generates trajectories between the initial and terminal states using fourth or fifth order polynomials. The trajectories are relative to a `referencePathFrenet` object. The `connect` function connects initial states to terminal states over a span of time.

For more details, see the “Highway Trajectory Planning Using Frenet Reference Path” example.

## Dynamic Capsule-Based Obstacle: Model and simulate ego bodies and obstacles using collision primitive objects in environment

The `dynamicCapsuleList` and `dynamicCapsuleList3D` objects manage two lists of collision primitive objects: ego bodies and obstacles. Use the object functions to dynamically add, remove, and update the geometry and future poses of ego bodies and obstacles in the environment. To validate stored trajectories, use the `checkCollision` function, which checks the collisions between ego bodies and obstacles at each time step.

For more details, see the “Highway Trajectory Planning Using Frenet Reference Path” example.

## SLAM Map Builder Update on Data Import: Import lidar scans and odometry data from MATLAB workspace

The **SLAM Map Builder** app now allows you to import lidar scans and odometry data from the MATLAB® workspace. To import data from the workspace, select **Import > Import from workspace**. Workspace import does not require a ROS Toolbox license.

## Pose Graph Optimization Updates: Additional optimization functionality for pose graphs

The `poseGraph` and `poseGraph3D` objects now support the `edgeResidualErrors` function. This function computes edge residual errors for each edge in the pose graph given the current pose node estimates.

The `trimLoopClosures` function optimizes pose graphs by removing bad loop closure edges that would otherwise cause edge residual errors.

The `poseGraphSolverOptions` function creates solver options for pose graph optimization.

## Wheel Encoder Sensor Models: Simulate wheel encoder sensor readings

Use provided sensor models to simulate wheel encoder sensor readings for various types of vehicles. Wheel encoder sensor models include:

- `wheelEncoderAckermann` object
- `wheelEncoderBicycle` object
- `wheelEncoderDifferentialDrive` object
- `wheelEncoderUnicycle` object

## **Wheel Encoder Odometry: Compute vehicle odometry using wheel encoder sensor readings**

Use provided odometry models to compute vehicle odometry of various types of vehicles using wheel encoder sensor readings. Wheel encoder odometry models include:

- `wheelEncoderOdometryAckermann` object
- `wheelEncoderOdometryBicycle` object
- `wheelEncoderOdometryDifferentialDrive` object
- `wheelEncoderOdometryUnicycle` object

## **INS Sensor Model: Simulate inertial navigation and GPS readings**

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

## **GNSS Sensor Model: Simulate GNSS receiver readings**

Use the `gnssSensor` object to simulate Global Navigation Satellite System (GNSS) receiver readings.

## **Code Generation for 3-D Occupancy Map: Generate C/C++ code using `occupancyMap3D` object**

You can now generate code when using the `occupancyMap3D` object.

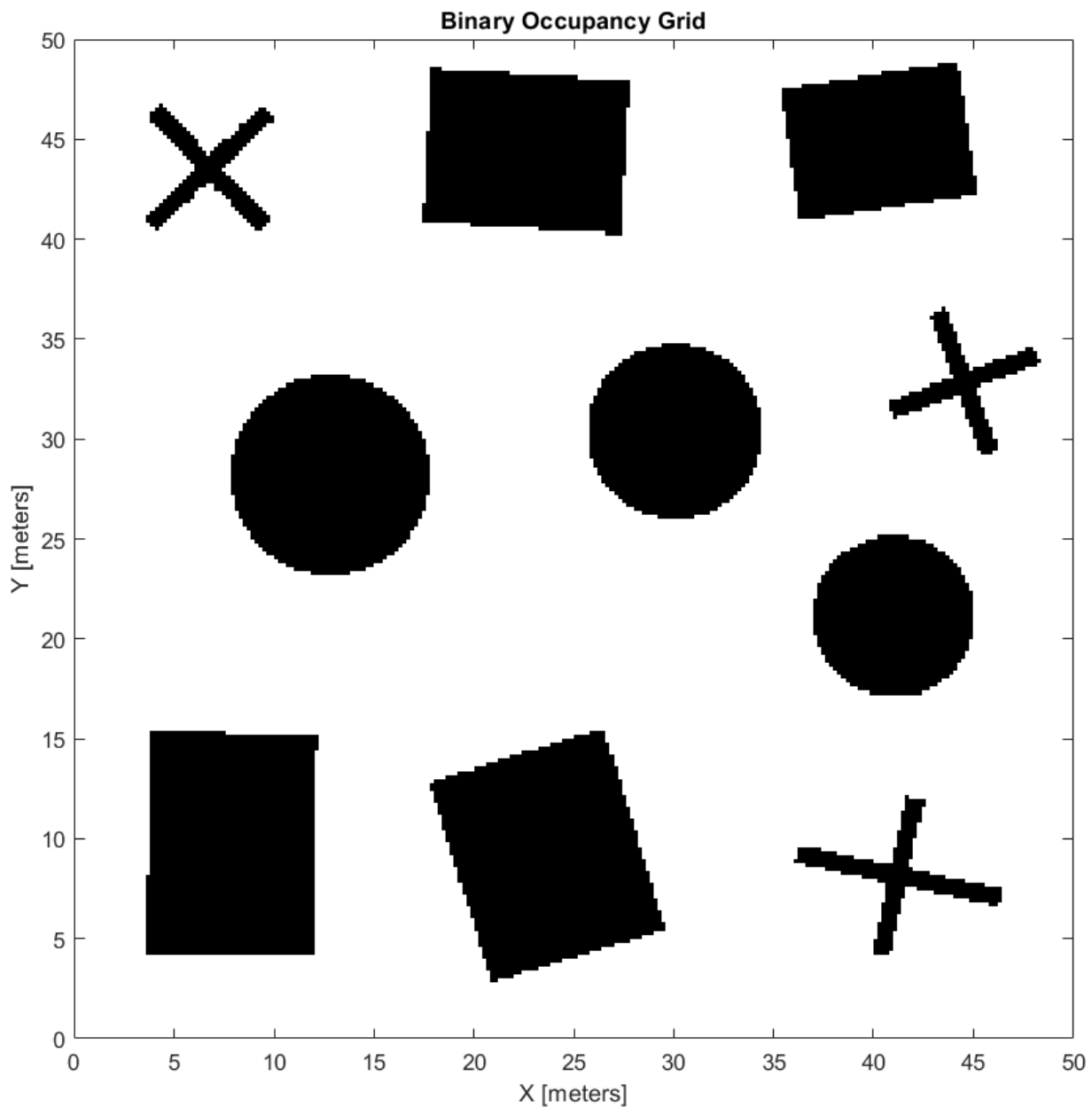
## **State Space and State Validator for 3-D Occupancy Map**

