Robotics System Toolbox™

Reference

MATLAB® & SIMULINK®

R2015b

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Robotics System Toolbox™ Reference

Revision History

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

Contents

Classes — Alphabetical List

robotics.BinaryOccupancyGrid class

Package: robotics

Create occupancy grid with binary values

Description

`BinaryOccupancyGrid` creates a 2-D occupancy grid object, which you can use to represent and visualize a robot workspace, including obstacles. The integration of sensor data and position estimates create a spatial representation of the approximate locations of the obstacles.

Occupancy grids are used in robotics algorithms such as path planning. They are also used in mapping applications, such as for finding collision-free paths, performing collision avoidance, and calculating localization. You can modify your occupancy grid to fit your specific application.

Each cell in the occupancy grid has a value representing the occupancy status of that cell. An occupied location is represented as `true (1)` and a free location is represented as `false (0)`.

The two coordinate systems supported are world and grid coordinates. The world coordinates origin is defined by `GridLocationInWorld`, which defines the bottom-left corner of the map. The number and size of grid locations are defined by the `Resolution`. Also, the first grid location with index `(1,1)` begins in the top-left corner of the grid.

Construction

`map = robotics.BinaryOccupancyGrid(width,height)` creates a 2-D binary occupancy grid representing a work space of `width` and `height` in meters. The default grid resolution is one cell per meter.

`map = robotics.BinaryOccupancyGrid(width,height,resolution)` creates a grid with `resolution` specified in cells per meter. The map is in world coordinates by default. You can use any of the arguments from previous syntaxes.

`map = robotics.BinaryOccupancyGrid(rows,cols,resolution,'grid')` creates a 2-D binary occupancy grid of size (`rows`,`cols`).

`map = robotics.BinaryOccupancyGrid(p)` creates a grid from the values in matrix `p`. The size of the grid matches the size of the matrix, with each cell value interpreted from its location in the matrix. `p` contains any numeric or logical type with zeros (0) and ones (1).

`map = robotics.BinaryOccupancyGrid(p,resolution)` creates a `BinaryOccupancyGrid` object with `resolution` specified in cells per meter.

Input Arguments

`width` — Map width
double in meters

Map width, specified as a double in meters.

Data Types: `double`

`height` — Map height
double in meters

Map width, specified as a double in meters.

Data Types: `double`

`resolution` — Grid resolution
1 (default) | double in cells per meter

Grid resolution, specified as a double in cells per meter.

Data Types: `double`

`p` — Input occupancy grid
matrix of ones and zeros

Input occupancy grid, specified as a matrix of ones and zeros. The size of the grid matches the size of the matrix. Each matrix element corresponds to an occupied location (`1`) or free location (`0`).

Properties

`GridSize` — Number of rows and columns in grid
two-element horizontal vector

Number of rows and columns in grid, stored as a two-element horizontal vector of the form `[rows cols]`. This value is read only.

`Resolution` — Grid resolution

1 (default) | scalar in cells per meter

Grid resolution, stored as a scalar in cells per meter. This value is read only.

Data Types: `double`

`XWorldLimits` — Minimum and maximum values of *x*-coordinates

two-element vector

Minimum and maximum values of *x*-coordinates, stored as a two-element horizontal vector of the form `[min max]`. These values indicate the world range of the *x*-coordinates in the grid. This value is read only.

`YWorldLimits` — Minimum and maximum values of *y*-coordinates

two-element vector

Minimum and maximum values of *y*-coordinates, stored as a two-element vector of the form `[min max]`. These values indicate the world range of the *y*-coordinates in the grid. This value is read only.

`GridLocationWorld` — `[x,y]` world coordinates of grid

[0 0] (default) | two-element vector

`[x,y]` world coordinates of the bottom-left corner of the grid, specified as a two-element vector.

Data Types: `double`

Methods

Examples

Create and Modify Binary Occupancy Grid

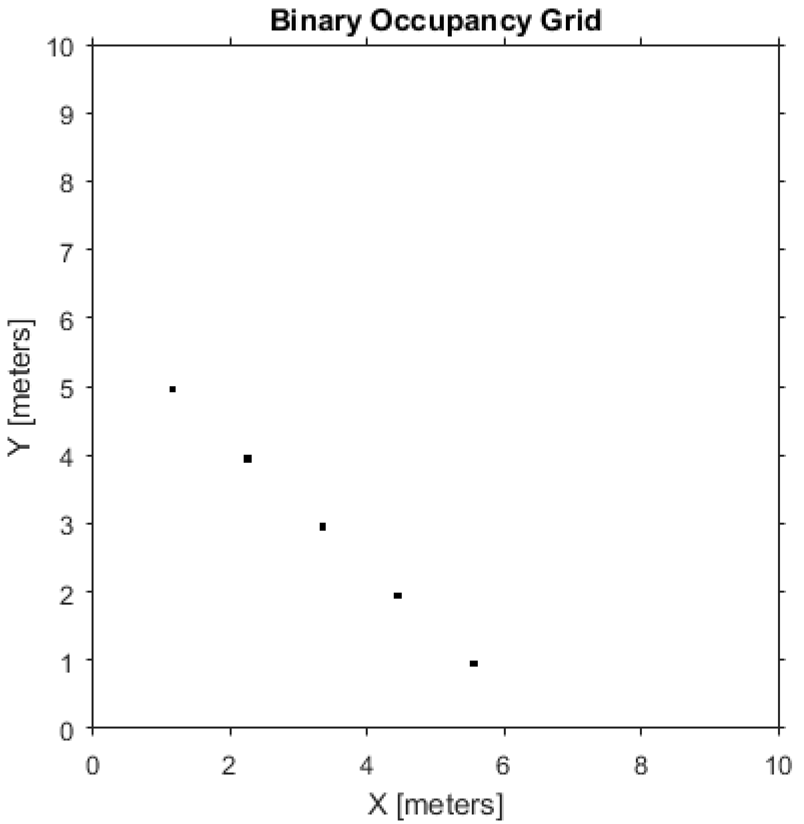
Create a 10m x 10m empty map.

```
map = robotics.BinaryOccupancyGrid(10,10,10);
```

Set occupancy of world locations and show map.

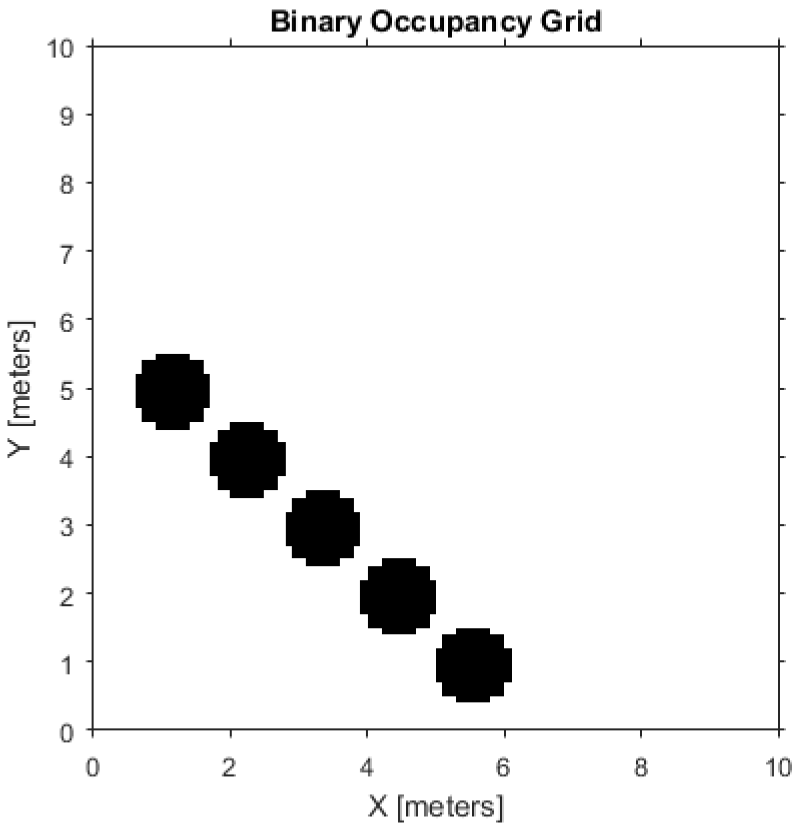
```
map = robotics.BinaryOccupancyGrid(10,10,10);
x = [1.2; 2.3; 3.4; 4.5; 5.6];
y = [5.0; 4.0; 3.0; 2.0; 1.0];

setOccupancy(map, [x y], ones(5,1))
figure
show(map)
```



Inflate occupied locations by a given radius.

```
inflate(map, 0.5)
figure
show(map)
```



Get grid locations from world locations.

```
ij = world2grid(map, [x y]);
```

Set grid locations to free locations.

```
setOccupancy(map, ij, zeros(5,1), 'grid')
figure
show(map)
```

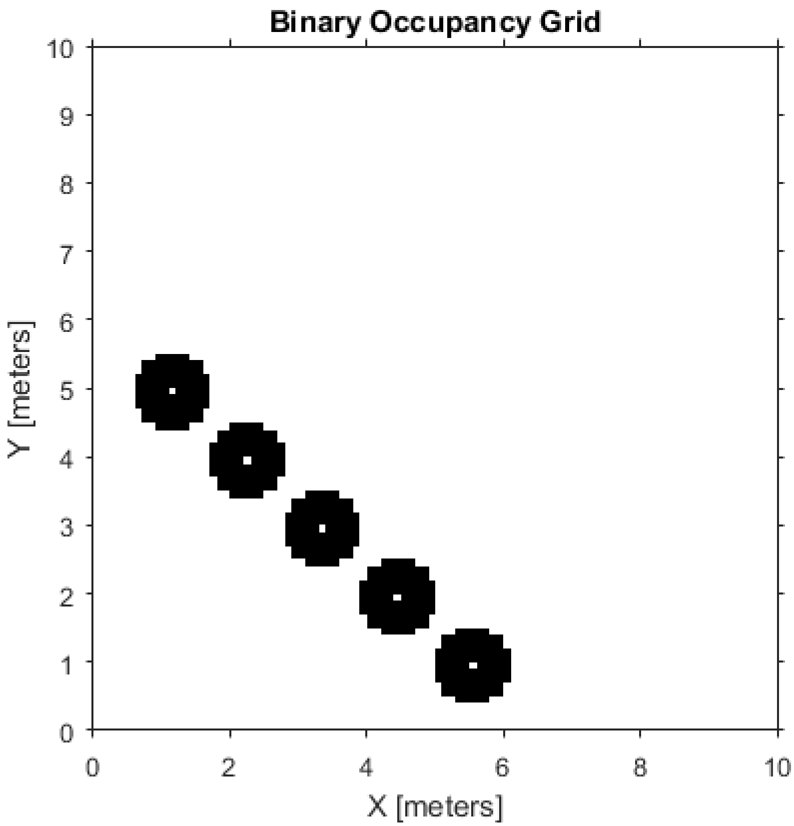


Image to Binary Occupancy Grid Example

This example shows how to convert an image to a binary occupancy grid for using with the Robotics System Toolbox®

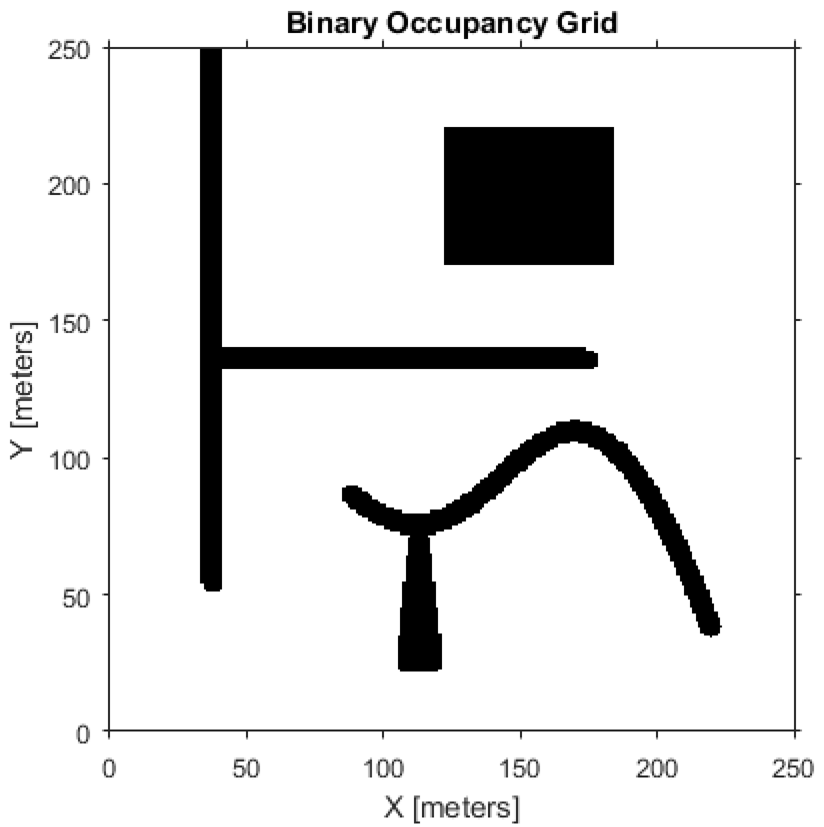
```
% Import Image
image = imread('imageMap.png');

% Convert to grayscale and then black and white image based on arbitrary
% threshold.
grayimage = rgb2gray(image);
bwimage = grayimage < 0.5;

% Use black and white image as matrix input for binary occupancy grid
```

```
grid = robotics.BinaryOccupancyGrid(bwimage);

show(grid)
```



- "Updating an Occupancy Grid From Range Sensor Data"

See Also

robotics.PRM | robotics.PurePursuit

More About

- "Occupancy Grids"

Introduced in R2015a

robotics.PRM class

Package: robotics

Create probabilistic roadmap path planner

Description

PRM creates a roadmap path planner object for the environment map specified in the Map property. The object uses the map to generate a roadmap, which is a network graph of possible paths in the map based on free and occupied spaces. You can customize the number of nodes, NumNodes, and the connection distance, ConnectionDistance, to fit the complexity of the map and find an obstacle-free path from a start to an end location.

After the map is defined, the PRM path planner generates the specified number of nodes throughout the free spaces in the map. A connection between nodes is made when a line between two nodes contains no obstacles and is within the specified connection distance.

After defining a start and end location, to find an obstacle-free path using this network of connections, use the findpath method. If findpath does not find a connected path, it returns an empty array. By increasing the number of nodes or the connection distance, you can improve the likelihood of finding a connected path, but tuning these properties is necessary. To see the roadmap and the generated path , use the visualization options in show. If you change any of the PRM properties, call update, show, or findpath to recreate the roadmap.

Construction

`planner = robotics.PRM` creates an empty roadmap with default properties. Before you can use the roadmap, you must specify a robotics.BinaryOccupancyGrid object in the Map property.

`planner = robotics.PRM(map)` creates a roadmap with map set as the Map property, where map is an object of the robotics.BinaryOccupancyGrid class.

`planner = robotics.PRM(map,numnodes)` sets the maximum number of nodes, numnodes, to the NumNodes property.

`planner = robotics.PRM(___,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments. `Name` is the property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Input Arguments

`map` — Map representation

`BinaryOccupancyGrid` object

Map representation, specified as a robotics.BinaryOccupancyGrid object. This object represents the environment of the robot. The object is a matrix grid with binary values indicating obstacles as `true` (1) and free locations as `false` (0).

`numnodes` — Maximum number of nodes in roadmap

50 (default) | scalar

Maximum number of nodes in roadmap, specified as a scalar. By increasing this value, the complexity and computation time for the path planner increases.

Properties

`'ConnectionDistance'` — Maximum distance between two connected nodes

`inf` (default) | scalar in meters

Maximum distance between two connected nodes, specified as the comma-separated pair consisting of `'ConnectionDistance'` and a scalar in meters. This property controls whether nodes are connected based on their distance apart. Nodes are connected only if no obstacles are directly in the path. By decreasing this value, the number of connections is lowered, but the complexity and computation time decreases as well.

`'Map'` — Map representation

`BinaryOccupancyGrid` object

Map representation, specified as the comma-separated pair consisting of 'Map' and a robotics.BinaryOccupancyGrid object. This object represents the environment of the robot. The object is a matrix grid with binary values indicating obstacles as `true` (1) and free locations as `false` (0).

'NumNodes' — Maximum distance between two connected nodes

inf (default) | scalar

Maximum distance between two connected nodes, specified as the comma-separated pair consisting of 'NumNodes' and a scalar. By increasing this value, the complexity and computation time for the path planner increases.

Methods

See Also

robotics.BinaryOccupancyGrid | robotics.PurePursuit

Related Examples

- "Path Planning in Environments of Different Complexity"

More About

- "Probabilistic Roadmaps (PRM)"

Introduced in R2015a

robotics.PurePursuit class

Package: robotics

Create controller to follow set of waypoints

Description

PurePursuit creates a controller object used to make a differential drive robot follow a set of waypoints. The object computes the linear and angular velocities for the robot. Given the current pose of the robot, you can calculate these velocities using the step method. Successive calls to step with updated poses provide updated velocity commands for the robot to follow a path along a desired set of waypoints. Use the MaxAngularVelocity and DesiredLinearVelocity properties to update the velocities based on the robot's performance.

The LookaheadDistanceproperty computes a look-ahead point on the path, which is a local goal for the robot. The angular velocity command is computed based on this point. Changing LookaheadDistance has a significant impact on the performance of the algorithm. A higher look-ahead distance results in a smoother trajectory for the robot, but can cause the robot to cut corners along the path. Too low of a look-ahead distance can result in oscillations in tracking the path, causing unstable behavior. For more information on the pure pursuit algorithm, see "Pure Pursuit Controller".

Construction

controller = robotics.PurePursuit creates a pure pursuit object, controller, that uses the pure pursuit algorithm to compute the linear and angular velocity inputs for a differential drive robot.

controller = robotics.PurePursuit(Name, Value) creates a pure pursuit object with additional options specified by one or more Name,Value pairs. Name is the property name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name-value pair arguments in any order as Name1,Value1,...,NameN,ValueN. Properties not specified retain their default values.

Properties

'DesiredLinearVelocity' — Desired constant linear velocity

0.1 (default) | scalar in meters per second

Desired constant linear velocity, specified as the comma-separated pair consisting of 'DesiredLinearVelocity' and a scalar in meters per second. The controller assumes that the robot drives at a constant linear velocity and that the computed angular velocity is independent of the linear velocity.

Data Types: double

'LookaheadDistance' — Look-ahead distance

1.0 (default) | scalar in meters

Look-ahead distance, specified as the comma-separated pair consisting of 'LookaheadDistance' and a scalar in meters. The look-ahead distance changes the response of the controller. A robot with higher look-ahead distance produces smooth paths but takes larger turns at corners. A robot with smaller look-ahead distance follows the path closely and takes sharp turns, but can produce oscillations in the path.

Data Types: double

'MaxAngularVelocity' — Maximum angular velocity

1.0 (default) | scalar in radians per second

Maximum angular velocity, specified as the comma-separated pair consisting of 'MaxAngularVelocity' and a scalar in radians per second. The controller saturates the absolute angular velocity output at the given value.

Data Types: double

'Waypoints' — Waypoints

[ ] (default) | n-by-2 array

Waypoints, specified as an n-by-2 array of [x y] pairs, where n is the number of waypoints. You can generate the waypoints from the PRM class or from another source.

Data Types: double

Methods

See Also

robotics.BinaryOccupancyGrid | robotics.PRM

Related Examples

- “Path Following for a Differential Drive Robot”

More About

- “Pure Pursuit Controller”

Introduced in R2015a

robotics.VectorFieldHistogram class

Package: robotics

Avoid obstacles using vector field histogram

Description

The vector field histogram (VFH) class enables your robot to avoid obstacles based on range sensor data. Given a range sensor reading in terms of ranges and angles, and a target direction to drive toward, the VFH controller computes an obstacle-free steering direction.

The class uses the VFH+ algorithm to compute the obstacle-free direction. First, the algorithm takes the ranges and angles from range sensor data and builds a polar histogram for obstacle locations. Then, it uses the input histogram thresholds to calculate a binary histogram that indicates occupied and free directions. Finally, the algorithm computes a masked histogram, which is computed from the binary histogram based on the minimum turning radius of the robot.

The algorithm selects multiple steering directions based on the open space and possible driving directions. A cost function, with weights corresponding to the previous, current, and target directions, calculates the cost of different possible directions. The algorithm then returns an obstacle-free direction with minimal cost. Using the obstacle-free direction, you can input commands to move your robot in that direction.

To use this class for your own application and environment, you must tune the properties of the algorithm. Property values depend on the type of robot, the range sensor, and the hardware you use.

Construction

`VFH = robotics.VectorFieldHistogram` returns a vector field histogram object that computes the obstacle-free steering direction using the `VFH+` algorithm.

`VFH = robotics.VectorFieldHistogram(Name,Value)` returns a vector field histogram object with each specified property name set to the specified value. You can specify multiple properties in any order as `Name1,Value1,...NameN,ValueN`.

Properties

`NumAngularSectors` — Number of angular sectors in histogram

180 (default) | scalar

Number of angular sectors, specified as a scalar. This property defines the number of bins used to create the histograms. This property is non-tunable. You can only set this when the object is initialized.

`DistanceLimits` — Limits for range readings

[0.05 2] (default) | 2-element vector

Limits for range readings, specified as a 2-element vector. The range readings specified in the step function are considered only if they fall within the distance limits. Use the lower distance limit to ignore false positives from poor sensor performance at lower ranges. Use the upper limit to ignore obstacles that are too far from the robot.

`RobotRadius` — Radius of the robot in meters

0.1 (default) | scalar

Radius of the robot in meters, specified as a scalar. This dimension defines the smallest circle that can circumscribe your robot. The robot radius is used to account for robot size when computing the obstacle-free direction.

`SafetyDistance` — Safety distance around the robot

0.1 (default) | scalar

Safety distance around the robot, specified as a scalar. This is a safety distance to leave around the robot position in addition to `RobotRadius`. The robot radius and safety distance are used to compute the obstacle-free direction.

`MinTurningRadius` — Minimum turning radius at current speed

0.1 (default) | scalar

Minimum turning radius for the robot moving at its current speed, specified as a scalar.

`TargetDirectionWeight` — Cost function weight for target direction

5 (default) | scalar

Cost function weight for moving toward the target direction, specified as a scalar. To follow a target direction, set this weight to be higher than the sum of

`CurrentDirectionWeight` and `PreviousDirectionWeight`. To ignore the target direction cost, set this weight to zero.

`CurrentDirectionWeight` — Cost function weight for current direction
2 (default) | scalar

Cost function weight for moving the robot in the current heading direction, specified as a scalar. Higher values of this weight produces efficient paths. To ignore the current direction cost, set this weight to zero.

`PreviousDirectionWeight` — Cost function weight for previous direction
2 (default) | scalar

Cost function weight for moving in the previously selected steering direction, specified as a scalar. Higher values of this weight produces smoother paths. To ignore the previous direction cost, set this weight to zero.

`HistogramThresholds` — Thresholds for binary histogram computation
[3 10] (default) | 2-element vector

Thresholds for binary histogram computation, specified as a 2-element vector. The algorithm uses these thresholds to compute the binary histogram from the polar obstacle density. Polar obstacle density values higher than the upper threshold are represented as occupied space (1) in the binary histogram. Values smaller than the lower threshold are represented as free space (0). Values that fall between the limits are set to the values in the previous binary histogram, with the default being free space (0).

Methods

Examples

Create a Vector Field Histogram Object

This example shows how to create a Vector Field Histogram (VFH) object and calculate a steering direction based on input laser scan data.