Use the `stateSpaceSE3` object to store parameters and states in the 3-D state-space representation.

Use the `validatorOccupancyMap3D` object to validate SE3 states in a 3-D occupancy map.

## **Generate Random Grid Maps for Planner Benchmarking**

Use the `mapClutter` function to generate a map with randomly scattered obstacles.



## Adjust Inertial Sensor Fusion Performance Using Filter Tuner

Use the `tune` function and the `tunerconfig` object to adjust the properties of the `imufilter`, `ahrsfilter`, and `insfilterAsync` objects. Adjusting these properties can impact performance.

For more details, see the "Automatic Tuning of the `insfilterAsync` Filter" example.

## **GPS Device Object: Connect to GPS receiver from host computer**

Use the `gpsdev` object to create a connection to a GPS receiver connected to the host computer running Navigation Toolbox™. You can create the object either by specifying the serial port as an input argument or by using the `serialport` object from MATLAB.

After you create the `gpsdev` object, use these functions to perform further actions:

- `read` - Obtain GPS data like latitude, longitude and altitude (LLA), ground speed, course, dilution of precisions, and GPS receiver time, along with timestamp and overrun information.
- `flush` - Clear the software buffers and serial port buffers.
- `writeBytes` - Write raw data to configure the GPS receiver.

## **NMEA Parser Object: Parse data from standard NMEA sentences sent from GNSS receivers**

Use the `nmeaParser` object to parse data from some of the standard NMEA (National Marine Electronics Association) sentences that are compliant with the NMEA 0183® specification.

The `nmeaParser` object parses data sent from GNSS receivers and identified by these NMEA message types: RMC, GGA, GSA, VTG, GLL, GST, ZDA, and HDT. The object outputs an array of structures corresponding to the data extracted from the requested NMEA message types.

## **Time Scope object: Bilevel measurements, triggers, and compiler support**

The `timescope` object now includes support for:

- Bilevel measurements - Measure transitions, overshoots, undershoots, and cycles.
- Triggers - Set triggers to sync repeating signals and pause the display when events occur.
- MATLAB Compiler™ support - Use the `mcc` function to compile code for deployment.