Create VFH object

```
vfh = robotics.VectorFieldHistogram;
```

Input laser scan data and target direction.

```
ranges = 10*ones(1, 500);
ranges(1, 225:275) = 1.0;
angles = linspace(-pi, pi, 500);
targetDir = 0;
```

Compute obstacle-free steering direction

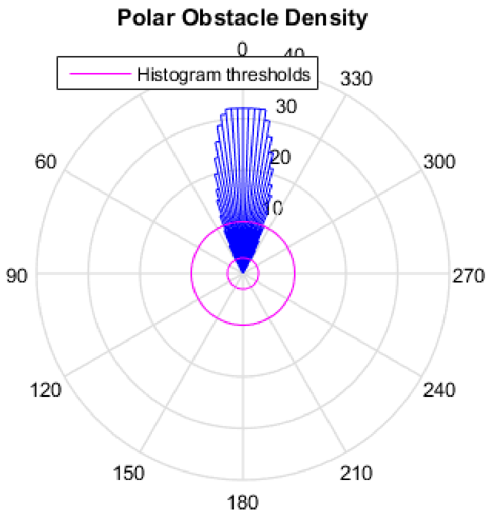
```
steeringDir = step(vfh, ranges, angles, targetDir)
```

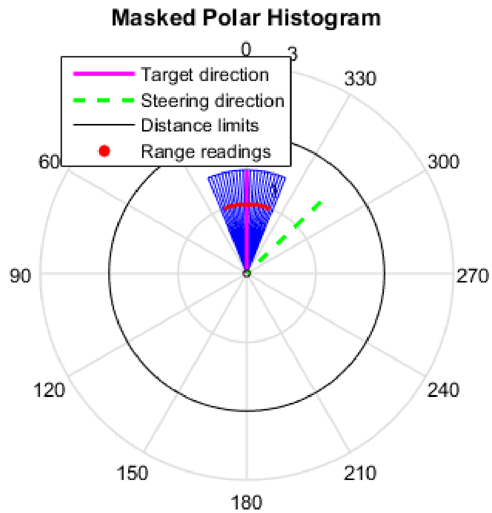
```
steeringDir =

   -0.8014
```

Visualize the VectorFieldHistogram computation

```
h = figure;
set(h,'Position',[50 50 800 400])
show(vfh);
```





References

[1] Borenstein, J., and Y. Koren. "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots." *IEEE Journal of Robotics and Automation*. Vol. 7, No. 3, 1991, pp.278–88.

[2] Uldrich, Iwan, and Johann Borenstein. "VFH : Reliable Obstacle Avoidance for Fast Mobile Robots." *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*. 1998, pp. 1572–1577.

See Also

robotics.PRM

More About

- “Vector Field Histogram”

Introduced in R2015b

Functions — Alphabetical List

Using BagSelection Objects

Create rosbag selection

Description

The `BagSelection` object is an index of the messages within a `rosbag`. You can use it to extract message data from a rosbag, select messages based on specific criteria, or create a `timeseries` of the message properties.

Examples

Create `rosbag` Selection Using `BagSelection` Object

Create a `BagSelection` object from a `rosbag` log file and parse out specific messages based on the selected criteria.

Set the path to the logfile

```
filepath = fullfile(fileparts(which('ROSWorkingWithRosbagsExample')), 'data', 'ex_multi
```

Create a `BagSelection` object of all the messages in the log file.

```
bagMsgs = robotics.ros.Bag.parse(filepath);
```

Select a subset of the messages based on their timestamp and topic.

```
bagMsgs2 = select(bagMsgs, 'Time', ...
           [bagMsgs.StartTime bagMsgs.StartTime + 1], 'Topic', '/odom');
```

Retrieve the messages in the selection as a cell array.

```
msgs = readMessages(bagMsgs2);
```

Return certain message properties as a time series.

```
ts = timeseries(bagMsgs2, 'Pose.Pose.Position.X', ...
           'Twist.Twist.Angular.Y');
```

- "Working with rosbag Logfiles"

Properties

`FilePath` — Absolute path to the `rosbag` file
string

This property is read only.

Absolute path to the `rosbag` file, specified as a string.

Data Types: `char`

`StartTime` — Timestamp of the first message in the selection
scalar

This property is read only.

Timestamp of the first message in the selection, specified as a scalar in seconds.

Data Types: `double`

`EndTime` — Timestamp of the last message in the selection
scalar

This property is read only.

Timestamp of the last message in the selection, specified as a scalar in seconds.

Data Types: `double`

`NumMessages` — Number of messages in the selection
scalar

This property is read only.

Number of messages in the selection, specified as a scalar. When you first load a `rosbag`, this property contains the number of messages in the `rosbag`. Once you select a subset of messages with `select`, the property shows the number of messages in this selection.

Data Types: `double`

`AvailableTopics` — Table of topics in the selection
table

This property is read only.

Table of topics in the selection, specified as a table. Each row in the table lists one topic, the number of messages for this topic, the message type, and the definition of the type.

Data Types: `table`

`MessageList` — List of messages in the selection
table

This property is read only.

List of messages in the selection, specified as a table. Each row in the table lists one message.

Data Types: `table`

Object Functions

readMessages	Read messages from rosbag
select	Select subset of messages in rosbag
timeseries	Creates a time series object for selected message properties

Create Object

rosbag	Open and parse rosbag log file
select	Select subset of messages in rosbag

See Also

`readMessages` | `select` | `timeseries`

More About

- "ROS Log Files (rosbags)"

Introduced in R2015a

Using Core Objects

Create ROS Core

Description

The ROS Core encompasses many key components and nodes that are essential for the ROS network. You must have exactly one ROS core running in the ROS network in order for nodes to communicate. Using this class allows the creation of a ROS core in MATLAB®. Once the core is created, you can connect to it by calling `rosinit` or `robotics.ros.Node`.

Examples

Create ROS Core

Create ROS Core on `localhost` and default port `11311`.

```
core = robotics.ros.Core;
```

Create ROS Core on Specific Port

Create ROS Core on `localhost` and port `12000`.

```
core = robotics.ros.Core(12000);
```

- “Connecting to a ROS Network”

Properties

`Port` — Network port at which the ROS master is listening

11311 (default) | scalar

This property is read only.

Network port at which the ROS master is listening, returned as a scalar.

MasterURI — The URI on which the ROS master can be reached
'http://<HOSTNAME>:11311' (default) | string

This property is read only.

The URI on which the ROS master can be reached, returned as a string. The MasterURI is constructed based on the host name of your computer. If your host name is not valid, the IP address of your first network interface is used.

Create Object

core = robotics.ros.Core returns a Core object and starts a ROS core in MATLAB. This ROS core has a default port of 11311. MATLAB only allows the creation of one core on any given port and displays an error if another core is detected on the same port.

core = robotics.ros.Core(port) starts a ROS core at the specified port, port.

See Also

Using Node Objects | rosinit

More About

- “ROS Network Setup”

External Websites

- ROS Core

Introduced in R2015a

Using Node Objects

Start ROS node and connect to ROS master

Description

The `robotics.ros.Node` object represents a ROS node in the ROS network. The object enables you to communicate with the rest of the ROS network. You must create a node before you can use other ROS functionality, such as publishers, subscribers, and services.

You can create a ROS node using the `rosinit` function, or by calling `robotics.ros.Node`:

- `rosinit` — Creates a single ROS node in MATLAB. You can specify an existing ROS master, or the function creates one for you. The `Node` object is not visible.
- `robotics.ros.Node`— Creates multiple ROS nodes for use on the same ROS network in MATLAB.

Examples

Create Multiple ROS Nodes

Create multiple ROS nodes. Use the `Node` object with publishers, subscribers, and other ROS functionality to specify with which node you are connecting to.

Create a ROS master.

```
master = robotics.ros.Core;
```

Initialize multiple nodes.

```
node1 = robotics.ros.Node('/test_node_1');
node2 = robotics.ros.Node('/test_node_2');
```

Use these nodes to perform seperate operations and send seperate messages. A message published by `node1` can be accessed by a subscriber running in `node2`.

```
pub = robotics.ros.Publisher(node1,'/chatter','std_msgs/String');
```

```
sub = robotics.ros.Subscriber(node2,'/chatter','std_msgs/String');

msg = rosmessage('std_msgs/String');
msg.Data = 'Message from Node 1';
```

Send a message from `node1`. The subscriber attached to `node2` will receive the message.

```
send(pub,msg) % Sent from node 1
pause(1) % Wait for message to update
sub.LatestMessage


ans = 

  ROS String message with properties:

    MessageType: 'std_msgs/String'
           Data: 'Message from Node 1'

  Use showdetails to show the contents of the message
```

Clear the ROS network of publisher, subscriber, and nodes. Delete the `Core` object to shut down the ROS master.

```
clear('pub','sub','node1','node2')
clear('master')
```

Connect to Multiple ROS Masters

Connecting to multiple ROS masters is possible using MATLAB. Typically, this need arises when you try to connect to multiple robots at the same time. When you connect to multiple ROS masters, you are connecting to two different ROS networks, each with their own set of nodes and topics.

MATLAB can connect to multiple ROS masters by creating multiple nodes. For example, assume that you have two robots that are wirelessly connected to your computer running MATLAB. The first robot has IP address 192.168.1.1 and the second robot has IP address 192.168.1.2.

Create one node connecting to the first robot and one node connecting to the second robot.

```
node1 = robotics.ros.Node('/test_node_1','192.168.1.1');
node2 = robotics.ros.Node('/test_node_2','192.168.1.2');
```

You can now create subscribers, publishers, and other ROS entities and attach them to each node. For

Create a subscriber for a topic on robot 1.

```
sub = robotics.ros.Subscriber(node1, '/topic_robot1');
```

Create a publisher and a message to send with this publisher.

```
pub = robotics.ros.Publisher(node2, '/topic_robot2');
msg = rosmessage(pub);
```

The same steps apply for service clients, service servers, parameter trees, and transformation tree objects. You can now write MATLAB code that interacts with both robots at the same time.

Properties

`Name` — Name of the node

string

Name of the node, specified as a string. The node name must be a valid ROS graph name. See ROS Names.

`MasterURI` — URI of the ROS master

string

URI of the ROS master, specified as a string. The node is connected to the ROS master with the given URI.

`NodeURI` — URI for the node

string

URI for the node, specified as a string. The node uses this URI to advertise itself on the ROS network for others to connect to it.

`CurrentTime` — Current ROS network time

`Time` object

Current ROS network time, specified as a `Time` object. For more information, see `rostime`.

Create Object

`N = robotics.ros.Node(Name)` initializes the ROS node with `Name` and tries to connect to the ROS master at default URI, `http://localhost:11311`.

`N = robotics.ros.Node(Name,Host)` tries to connect to the ROS master at the specified IP address or host name, `Host` using the default port number, `11311`.

`N = robotics.ros.Node(Name,Host,Port)`tries to connect to the ROS master with port number, `Port`.

`N = robotics.ros.Node(Name,MasterURI,Port)` tries to connect to the ROS master at the specified IP address, `MasterURI`.

`N = robotics.ros.Node( ___ ,'NodeHost',HostName)` specifies the IP address or host name that the node uses to advertise itself to the ROS network. Examples include `'192.168.1.1'` or `'comp-home'`. You can use any of the arguments from the previous syntaxes.

See Also

`rosinit` | `rosshutdown`

More About

- “ROS Network Setup”

External Websites

- ROS Nodes

Introduced in R2015a

Using ParameterTree Objects

Access ROS parameter server

Description

A robotics.ros.ParameterTree object communicates with the ROS parameter server. The ROS parameter server can store strings, integers, doubles, Booleans, and cell arrays. The parameters are accessible globally over the ROS network. You can use these parameters to store static data such as configuration parameters.

Examples

Create ROS ParameterTree Object and Modify Parameters

Start the ROS master and create a ROS node.

```
master = robotics.ros.Core;
node = robotics.ros.Node('/test1');
```

Create the parameter tree object.

```
ptree = robotics.ros.ParameterTree(node);
```

Set multiple parameters.

```
set(ptree,'DoubleParam',1.0)
set(ptree,'CharParam','test')
set(ptree,'CellParam',{{'test'},{1,2}});
```

View the available parameters.

```
parameters = ptree.AvailableParameters
```

```
parameters =

    '/CellParam'
    '/CharParam'
    '/DoubleParam'
```

Get a parameter value.

```
data = get(ptree,'CellParam')
```

```
data = 

    {1x1 cell}    {1x2 cell}
```

Search for a parameter name.

```
search(ptree,'char')
```

```
ans = 

    '/CharParam'
```

Delete the parameter tree and ROS node. Shut down the ROS master.

```
clear('ptree','node')
clear('master')
```

- "Accessing the ROS Parameter Server"

Properties

`AvailableParameters` — List of parameter names on the server

cell array

This property is read only.

List of parameter names on the server, specified as a cell array.

Example: `{'/myParam';'/robotSize';'/hostname'}`

Data Types: `cell`

Object Functions

get	Get ROS parameter value

has	Check if ROS parameter name exists
search	Search ROS network for parameter names
set	Set value of ROS parameter; add new parameter
del	Delete a ROS parameter

Create Object

`ptree = robotics.ros.ParameterTree(node)` returns a `ParameterTree` object to communicate with the ROS parameter server. The parameter tree attaches to the ROS node, `node`. To connect to the global node, specify `node` as `[]`.

See Also

`del` | `get` | `has` | `rosparam` | `search` | `set`

Introduced in R2015a

Using Publisher Objects

Create ROS publisher

Description

The `Publisher` object represents a publisher on the ROS network. The object publishes to an available topic or to a topic that it creates. This topic has an associated message type. When the `Publisher` object publishes a message to the topic, all subscribers to the topic receive this message. The same topic can have multiple publishers and subscribers.

You can create a `Publisher` object using the `rospublisher` function, or by calling `robotics.ros.Publisher`:

- `rospublisher` only works with the global node using `rosinit`. It does not require a node object handle as an argument.
- `robotics.ros.Publisher` works with additional nodes that are created using `robotics.ros.Node`. It requires a node object handle as the first argument.

Examples

Create ROS Publisher with `rospublisher` and View Properties

Create a ROS publisher and view the associated properties for the `robotics.ros.Publisher` object. Add a subscriber to view the updated properties.

Start ROS master.

```
rosinit
```

```
Initializing ROS master on http://bat5236win64:11311/.
Initializing global node /matlab_global_node_14586 with NodeURI http://bat5236win64:55
```

Create a publisher and view its properties.

```
pub = rospublisher('/chatter','std_msgs/String');

topic = pub.TopicName
subCount = pub.NumSubscribers
```

```
topic =

/chatter


subCount =

     0
```

Subscriber to the publisher topic and view the changes in the `NumSubscribers` property.

```
sub = rossubscriber('/chatter');
pause(1)

subCount = pub.NumSubscribers

rosshutdown
```

```
subCount =

     1

Shutting down global node /matlab_global_node_14586 with NodeURI http://bat5236win64:5
Shutting down ROS master on http://bat5236win64:11311/.
```

Use ROS Publisher Object

Create a `Publisher` object using the class constructor.

Start the ROS master.

```
master = robotics.ros.Core;
```

Create a ROS node, which connects to the master.

```
node = robotics.ros.Node('/test1');
```

Create a publisher and send string data. The publisher attaches to the node object in the first argument.

```
pub = robotics.ros.Publisher(node, '/robotname', 'std_msgs/String');
```

```
msg = rosmessage(rostype.std_msgs_String);
msg.Data = 'robot1';
send(pub,msg);
```

Clear the publisher and ROS node. Shut down the ROS master.

```
clear('pub','node')
clear('master')
```

- “Exchanging Data with ROS Publishers and Subscribers”

Properties

`TopicName` — Name of the published topic

string

This property is read only.

Name of the published topic, specified as a string. If the topic does not exist, the object creates the topic using its associated message type.

Example: `'/chatter'`

Data Types: `char`

`MessageType` — Message type of published messages

string

This property is read only.

Message type of published messages, specified as a string. This message type remains associated with the topic and must be used for new messages published.

Example: `'std_msgs/String'`

Data Types: `char`

`IsLatching` — Indicator of whether publisher is latching

`true` (default) | `false`

This property is read only.

Indicator of whether publisher is latching, specified as `true` or `false`. A publisher that is latching saves the last sent message and resends it to any new subscribers.

Data Types: `logical`

`NumSubscribers` — Number of subscribers
integer

This property is read only.

Number of subscribers to the published topic, specified as an integer.

Data Types: `double`

Object Functions

send	Publish ROS message to topic
rosmessage	Create ROS messages

Create Object

`pub = robotics.ros.Publisher(node,topicname)` creates a publisher for a topic with name, `topicname`. This topic must exist already so the message type can be retrieved. `node` is the `robotics.ros.Node` object handle that this publisher attaches to. If `node` is specified as `[]`, the publisher tries to attach to the global node.

`pub = robotics.ros.Publisher(node,topicname,type)` creates a publisher with specified message type, `type`. If the topic already exists, MATLAB checks the message type and displays an error if the input type differs. If the ROS master topic list already contains a matching topic, the ROS master adds the MATLAB global node to the list of publishers for that topic.

`pub = robotics.ros.Publisher( ___ , 'IsLatching',value)` specifies if the publisher is latching with a Boolean, `value`. If a publisher is latching, it saves the last sent message and sends it to any new subscribers. By default, `IsLatching` is enabled. You can use any combination of previous inputs with this syntax.

See Also

`rosmessage` | `rospublisher` | `rossubscriber` | `send`

Introduced in R2015a

Using ServiceClient Objects

Connect to ROS service server

Description

Use `robotics.ros.ServiceClient` to create a ROS service client object. This service client uses a persistent connection to send requests to, and receive responses from, a ROS service server. The connection persists until the service client is deleted or the service server becomes unavailable.

Examples

Use ROS Service Server with `ServiceServer` and `ServiceClient` Objects

Create a ROS service serve by creating a `ServiceServer` object and use `ServiceClient` objects to request information over the network. The callback function used by the server takes a string, reverses it, and returns the reversed string.

Start the ROS master and node.

```
master = robotics.ros.Core;
node = robotics.ros.Node('/test');
```

Create a service server. This server expects a string as a request and responds with a string based on the callback.

```
server = robotics.ros.ServiceServer(node,'/data/string',...
                                    'roseus/StringString');
```

Create a callback function. This function takes an input string as the `Str` property of `req` and returns it as the `Str` property of `resp`. You must create and save this function seperately. `req` is a ROS message you create using `rosmessage`.

```
% Copyright 2015 The MathWorks, Inc.
```

```
function [resp] = flipString(~,req,resp)
% FLIPSTRING Reverses the order of a string in REQ and returns it in RESP.
resp.Str = fliplr(req.Str);
end
```

Save this code as a file named `flipString.m` to a folder on your MATLAB® path.

Assign the callback function for incoming service calls.

```
server.NewRequestFcn = @flipString;
```

Create a service client and connect to the service server. Create a request message based on the client.

```
client = robotics.ros.ServiceClient(node,'/data/string');
request = rosmessage(client);
request.Str = 'hello world';
```

Send a service request and wait for a response. Specify that the service waits 3 seconds for a response.

```
response = call(client,request,'Timeout',3)
```

```
response = 

  ROS StringStringResponse message with properties:

    MessageType: 'roseus/StringStringResponse'
            Str: 'dlrow olleh'

  Use showdetails to show the contents of the message
```

The response is a flipped string from the request message.

Clear the service client, service server, and ROS node. Shut down the ROS master.

```
clear('client', 'server', 'node')
clear('master')
```

- “Calling and Providing ROS Services”

Properties

`ServiceName` — Name of the service
string

This property is read only.

Name of the service, specified as a string.

Example: `'/gazebo/get_model_state'`

`ServiceType` — Type of service
string

This property is read only.

Type of service, specified as a string.

Example: `'rostype.gazebo_msgs_GetModelState'`

Object Functions

rosmessage	Create ROS messages
call	Call the ROS service server and receive a response

Create Object

`client = robotics.ros.ServiceClient(node, name)` creates a service client that connects to a service server. The client gets its service type from the server. The service client attaches to the `robotics.ros.Node` object handle, `node`.

`client = robotics.ros.ServiceClient(node, name,'Timeout',timeout)` specifies a timeout period in seconds for the client to connect the service server.

See Also

`call` | `rosmessage` | `rossvcserver`

Introduced in R2015a

Using ServiceServer Objects

Create ROS service server

Description

Use `robotics.ros.ServiceServer` to create a ROS service server that can receive requests from, and send responses to, a ROS service client. You must create the service server before creating the service client.

When you create the service client, it establishes a connection to the server. The connection persists while both client and server exist and can reach each other. When you create the service server, it registers itself with the ROS master. To get a list of services, or to get information about a particular service that is available on the current ROS network, use the `rosservice` function.

The service has an associated message type and contains a pair of messages: one for the request and one for the response. The service server receives a request, constructs an appropriate response based on a call function, and returns it to the client. The behavior of the service server is inherently asynchronous, because it becomes active only when a service client connects to the ROS network and issues a call.

Examples

Use ROS Service Server with `ServiceServer` and `ServiceClient` Objects

Create a ROS service serve by creating a `ServiceServer` object and use `ServiceClient` objects to request information over the network. The callback function used by the server takes a string, reverses it, and returns the reversed string.

Start the ROS master and node.

```
master = robotics.ros.Core;
node = robotics.ros.Node('/test');
```

Create a service server. This server expects a string as a request and responds with a string based on the callback.

```
server = robotics.ros.ServiceServer(node,'/data/string',...
```

```
                                                'roseus/StringString');
```

Create a callback function. This function takes an input string as the Str property of req and returns it as the Str property of resp. You must create and save this function seperately. req is a ROS message you create using rosmessage.

```
% Copyright 2015 The MathWorks, Inc.

function [resp] = flipString(~,req,resp)
% FLIPSTRING Reverses the order of a string in REQ and returns it in RESP.
resp.Str = fliplr(req.Str);
end
```

Save this code as a file named flipString.m to a folder on your MATLAB® path.

Assign the callback function for incoming service calls.

```
server.NewRequestFcn = @flipString;
```

Create a service client and connect to the service server. Create a request message based on the client.

```
client = robotics.ros.ServiceClient(node,'/data/string');
request = rosmessage(client);
request.Str = 'hello world';
```

Send a service request and wait for a response. Specify that the service waits 3 seconds for a response.

```
response = call(client,request,'Timeout',3)
```

```
response = 

  ROS StringStringResponse message with properties:

    MessageType: 'roseus/StringStringResponse'
            Str: 'dlrow olleh'

  Use showdetails to show the contents of the message
```

The response is a flipped string from the request message.

Clear the service client, service server, and ROS node. Shut down the ROS master.

```
clear('client', 'server', 'node')
clear('master')
```

- "Calling and Providing ROS Services"

Properties

`ServiceName` — Name of the service

string

This property is read only.

Name of the service, specified as a string.

Example: `'/gazebo/get_model_state'`

`ServiceType` — Type of service

string

This property is read only.

Type of service, specified as a string.

Example: `'rostype.gazebo_msgs_GetModelState'`

`NewMessageFcn` — Callback property

function handle | cell array

Callback property, specified as a function handle or cell array. In the first element of the cell array, specify either a function handle or a string representing a function name. In subsequent elements, specify user data.

The subscriber callback function requires at least two input arguments. The first argument, `src`, is the associated subscriber object. The second argument, `msg`, is the received message object. The function header for the callback is:

```
function subCallback(src,msg)
```

Specify the `NewMessageFcn` property as:

```
sub.NewMessageFcn = @subCallback;
```

When setting the callback, you pass additional parameters to the callback function by including both the callback function and the parameters as elements of a cell array. The function header for the callback is:

```
function subCallback(src,msg,userData)
```

Specify the `NewMessageFcn` property as:

```
sub.NewMessageFcn = {@subCallback,userData};
```

Object Functions

rosmessage	Create ROS messages

Create Object

`server = robotics.ros.ServiceServer(node, name,type)` creates a service server that attaches to the ROS node, `node`. The server becomes available through the specified service name and type once a callback function handle is specified in `NewMessageFcn`.

`server = robotics.ros.ServiceServer(node, name,type,callback)` specifies the callback function which is set to the `NewMessageFcn` property.

See Also

`rossvcclient`

Introduced in R2015a

Using Subscriber Objects

Create a ROS subscriber

Description

The `Subscriber` object represents a subscriber on the ROS network. The `robotics.ros.Subscriber` object subscribed to an available topic or to a topic that it creates. This topic has an associated message type. Publishers can send messages over the network that the `Subscriber` object receives.

You can create a `Subscriber` object by using the `rossubscriber` function, or by calling `robotics.ros.Subscriber`:

- `rossubscriber` only works with the global node using `rosinit`. It does not require a node object handle as an argument.
- `robotics.ros.Subscriber` works with additional nodes that are created using `robotics.ros.Node`. It requires a node object handle as the first argument.

Examples

Use ROS Subscriber Object

Use a ROS `Subscriber` object to receive messages over the ROS network.

Start the ROS master and node.

```
master = robotics.ros.Core;
node = robotics.ros.Node('/test');
```

Create a publisher and subscriber to send and receive a message over the ROS network.

```
pub = robotics.ros.Publisher(node,'/chatter','std_msgs/String');
pause(1)
sub = robotics.ros.Subscriber(node,'/chatter','std_msgs/String');
```

Send a message over the network.

```
msg = rosmessage(rostype.std_msgs_String);
```

```
msg.Data = 'hello world';
send(pub,msg)
```

View the message data using the `LatestMesasge` property of the `Subscriber` object.

```
pause(1)
sub.LatestMessage
```

```
ans = 

  ROS String message with properties:

    MessageType: 'std_msgs/String'
           Data: 'hello world'

  Use showdetails to show the contents of the message
```

Clear the publisher, subscriber, and ROS node. Shut down the ROS master.

```
clear('pub','sub','node')
clear('master')
```

- “Exchanging Data with ROS Publishers and Subscribers”

Properties

`TopicName` — Name of the subscribed topic

string

This property is read only.

Name of the subscribed topic, specified as a string. If the topic does not exist, the object creates the topic using its associated message type.

Example: `'/chatter'`

Data Types: `char`

`MessageType` — Message type of subscribed messages

string

This property is read only.

Message type of subscribed messages, specified as a string. This message type remains associated with the topic.

Example: 'std_msgs/String'

Data Types: char

LatestMessage — Latest message sent to the topic

Message object

Latest message sent to the topic, specified as a Message object. The Message object is specific to the given MessageType. If the subscriber has not received a message, then the Message object is empty.

BufferSize — Buffer size

1 (default) | scalar

Buffer size of the incoming message queue, specified as the comma-separated pair consisting of 'BufferSize' and a scalar. If messages arrive faster and than your callback can process them, they are deleted once the incoming queue is full.

NewMessageFcn — Callback property

function handle | cell array

Callback property, specified as a function handle or cell array. In the first element of the cell array, specify either a function handle or a string representing a function name. In subsequent elements, specify user data.

The subscriber callback function requires at least two input arguments. The first argument, src, is the associated subscriber object. The second argument, msg, is the received message object. The function header for the callback is:

```
function subCallback(src,msg)
```

Specify the NewMessageFcn property as:

```
sub.NewMessageFcn = @subCallback;
```

When setting the callback, you pass additional parameters to the callback function by including both the callback function and the parameters as elements of a cell array. The function header for the callback is:

```
function subCallback(src,msg,userData)
```

Specify the `NewMessageFcn` property as:

```
sub.NewMessageFcn = {@subCallback,userData};
```

Object Functions

receive	Wait for new ROS message
rosmessage	Create ROS messages

Create Object

`sub = robotics.ros.Subscriber(node,topicname)` subscribes to a topic with name, `topicname`. `node` is the `robotics.ros.Node` object handle that this publisher attaches to.

`sub = robotics.ros.Subscriber(node,topicname,type)` specifies the message type, `type`, of the topic. If a topic with the same name exists with a different message type, MATLAB creates a new topic with the given message type.

`sub = robotics.ros.Subscriber(node,topicname,callback)` specifies a callback function, and optional data, to run when the subscriber object receives a topic message. See `NewMessageFcn` in “Properties” on page 2-26 for more information about the callback function.

`sub = robotics.ros.Subscriber(node,topicname,type,callback)` specifies the topic name, type and callback function for the subscriber.

`sub = robotics.ros.Subscriber( ___ ,'BufferSize',value)` specifies the queue size in `value` for incoming messages. See the `BufferSize` in “Properties” on page 2-26 for more information. You can use any combination of previous inputs with this syntax.

See Also

`receive` | `rosmessage` | `rospublisher` | `rossubscriber`

Introduced in R2015a

Using TransformationTree Objects

Receive, send, and apply ROS transformations

Description

ROS uses the tf transform library to store the relationship between multiple coordinate frames. The relative transformations between these coordinate frames are maintained in a tree structure. Querying this tree lets you transform entities like poses and points between any two coordinate frames.

Examples

Use `TransformationTree` Object

Create a ROS transformation tree. You can then view or use transformation information for different coordinate frames setup in the ROS network.

Start ROS network and broadcast sample transformation data.

```
rosinit
node = robotics.ros.Node('/testTf');
exampleHelperROSStartTfPublisher
```

```
Initializing ROS master on http://bat5236win64:11311/.
Initializing global node /matlab_global_node_99770 with NodeURI http://bat5236win64:56
Using Master URI http://localhost:11311 from the global node to connect to the ROS mas
```

Retrieve the `TransformationTree` object. Pause to wait for tftree to update.

```
tftree = robotics.ros.TransformationTree(node);
pause(1)
```

View available coordinate frames and the time when they were last received.

```
frames = tftree.AvailableFrames
updateTime = tftree.LastUpdateTime
```

```
frames =
```

```
    'camera_center'
    'mounting_point'
    'robot_base'


updateTime =

  ROS Time with properties:

     Sec: 1440450735
    Nsec: 863000000
```

Wait for the transform between two frames, `'camera_center'` and `'robot_base'`. This will wait until the transformation is valid and block all other operations. A time out of 5 seconds is also given.

```
waitForTransform(tftree,'robot_base','camera_center',5)
```

Define a point in the camer's coordinate frame.

```
pt = rosmessage('geometry_msgs/PointStamped');
pt.Header.FrameId = 'camera_center';
pt.Point.X = 3;
pt.Point.Y = 1.5;
pt.Point.Z = 0.2;
```

Tranform the point into the `'base_link'` frame.

```
tfpt = transform(tftree, 'robot_base', pt)
```

```
tfpt =

  ROS PointStamped message with properties:

    MessageType: 'geometry_msgs/PointStamped'
         Header: [1x1 Header]
          Point: [1x1 Point]

  Use showdetails to show the contents of the message
```

Display the transformed point coordinates.

```
tfpt.Point
```

```
ans = 

  ROS Point message with properties:

    MessageType: 'geometry_msgs/Point'
              X: 1.2000
              Y: 1.5000
              Z: -2.5000

  Use showdetails to show the contents of the message
```

Clear ROS node. Shut down ROS master.

```
clear('node')
rosshutdown
```

```
Shutting down global node /matlab_global_node_99770 with NodeURI http://bat5236win64:5
Shutting down ROS master on http://bat5236win64:11311/.
```

- "Accessing the tf Transformation Tree in ROS"

Properties

`AvailableFrames` — List of all available coordinate frames

cell array

This property is read only.

List of all available coordinate frames, specified as a cell array. This list of available frames updates if new transformations are received by the transformation tree object.

Example: `{'camera_center';'mounting_point';'robot_base'}`

Data Types: `cell`

`LastUpdateTime` — Time when the last transform was received

ROS `Time` object

This property is read only.

Time when the last transform was received, specified as a ROS `Time` object.

Object Functions

waitForTransform	Wait until a transformation is available
getTransform	Retrieve the transformation between two coordinate frames
transform	Transform message entities into target coordinate frame
sendTransform	Send transformation to ROS network

Create Object

`trtree = robotics.ros.TransformationTree(node)` creates a ROS transformation tree object handle that the transformation tree is attached to. `node` is the node connected to the ROS network that publishes transformations.

See Also

`getTransform` | `rostf` | `sendTransform` | `transform` | `waitForTransform`

Introduced in R2015a

angdiff

Difference between two angles

Syntax

`delta = angdiff(alpha,beta)`

`delta = angdiff(alpha)`

Description

`delta = angdiff(alpha,beta)` calculates the difference between the angles `alpha` and `beta`. This function subtracts `alpha` from `beta` with the result wrapped on the interval `[-pi,pi]`. You can specify the input angles as single values or as arrays of angles that have the same number of values.

`delta = angdiff(alpha)` returns the angular difference between adjacent elements of `alpha` along the first dimension whose size does not equal 1. The first entry is subtracted from the second, the second from the third, etc. The output, `delta`, will be a matrix of size *m-1*-by-*n* given that `alpha` is a *m*-by-*n* matrix and *m* is greater than 1 and *n* is greater than zero.

Examples

Calculate Difference Between Two Angles

```
d = angdiff(pi,2*pi)
```

```
d =

    3.1416
```

Calculate Difference Between Two Angle Arrays

```
d = angdiff([pi/2 3*pi/4 0],[pi pi/2 -pi])
```

```
d =

    1.5708   -0.7854   -3.1416
```

Calculate Angle Differences of Adjacent Elements

```
angles = [pi pi/2 pi/4 pi/2];
d = angdiff(angles)
```

```
d =

   -1.5708   -0.7854    0.7854
```

Input Arguments

`alpha` — Angle in radians

scalar | vector | matrix | multidimensional array

Angle in radians, specified as a scalar, vector, matrix, or multidimensional array. This is the angle that is subtracted from `beta` when specified.

Example: `pi/2`

`beta` — Angle in radians

scalar | vector | matrix | multidimensional array

Angle in radians, specified as a scalar, vector, matrix, or multidimensional array of the same size as `alpha`. This is the angle that `alpha` is subtracted from when specified.

Example: `pi/2`

Output Arguments

`delta` — Difference between two angles

scalar | vector | matrix | multidimensional array

Angular difference between two angles, returned as a scalar, vector, or array. `delta` is wrapped to the interval `[-pi,pi]`.

Introduced in R2015a

apply

Transform message entities into target frame

Syntax

```
tfentity = apply(tfmsg,entity)
```

Description

`tfentity = apply(tfmsg,entity)` applies the transformation represented by the `'TransformStamped'` ROS message to the input message object `entity`.

This function determines the message type of `entity` and apples the appropriate transformation method to it. If the object cannot handle a particular message type, then MATLAB displays an error message.

If you only want to use the most current transformation, call `transform` instead. If you want to store a transformation message for later use, call`getTransform` and then call `apply`.

Examples

Apply Transformation to a Point

```
tfPoint = apply(transform,point);
```

Input Arguments

`tfmsg` — Transformation message
`TransformStamped` ROS message handle

Transformation message, specified as a `TransformStamped` ROS message handle. The `tfmsg` is a ROS message of type: `geometry_msgs/TransformStamped`.

`entity` — ROS message
`Message` object handle

ROS message, specified as a `Message` object handle.

Supported messages are:

- `geometry_msgs/PointStamped`
- `geometry_msgs/PoseStamped`
- `geometry_msgs/PointCloud2Stamped`
- `geometry_msgs/QuaternionStamped`
- `geometry_msgs/Vector3Stamped`

Output Arguments

`tfentity` — Transformed ROS message
`Message` object handle

Transformed ROS message, returned as a `Message` object handle.

See Also

`getTransform` | `transform`

Introduced in R2015a

axang2quat

Convert axis-angle rotation to quaternion

Syntax

```
quat = axang2quat(axang)
```

Description

`quat = axang2quat(axang)` converts a rotation given in axis-angle form, `axang`, to quaternion, `quat`.

Examples

Convert Axis-Angle Rotation to Quaternion

```
axang = [1 0 0 pi/2];
quat = axang2quat(axang)
```

```
quat =

    0.7071    0.7071         0         0
```

Input Arguments

`axang` — Rotation given in axis-angle form

n-by-4 matrix

Rotation given in axis-angle form, specified as an n-by-4 matrix of n axis-angle rotations. The first three elements of every row specify the rotation axis, and the last element defines the rotation angle (in radians).

Example: `[1 0 0 pi/2]`

Output Arguments

`quat` — Unit quaternion
n-by-4 matrix

Unit quaternion, returned as an n-by-4 matrix containing n quaternions. Each quaternion, one per row, is of the form $q = [w\ x\ y\ z]$, with w as the scalar number.

Example: `[0.7071 0.7071 0 0]`

See Also

`quat2axang`

Introduced in R2015a

axang2rotm

Convert axis-angle rotation to rotation matrix

Syntax

```
rotm = axang2rotm(axang)
```

Description

`rotm = axang2rotm(axang)` converts a rotation given in axis-angle form, `axang`, to an orthonormal rotation matrix, `rotm`. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Examples

Convert Axis-Angle Rotation to Rotation Matrix

```
axang = [0 1 0 pi/2];
rotm = axang2rotm(axang)
```

```
rotm =

    0.0000         0    1.0000
         0    1.0000         0
   -1.0000         0    0.0000
```

Input Arguments

`axang` — Rotation given in axis-angle form

n-by-4 matrix

Rotation given in axis-angle form, specified as an *n*-by-4 matrix of *n* axis-angle rotations. The first three elements of every row specify the rotation axis, and the last element defines the rotation angle (in radians).

Example: `[1 0 0 pi/2]`

Output Arguments

`rotm` — Rotation matrix

3-by-3-by-*n* matrix

Rotation matrix, returned as a 3-by-3-by-*n* matrix containing *n* rotation matrices. Each rotation matrix has a size of 3-by-3 and is orthonormal. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Example: `[0 0 1; 0 1 0; -1 0 0]`

See Also

`rotm2axang`

Introduced in R2015a

axang2tform

Convert axis-angle rotation to homogeneous transformation

Syntax

`tform = axang2tform(axang)`

Description

`tform = axang2tform(axang)` converts a rotation given in axis-angle form, `axang`, to a homogeneous transformation matrix, `tform`. When using the transformation matrix, premultiply it with the coordinates to be transformed (as opposed to postmultiplying).

Examples

Convert Axis-Angle Rotation to Homogeneous Transformation

```
axang = [1 0 0 pi/2];
tform = axang2tform(axang)
```

```
tform =

    1.0000         0         0         0
         0    0.0000   -1.0000         0
         0    1.0000    0.0000         0
         0         0         0    1.0000
```

Input Arguments

`axang` — Rotation given in axis-angle form
n-by-4 matrix

Rotation given in axis-angle form, specified as an n-by-4 matrix of n axis-angle rotations. The first three elements of every row specify the rotation axis, and the last element defines the rotation angle (in radians).

Example: `[1 0 0 pi/2]`

Output Arguments

`tform` — Homogeneous transformation

4-by-4-by-n matrix

Homogeneous transformation matrix, specified by a 4-by-4-by-n matrix of n homogeneous transformations. When using the transformation matrix, premultiply it with the coordinates to be formed (as opposed to postmultiplying).

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

See Also

`tform2axang`

Introduced in R2015a

call

Call the ROS service server and receive a response

Syntax

```
response = call(serviceclient)
response = call(serviceclient,requestmsg)
response = call(___,Name,Value)
```

Description

`response = call(serviceclient)` sends a default service request message and waits for a service `response`. The default service request message is an empty message of type `serviceclient.ServiceType`.

`response = call(serviceclient,requestmsg)` specifies a service request message, `requestmsg`, to be sent to the service.

`response = call(___,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments, using any of the arguments from the previous syntaxes. `Name` must appear inside single quotes `('')`. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Examples

Create Service Client and Call for Response Using Default Message

```
client = rossvcclient('/gazebo/get_model_state');
response = call(client);
```

Call for Response Using Specific Request Message

```
reqmessage = rosmessage(client);
```

```
response = call(client,reqmessage);
```

Wait for Response Using Timeout of Five Seconds

```
response = call(client,reqmessage,'TimeOut',5);
```

Input Arguments

`serviceclient` — Service client
`ServiceClient` object handle

Service client, specified as a `ServiceClient` object handle.

`requestmsg` — Request message
`Message` object handle

Request message, specified as a `Message` object handle. The default message type is `serviceclient.ServiceType`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'TimeOut',5`

`'TimeOut'` — Timeout for service response in seconds
inf (default) | scalar

Timeout for service response in seconds, specified as a comma-separated pair consisting of `'Timeout'` and a scalar. If the service client does not receive a service response and the timeout period elapses, `call` displays an error message and lets MATLAB continue running the current program. The default value of `inf` blocks MATLAB from running the current program until the service client receives a service response.

Output Arguments

`response` — Response message
`Message` object handle

llResponse message sent by the service server, returned as a `Message` object handle.

See Also

`rossvcclient`

Introduced in R2015a

cart2hom

Convert Cartesian coordinates to homogeneous coordinates

Syntax

```
hom = cart2hom(cart)
```

Description

`hom = cart2hom(cart)` converts a set of points in Cartesian coordinates to homogeneous coordinates.

Examples

Convert 3-D Cartesian Points to Homogeneous Coordinates

```
c = [0.8147 0.1270 0.6324; 0.9058 0.9134 0.0975];
h = cart2hom(c)
```

```
h =

    0.8147    0.1270    0.6324    1.0000
    0.9058    0.9134    0.0975    1.0000
```

Input Arguments

`cart` — Cartesian coordinates

n-by-(k–1) matrix

Cartesian coordinates, specified as an n-by-(k–1) matrix, containing n points. Each row of `cart` represents a point in (k–1)-dimensional space. k must be greater than or equal to 2.

Example: `[0.8147 0.1270 0.6324; 0.9058 0.9134 0.0975]`

Output Arguments

`hom` — Homogeneous points
n-by-*k* matrix

Homogeneous points, returned as an *n*-by-*k* matrix, containing *n* points. *k* must be greater than or equal to 2.

Example: `[0.2785 0.9575 0.1576 0.5; 0.5469 0.9649 0.9706 0.5]`

See Also
`hom2cart`

Introduced in R2015a

definition

Retrieve definition of ROS message type

Syntax

```
def = definition(msg)
```

Description

`def = definition(msg)` returns the ROS definition of the message type associated with the message object, `msg`. The details of the message definition include the structure, property data types, and comments from the authors of that specific message.

Examples

Access ROS Message Definition for Message

Create a `Point` Message.

```
point = rosmessage('geometry_msgs/Point');
```

Access the definition.

```
def = definition(point)
```

```
def =

% This contains the position of a point in free space
double X
double Y
double Z
```

Input Arguments

`msg` — ROS message
`Message` object handle

ROS message, specified as a `Message` object handle. This message can be created using the `rosmessage` function.

Output Arguments

`def` — Details of message definition
string

Details of the information inside the ROS message definition, returned as a string.

See Also

`rosmessage` | `rosmsg`

Introduced in R2015a

del

Delete a ROS parameter

Syntax

```
del(ptree,paramname)
```

Description

`del(ptree,paramname)` deletes a parameter with name `paramname` from the parameter tree, `ptree`. The parameter is also deleted from the ROS parameter server. If the specified `paramname` does not exist, the function displays an error.

Examples

Delete Parameter on ROS Master

Create parameter tree, 'MyParam' parameter, and check existence.

```
ptree = rosparam;
set(ptree,'MyParam','test')
has(ptree,'MyParam')
```

```
ans =

     1
```

Delete parameter and check existence.

```
del(ptree,'MyParam')
has(ptree,'MyParam')
```

```
ans =
```

```
0
```

Input Arguments

`ptree` — Parameter tree
`ParameterTree` object handle

Parameter tree, specified as a `ParameterTree` object handle. Create this object using the `rosparam` function.

`paramname` — ROS parameter name
string

ROS parameter name, specified as a string. This string must match the parameter name exactly.

See Also

`has` | `rosparam` | `set`

Introduced in R2015a

eul2quat

Convert Euler angles to quaternion

Syntax

```
quat = eul2quat(eul)
quat = eul2quat(eul,sequence)
```

Description

`quat = eul2quat(eul)` converts a given set of Euler angles, `eul`, to the corresponding quaternion, `quat`. The default order for Euler angle rotations is `'ZYX'`.

`quat = eul2quat(eul,sequence)` converts a set of Euler angles into a quaternion. The Euler angles are specified in the axis rotation sequence, `sequence`. The default order for Euler angle rotations is `'ZYX'`.

Examples

Convert Euler Angles to Quaternion

```
eul = [0 pi/2 0];
qZYX = eul2quat(eul)
```

```
qZYX =

    0.7071         0    0.7071         0
```

Convert Euler Angles to Quaternion Using Default ZYZ Axis Order

```
eul = [pi/2 0 0];
qZYZ = eul2quat(eul,'ZYZ')
```

```
qZYZ =
```

```
    0.7071         0         0    0.7071
```

Input Arguments

`eul` — Euler rotation angles

n-by-3 matrix

Euler rotation angles in radians, specified as an *n*-by-3 array of Euler rotation angles. Each row represents one Euler angle set.

Example: `[0 0 1.5708]`

`sequence` — Axis rotation sequence

`'ZYX'` (default) | `'ZYZ'`

Axis rotation sequence for the Euler angles, specified as one of these strings:

- `'ZYX'` (default) – The order of rotation angles is *z*-axis, *y*-axis, *x*-axis.
- `'ZYZ'` – The order of rotation angles is *z*-axis, *y*-axis, *z*-axis.

Output Arguments

`quat` — Unit quaternion

n-by-4 matrix

Unit quaternion, returned as an *n*-by-4 matrix containing *n* quaternions. Each quaternion, one per row, is of the form $q = [w\ x\ y\ z]$, with *w* as the scalar number.

Example: `[0.7071 0.7071 0 0]`

See Also

`quat2eul`

Introduced in R2015a

eul2rotm

Convert Euler angles to rotation matrix

Syntax

```
rotm = eul2rotm(eul)
rotm = eul2rotm(eul,sequence)
```

Description

`rotm = eul2rotm(eul)` converts a set of Euler angles, `eul`, to the corresponding rotation matrix, `rotm`. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying). The default order for Euler angle rotations is `'ZYX'`.

`rotm = eul2rotm(eul,sequence)` converts Euler angles to a rotation matrix, `rotm`. The Euler angles are specified in the axis rotation sequence, `sequence`. The default order for Euler angle rotations is `'ZYX'`.

Examples

Convert Euler Angles to Rotation Matrix

```
eul = [0 pi/2 0];
rotmZYX = eul2rotm(eul)
```

```
rotmZYX =

    0.0000         0    1.0000
         0    1.0000         0
   -1.0000         0    0.0000
```

Convert Euler Angles to Rotation Matrix Using ZYZ Axis Order

```
eul = [0 pi/2 pi/2];
rotmZYZ = eul2rotm(eul,'ZYZ')
```

```
rotmZYZ =

    0.0000   -0.0000    1.0000
    1.0000    0.0000         0
   -0.0000    1.0000    0.0000
```

Input Arguments

`eul` — Euler rotation angles

n-by-3 matrix

Euler rotation angles in radians, specified as an n-by-3 array of Euler rotation angles. Each row represents one Euler angle set.

Example: `[0 0 1.5708]`

`sequence` — Axis rotation sequence

`'ZYX'` (default) | `'ZYZ'`

Axis rotation sequence for the Euler angles, specified as one of these strings:

- `'ZYX'` (default) – The order of rotation angles is z-axis, y-axis, x-axis.
- `'ZYZ'` – The order of rotation angles is z-axis, y-axis, z-axis.

Output Arguments

`rotm` — Rotation matrix

3-by-3-by-n matrix

Rotation matrix, returned as a 3-by-3-by-n matrix containing n rotation matrices. Each rotation matrix has a size of 3-by-3 and is orthonormal. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Example: `[0 0 1; 0 1 0; -1 0 0]`

See Also

`rotm2eul`

Introduced in R2015a

eul2tform

Convert Euler angles to homogeneous transformation

Syntax

```
eul = eul2tform(eul)
tform = eul2tform(eul,sequence)
```

Description

`eul = eul2tform(eul)` converts a set of Euler angles, `eul`, into a homogeneous transformation matrix, `tform`. When using the transformation matrix, premultiply it with the coordinates to be transformed (as opposed to postmultiplying). The default order for Euler angle rotations is `'ZYX'`.

`tform = eul2tform(eul,sequence)` converts Euler angles to a homogeneous transformation. The Euler angles are specified in the axis rotation sequence, `sequence`. The default order for Euler angle rotations is `'ZYX'`.

Examples

Convert Euler Angles to Homogeneous Transformation Matrix

```
eul = [0 pi/2 0];
tformZYX = eul2tform(eul)
```

```
tformZYX =

    0.0000         0    1.0000         0
         0    1.0000         0         0
   -1.0000         0    0.0000         0
         0         0         0    1.0000
```

Convert Euler Angles to Homogeneous Transformation Matrix Using ZYZ Axis Order

```
eul = [0 pi/2 pi/2];
```

```
tformZYZ = eul2tform(eul,'ZYZ')
```

```
tformZYZ =

    0.0000   -0.0000    1.0000         0
    1.0000    0.0000         0         0
   -0.0000    1.0000    0.0000         0
         0         0         0    1.0000
```

Input Arguments

`eul` — Euler rotation angles

n-by-3 matrix

Euler rotation angles in radians, specified as an *n*-by-3 array of Euler rotation angles. Each row represents one Euler angle set.

Example: `[0 0 1.5708]`

`sequence` — Axis rotation sequence

`'ZYX'` (default) | `'ZYZ'`

Axis rotation sequence for the Euler angles, specified as one of these strings:

- `'ZYX'` (default) – The order of rotation angles is *z*-axis, *y*-axis, *x*-axis.
- `'ZYZ'` – The order of rotation angles is *z*-axis, *y*-axis, *z*-axis.

Output Arguments

`tform` — Homogeneous transformation

4-by-4-by-*n* matrix

Homogeneous transformation matrix, specified by a 4-by-4-by-*n* matrix of *n* homogeneous transformations. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

See Also

`tform2eul`

Introduced in R2015a

get

Get ROS parameter value

Syntax

```
pvalue = get(ptree,paramname)
```

Description

`pvalue = get(ptree,paramname)` gets the value of the parameter with the name `paramname` from the parameter tree object `ptree`.

Examples

Set and Get Parameter Value

Create the parameter tree.

```
ptree = rosparam;
```

Set the parameter value.

```
set(ptree,'DoubleParam',1.0)
```

Get the parameter value.

```
get(ptree,'DoubleParam')
```

```
ans =

     1
```

Input Arguments

`ptree` — Parameter tree
`ParameterTree` object handle

Parameter tree, specified as a `ParameterTree` object handle. Create this object using the `rosparam` function.

`paramname` — ROS parameter name
string

ROS parameter name, specified as a string. This string must match the parameter name exactly.

Output Arguments

`pvalue` — Parameter value
int32 | logical | char | double | cell array

Parameter value, returned as either a `int32`, `logical`, `double`, `char`, or `cell array`. `pvalue` matches the value of the specified`paramname` and the supported data type in `ParameterTree`. Currently, Base64–encoded binary data and iso8601 data from ROS are not supported.

See Also
`rosparam` | `set`

Introduced in R2015a

getTransform

Retrieve the transformation between two coordinate frames

Syntax

```
tf = getTransform(tftree,targetframe,sourceframe)
```

Description

`tf = getTransform(tftree,targetframe,sourceframe)` returns the latest known transformation between two coordinate frames. Transformations are structured as a 3-D translation (3-element vector) and a 3-D rotation (quaternion).

Examples

Get Transformation

```
tf = getTransform(tftree,'/camera_depth_frame','/base_link');
```

Input Arguments

`tftree` — ROS transformation tree

`TransformationTree` object handle

ROS transformation tree, specified as a `TransformationTree` object handle. You can create a transformation tree by calling the `rostf` function.

`targetframe` — Target coordinate frame

string

Target coordinate frame, specified as a string. You can view the available frames for transformation by calling `tftree.AvailableFrames`.

`sourceframe` — Initial coordinate frame

string

Initial coordinate frame, specified as a string. You can view the available frames for transformation by calling `tftree.AvailableFrames`.

Output Arguments

`tf` — Transformation between coordinate frames
`TransformStamped` object handle

Transformation between coordinate frames, returned as a `TransformStamped` object handle. Transformations are structured as a 3-D translation (3-element vector) and a 3-D rotation (quaternion).

See Also

`transform` | `waitForTransform`

Introduced in R2015a

has

Check if ROS parameter name exists

Syntax

```
exists = has(ptree,paramname)
```

Description

`exists = has(ptree,paramname)` checks if the parameter with name `paramname` exists in the parameter tree, `ptree`.

Examples

Check If ROS Parameter Exists

Create a parameter tree and check for the 'MyParam' parameter.

```
ptree = rosparam;
```

```
has(ptree,'MyParam')
```

```
ans =

     0
```

Create a 'MyParam' parameter and verify that it exists.

```
set(ptree,'MyParam','test')
```

```
has(ptree,'MyParam')
```

```
ans =
```

```
      1
```

Input Arguments

`ptree` — Parameter tree
`ParameterTree` object handle

Parameter tree, specified as a `ParameterTree` object handle. Create this object using the `rosparam` function.

`paramname` — ROS parameter name
string

ROS parameter name, specified as a string. This string must match the parameter name exactly.

Output Arguments

`exists` — Flag indicating whether the parameter exists
true | false

Flag indicating whether the parameter exists, returned as `true` or `false`.

See Also

`get` | `rosparam` | `search` | `set`

Introduced in R2015a

hom2cart

Convert homogeneous coordinates to Cartesian coordinates

Syntax

```
cart = hom2cart(hom)
```

Description

`cart = hom2cart(hom)` converts a set of homogeneous points to Cartesian coordinates.

Examples

Convert Homogeneous Points to 3-D Cartesian Points

```
h = [0.2785 0.9575 0.1576 0.5; 0.5469 0.9649 0.9706 0.5];
c = hom2cart(h)
```

```
c =

    0.5570    1.9150    0.3152
    1.0938    1.9298    1.9412
```

Input Arguments

`hom` — Homogeneous points

n-by-*k* matrix

Homogeneous points, specified as an *n*-by-*k* matrix, containing *n* points. *k* must be greater than or equal to 2.

Example: `[0.2785 0.9575 0.1576 0.5; 0.5469 0.9649 0.9706 0.5]`

Output Arguments

`cart` — Cartesian coordinates

n-by-(*k*–1) matrix

Cartesian coordinates, returned as an *n*-by-(*k*–1) matrix, containing *n* points. Each row of `cart` represents a point in (*k*–1)-dimensional space. *k* must be greater than or equal to 2.

Example: `[0.8147 0.1270 0.6324; 0.9058 0.9134 0.0975]`

See Also

`cart2hom`

Introduced in R2015a

plot

Display ROS laser scan messages on custom plot

Syntax

```
plot(scan)
plot(scan,Name,Value)
linehandle = plot( ___ )
```

Description

`plot(scan)` creates a line plot of the laser scan in *xy*-coordinates that is based on the input `LaserScan` object message. Axes are automatically scaled to the maximum range that the laser scanner supports.

`plot(scan,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments. Name must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`linehandle = plot( ___ )` returns a column vector of line series handles, using any of the arguments from previous syntaxes. Use `linehandle` to modify properties of the line series after it is created.

When plotting ROS laser scan messages, MATLAB follows the standard ROS convention for axis orientation. This convention states that **positive *x* is forward, positive *y* is left, and positive *z* is up**. For more information, see Axis Orientation on the ROS Wiki.

Examples

Plot Laser Scan

```
plot(scan);
```

Plot Laser Scan with Maximum Range Specified

```
plot(scan,'MaximumRange',10);
```

Save Line Handle for Laser Scan Plot

```
linehandle = plot(scan);
```

Input Arguments

`scan` — Laser scan message

`LaserScan` object handle

`'sensor_msgs/LaserScan'` ROS message, specified as a `LaserScan` object handle.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MaximumRange',5`

`'Parent'` — Parent of axes

axes object

Parent of axes, specified as the comma-separated pair consisting of `'Parent'` and an axes object in which the laser scan is drawn. By default, the laser scan is plotted in the currently active axes.

`'MaximumRange'` — Range of laser scan

`scan.RangeMax` (default) | scalar

Range of laser scan, specified as the comma-sepearated pair consisting of `'MaximumRange'` and a scalar. When you specify this name-value pair argument, the minimum and maximum *x*-axis limits and the maximum *y*-axis limit are set based on specified value. The minimum *y*-axis limit is automatically determined by the opening angle of the laser scanner.

Outputs

`linehandle` — One or more chart line objects

scalar | vector

One or more chart line objects, returned as a scalar or a vector. These are unique identifiers, which you can use to query and modify properties of a specific chart line.

See Also

`readCartesian`

Introduced in R2015a

quat2axang

Convert quaternion to axis-angle rotation

Syntax

```
axang = quat2axang(quat)
```

Description

`axang = quat2axang(quat)` converts a quaternion, `quat`, to the equivalent axis-angle rotation, `axang`.

Examples

Convert Quaternion to Axis-Angle Rotation

```
quat = [0.7071 0.7071 0 0];
axang = quat2axang(quat)
```

```
axang =

    1.0000         0         0    1.5708
```

Input Arguments

`quat` — Unit quaternion
n-by-4 matrix

Unit quaternion, specified as an *n*-by-4 matrix containing *n* quaternions. Each quaternion, one per row, is of the form $q = [w\ x\ y\ z]$, with w as the scalar number.

Example: `[0.7071 0.7071 0 0]`

Output Arguments

`axang` — Rotation given in axis-angle form

n-by-4 matrix

Rotation given in axis-angle form, returned as an *n*-by-4 matrix of *n* axis-angle rotations. The first three elements of every row specify the rotation axis, and the last element defines the rotation angle (in radians).

Example: `[1 0 0 pi/2]`

See Also

`axang2quat`

Introduced in R2015a

quat2eul

Convert quaternion to Euler angles

Syntax

```
eul = quat2eul(quat)
eul = quat2eul(quat,sequence)
```

Description

`eul = quat2eul(quat)` converts a quaternion rotation, `quat`, to the corresponding Euler angles, `eul`. The default order for Euler angle rotations is `'ZYX'`.

`eul = quat2eul(quat,sequence)` converts a quaternion into Euler angles. The Euler angles are specified in the axis rotation sequence, `sequence`. The default order for Euler angle rotations is `'ZYX'`.

Examples

Convert Quaternion to Euler Angles

```
quat = [0.7071 0.7071 0 0];
eulZYX = quat2eul(quat)
```

```
eulZYX =

         0         0    1.5708
```

Convert Quaternion to Euler Angles Using ZYZ Axis Order

```
quat = [0.7071 0.7071 0 0];
eulZYZ = quat2eul(quat,'ZYZ')
```

```
eulZYZ =
```

```
1.5708   -1.5708   -1.5708
```

Input Arguments

`quat` — Unit quaternion

n-by-4 matrix

Unit quaternion, specified as an *n*-by-4 matrix containing *n* quaternions. Each quaternion, one per row, is of the form $q = [w\ x\ y\ z]$, with *w* as the scalar number.

Example: `[0.7071 0.7071 0 0]`

`sequence` — Axis rotation sequence

`'ZYX'` (default) | `'ZYZ'`

Axis rotation sequence for the Euler angles, specified as one of these strings:

- `'ZYX'` (default) – The order of rotation angles is *z*-axis, *y*-axis, *x*-axis.
- `'ZYZ'` – The order of rotation angles is *z*-axis, *y*-axis, *z*-axis.

Output Arguments

`eul` — Euler rotation angles

n-by-3 matrix

Euler rotation angles in radians, returned as an *n*-by-3 array of Euler rotation angles. Each row represents one Euler angle set.

Example: `[0 0 1.5708]`

See Also

`eul2quat`

Introduced in R2015a

quat2rotm

Convert quaternion to rotation matrix

Syntax

```
rotm = quat2rotm(quat)
```

Description

`rotm = quat2rotm(quat)` converts a quaternion `quat` to an orthonormal rotation matrix, `rotm`. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Examples

Convert Quaternion to Rotation Matrix

```
quat = [0.7071 0.7071 0 0];
rotm = quat2rotm(quat)
```

```
rotm =

    1.0000         0         0
         0   -0.0000   -1.0000
         0    1.0000   -0.0000
```

Input Arguments

`quat` — Unit quaternion

n-by-4 matrix

Unit quaternion, specified as an n-by-4 matrix containing n quaternions. Each quaternion, one per row, is of the form $q = [w\ x\ y\ z]$, with w as the scalar number.

Example: `[0.7071 0.7071 0 0]`

Output Arguments

`rotm` — Rotation matrix
3-by-3-by-*n* matrix

Rotation matrix, returned as a 3-by-3-by-*n* matrix containing *n* rotation matrices. Each rotation matrix has a size of 3-by-3 and is orthonormal. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Example: `[0 0 1; 0 1 0; -1 0 0]`

See Also
`rotm2quat`

Introduced in R2015a

quat2tform

Convert quaternion to homogeneous transformation

Syntax

```
tform = quat2tform(quat)
```

Description

`tform = quat2tform(quat)` converts a quaternion, `quat`, to a homogeneous transformation matrix, `tform`. When using the transformation matrix, premultiply it with the coordinates to be transformed (as opposed to postmultiplying).

Examples

Convert Quaternion to Homogeneous Transformation

```
quat = [0.7071 0.7071 0 0];
tform = quat2tform(quat)
```

```
tform =

    1.0000         0         0         0
         0   -0.0000   -1.0000         0
         0    1.0000   -0.0000         0
         0         0         0    1.0000
```

Input Arguments

`quat` — Unit quaternion

n-by-4 matrix

Unit quaternion, specified as an n-by-4 matrix containing n quaternions. Each quaternion, one per row, is of the form $q = [w\ x\ y\ z]$, with w as the scalar number.

Example: [0.7071 0.7071 0 0]

Output Arguments

tform — Homogeneous transformation
4-by-4-by-*n* matrix

Homogeneous transformation matrix, returned as a 4-by-4-by-*n* matrix of *n* homogeneous transformations. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Example: [0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]

See Also

tform2quat

Introduced in R2015a

readAllFieldNames

Get all available field names from ROS point cloud

Syntax

```
fieldnames = readAllFieldNames(pcloud)
```

Description

`fieldnames = readAllFieldNames(pcloud)` gets the names of all point fields that are stored in the `PointCloud2` object message, `pcloud`, and returns them in `fieldnames`.

Examples

Read All Fields from Point Cloud Message

```
fieldnames = readAllFieldNames(pcloud);
```

Input Arguments

`pcloud` — Point cloud

`PointCloud2` object handle

Point cloud, specified as a `PointCloud2` object handle for a `'sensor_msgs/PointCloud2'` ROS message.

Output Arguments

`fieldnames` — List of field names in `PointCloud2` object

cell array of strings

List of field names in `PointCloud2` object, returned as a cell array of strings. If no fields exist in the object, `fieldname` returns an empty cell array.

See Also

`readField`

Introduced in R2015a

readBinaryOccupancyGrid

Read binary occupancy grid

Syntax

```
map = readBinaryOccupancyGrid(msg)
map = readBinaryOccupancyGrid(msg,thresh)
map = readBinaryOccupancyGrid(msg,thresh,val)
```

Description

`map = readBinaryOccupancyGrid(msg)` returns a robotics.BinaryOccupancyGrid object by reading the data inside a ROS message, `msg`, which must be a `'nav_msgs/OccupancyGrid'` message. All message data values greater than or equal to the occupancy threshold are set to occupied, `1`, in the map. All other values, including unknown values (`-1`) are set to unoccupied, `0`, in the map.

`map = readBinaryOccupancyGrid(msg,thresh)` specifies a threshold, `thresh`, for occupied values. All values greater than or equal to the threshold are set to occupied, `1`. All other values are set to unoccupied, `0`.

`map = readBinaryOccupancyGrid(msg,thresh,val)` specifies a value to set for unknown values (`-1`). By default, all unknown values are set to unoccupied, `0`.

Examples

Read Data from Message

Create a occupancy grid message and populate it with data.

```
msg = rosmessage('nav_msgs/OccupancyGrid');
msg.Info.Height = 10;
msg.Info.Width = 10;
msg.Info.Resolution = 0.1;
msg.Data = 100*rand(100,1);
```

Read data from message

```
map = readBinaryOccupancyGrid(msg);
```

Read Message Data with Threshold

Threshold for occupied values is set to 65 and greater.

```
map = readBinaryOccupancyGrid(msg,65);
```

Read Message Data with Threshold and Unknown Value Replacement

```
map = readBinaryOccupancyGrid(msg,65,1);
```

Input Arguments

`msg` — `'nav_msgs/OccupancyGrid'` ROS message
`OccupancyGrid` object handle

`'nav_msgs/OccupancyGrid'` ROS message, specified as a `OccupancyGrid` object handle.

`thresh` — Threshold for occupied values
50 (default) | scalar

Threshold for occupied values, specified as a scalar. Any value greater than or equal to the threshold is set to occupied, `1`. All other values are set to unoccupied, `0`.

Data Types: `double`

`val` — Value to replace unknown values
0 (default) | 1

Value to replace unknown values, specified as either `0` or `1`. Unknown message values (`-1`) are set to the given value.

Data Types: `double` | `logical`

Output Arguments

`map` — Binary occupancy grid
`BinaryOccupancyGrid` object handle

Binary occupancy grid, returned as a `BinaryOccupancyGrid` object handle. `map` is converted from a `'nav_msgs/OccupancyGrid'` message on the ROS network. It is an object with a grid of binary values, where `1` indicates an occupied location and `0` indications an unoccupied location.

See Also

robotics.BinaryOccupancyGrid | `writeBinaryOccupancyGrid`

Introduced in R2015a

readCartesian

Read laser scan ranges in Cartesian coordinates

Syntax

```
cart = readCartesian(scan)
cart = readCartesian(___,Name,Value)
[angles,cart] = readCartesian(___)
```

Description

`cart = readCartesian(scan)` converts the polar measurements of the laser scan object, `scan`, into Cartesian coordinates, `cart`. This function uses the metadata in the message, such as angular resolution and opening angle of the laser scanner, to perform the conversion. Invalid range readings, usually represented as `NaN`, are ignored in this conversion.

`cart = readCartesian(___,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments. Name must appear inside single quotes `('')`. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`[angles,cart] = readCartesian(___)` returns the scan angles, `angles` that are associated with each Cartesian coordinate. Angles are measured counter-clockerwise around the positive *z*-axis, with the zero angle along the *x*-axis. `angles` is returned in radians and wrapped to the [–`pi`, `pi`] interval.

Examples

Read Laser Scan and Convert to Cartesian Coordinates

```
cart = readCartesian(scan);
```

Read Laser Scan and Specify Scan Range

```
cart = readCartesian(scan,'RangeLimit',[0 10]);
```

Input Arguments

`scan` — Laser scan message
`LaserScan` object handle

`'sensor_msgs/LaserScan'` ROS message, specified as a `LaserScan` object handle.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'RangeLimits',[-2 2]`

`'RangeLimits'` — Minimum and maximum range for scan in meters
`[scan.RangeMin scan.RangeMax]` (default) | 2-element `[min max]` vector

Minimum and maximum range for scan in meters, specified as a 2-element `[min max]` vector. All ranges smaller than `min` or larger than `max` are ignored during the conversion to Cartesian coordinates.

Output Arguments

`cart` — Cartesian coordinates of laser scan
n–by–2 matrix in meters

Cortesian coordinates of laser scan, returned as an *n*-by-2 matrix in meters.

`angles` — Scan angles for laser scan data
n–by–1 matrix in radians

Scan angles for laser scan data, returned as an *n*-by-1 matrix in radians. Angles are measured counter-clockerwise around the positive *z*-axis, with the zero angle along the *x*-axis. `angles` is returned in radians and wrapped to the [–`pi`, `pi`] interval.

See Also

`plot` | `readScanAngles`

Introduced in R2015a

readField

Read point cloud data based on field name

Syntax

```
fielddata = readField(pcloud,fieldname)
```

Description

`fielddata = readField(pcloud,fieldname)` reads the point field from the point cloud, `pcloud`, specified by `fieldname` and returns it in `fielddata`. If `fieldname` does not exist, the function displays an error. To preserve the structure of the point cloud data, see "Preserving Point Cloud Structure" on page 2-89.

Examples

Read *x* Coordinates for All Points

```
x = readField(pcloud,'x');
```

Input Arguments

`pcloud` — Point cloud
`PointCloud2` object handle

Point cloud, specified as a `PointCloud2` object handle for a `'sensor_msgs/PointCloud2'` ROS message.

`fieldname` — Field name of point cloud data
string

Field name of point cloud data, specified as a string. This string must match the field name exactly. If `fieldname` does not exist, the function displays an error.

Output Arguments

`fielddata` — List of field values from point cloud
matrix

List of field values from point cloud, returned as a matrix. Each row of is a point cloud reading, where *n* is the number of points and *c* is the number of values for each point. If the point cloud object being read has the `PreserveStructureOnRead` property set to true, the points are returned as an *h*-by-*w*-by-*c* matrix. For more information, see “Preserving Point Cloud Structure” on page 2-89.

More About

Tips

Point cloud data can be organized in either 1-D lists or in 2-D image styles. 2-D image styles usually come from depth sensors or stereo cameras. The input `PointCloud2` object contains a `PreserveStructureOnRead` property that is either `true` or `false` (default). Suppose you set the property to `true`.

```
pcloud.PreserveStructureOnRead = true;
```

Now calling any read functions (`readXYZ`,`readRGB`, or `readField`) preserves the organizational structure of the point cloud. When you preserve the structure, the output matrices are of size *m*-by-*n*-by-*d*, where *m* is the height, *n* is the width, and *d* is the number of return values for each point. Otherwise, all points are returned as a *x*-by-*d* list. This structure can only be preserved if the point cloud is organized.

See Also

`readAllFieldNames`

Introduced in R2015a

readImage

Convert ROS image data into MATLAB image

Syntax

```
img = readImage(msg)
[img,alpha] = readImage(msg)
```

Description

`img = readImage(msg)` converts the raw image data in the message object, `msg`, into an image matrix, `img`. You can call `readImage` using either `'sensor_msgs/Image'` or `'sensor_msgs/CompressedImage'` messages.

ROS image message data is stored in a format that is not compatible with further image processing in MATLAB. Based on the specified encoding, this function converts the data into an appropriate MATLAB image and returns it in `img`.

`[img,alpha] = readImage(msg)` returns the alpha channel of the image in `alpha`. If the image does not have an alpha channel, then `alpha` is empty.

Examples

Read ROS Image Data

```
[img,alpha] = readImage(obj);
```

Input Arguments

`msg` — ROS image message

`Image` object handle | `CompressedImage` object handle

`'sensor_msgs/Image'` or `'sensor_msgs/CompressedImage'` ROS image message, specified as an `Image` or `Compressed Image` object handle.

Output Arguments

`img` — Image

grayscale image matrix | RBG image matrix | *m*-by-*n*-by-3 array

Image, returned as a matrix representing a grayscale or RGB image or as a*m*-by-*n*-by-3 array, depending on the sensor image.

`alpha` — Alpha channel

`uint8` grayscale image

Alpha channel, returned as a `uint8` grayscale image. If no alpha channel exists, `alpha` is empty.

More About

Tips

ROS image messages can have different encodings. The encodings supported for images are different for `'sensor_msgs/Image'` and `'sensor_msgs/CompressedImage'` message types. Less compressed images are supported. The following encodings for raw images of size *M*x*N* are supported using the `'sensor_msgs/Image'` message type (**`'sensor_msgs/CompressedImage'` support is in bold**):

- **`rgb8, rgba8, bgr8, bgra8`**: `img` is an `rgb` image of size *M*x*N*x3. The alpha channel is returned in `alpha`. Each value in the outputs is represented as a `uint8`.
- `rgb16, rgba16, bgr16, bgra16`: `img` is an RGB image of size *M*x*N*x3. The alpha channel is returned in `alpha`. Each value in the outputs is represented as a `uint16`.
- **`mono8`** images are returned as grayscale images of size *M*x*N*x1. Each pixel value is represented as a `uint8`.
- `mono16` images are returned as grayscale images of size *M*x*N*x1. Each pixel value is represented as a `uint16`.
- `32fcX` images are returned as floating-point images of size *M*x*N*x*D*, where *D* is 1, 2, 3, or 4. Each pixel value is represented as a `single`.
- `64fcX` images are returned as floating-point images of size *M*x*N*x*D*, where *D* is 1, 2, 3, or 4. Each pixel value is represented as a `double`.
- `8ucX` images are returned as matrices of size *M*x*N*x*D*, where *D* is 1, 2, 3, or 4. Each pixel value is represented as a `uint8`.

- `8scX` images are returned as matrices of size *M*x*N*x*D*, where *D* is 1, 2, 3, or 4. Each pixel value is represented as a `int8`.
- `16ucX` images are returned as matrices of size *M*x*N*x*D*, where *D* is 1, 2, 3, or 4. Each pixel value is represented as a `int16`.
- `16scX` images are returned as matrices of size *M*x*N*x*D*, where *D* is 1, 2, 3, or 4. Each pixel value is represented as a `int16`.
- `32scX` images are returned as matrices of size *M*x*N*x*D*, where *D* is 1, 2, 3, or 4. Each pixel value is represented as a `int32`.
- `bayer_X` images are returned as either Bayer matrices of size *M*x*N*x1, or as a converted image of size *M*x*N*x3 (Image Processing Toolbox™ is required).

The following encoding for raw images of size *M*x*N* is supported using the **`'sensor_msgs/CompressedImage'`** message type:

- `rgb8, rgba8, bgr8, bgra8`: `img` is an `rgb` image of size *M*x*N*x3. The alpha channel is returned in `alpha`. Each output value is represented as a `uint8`.

See Also

`writeImage`

Introduced in R2015a

readMessages

Read messages from rosbag

Syntax

```
msgs = readMessages(bag)
msgs = readMessages(bag,rows)
```

Description

`msgs = readMessages(bag)` returns data from all of the messages in the `BagSelection` object, `bag`. The messages are returned in a cell array of messages.

`msgs = readMessages(bag,rows)` returns data from messages in the rows specified by `rows`. The maximum range of the rows is `[1,bag.NumMessages]`.

Examples

Return All Messages as a Cell