## **New examples**

This release contains several new examples:

- “Wheel Encoder Error Sources”
- “Highway Trajectory Planning Using Frenet Reference Path”
- “GNSS Simulation Overview”
- “Automatic Tuning of the `insfilterAsync` Filter”

# R2020a

---

**Version: 1.1**

**New Features**

**Bug Fixes**

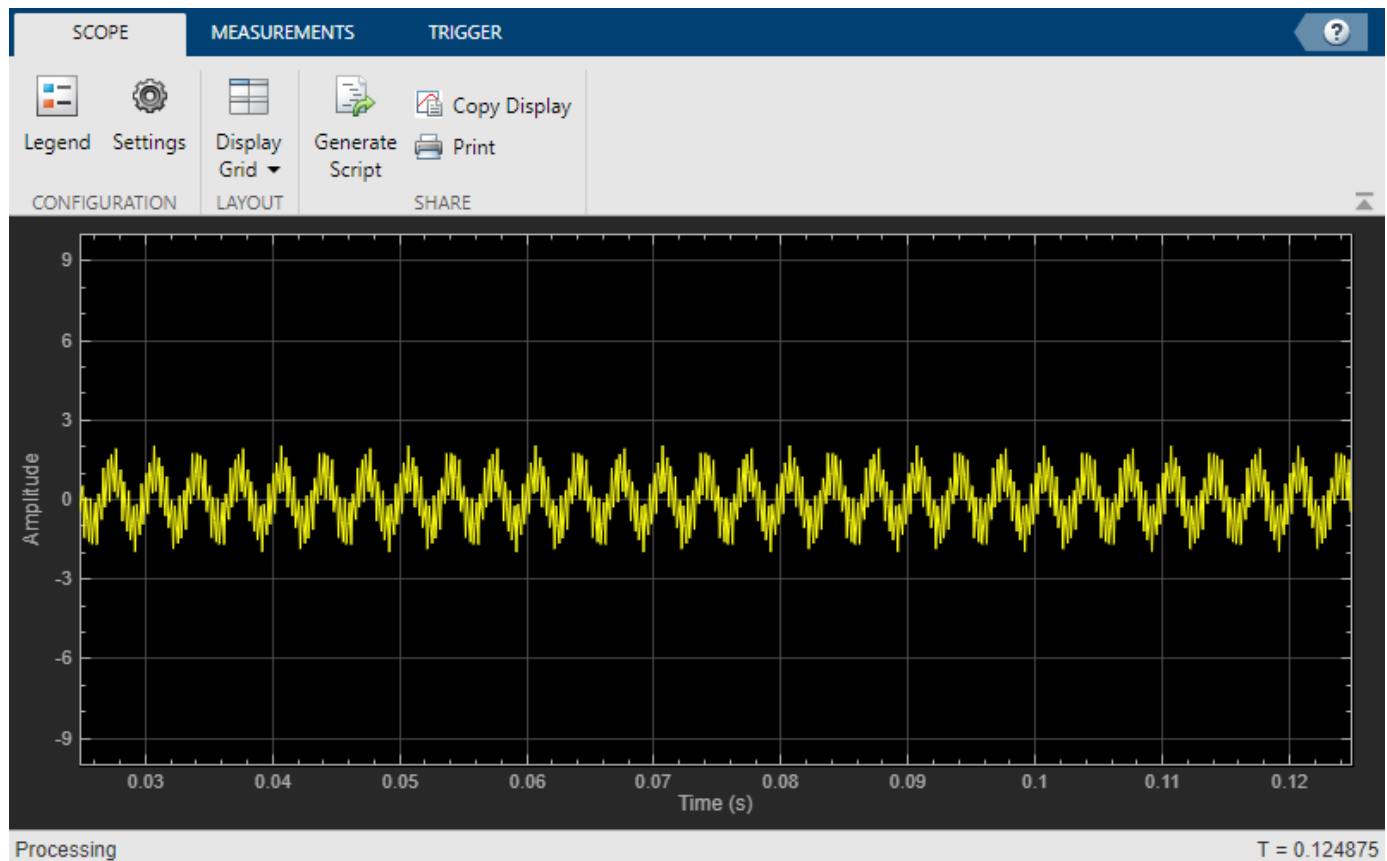
## New Time Scope object: Visualize signals in the time domain

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

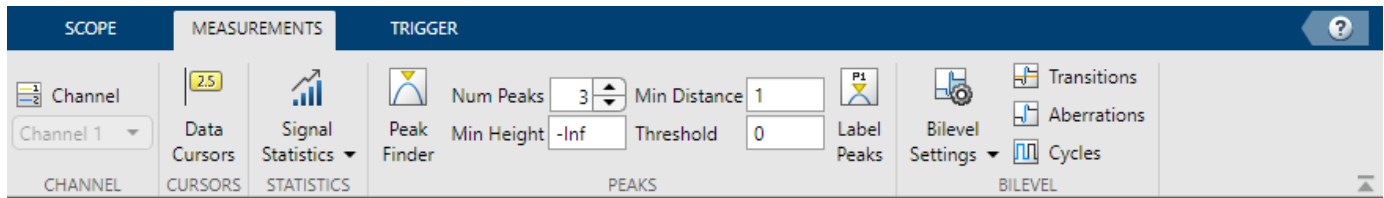
In the **Scope** tab, you can control the layout and configuration settings, and set the display settings of the Time Scope. You can also generate script to recreate your Time Scope with the same settings. When doing so, an editor window opens with the code required to recreate your `timescope` object.



### 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

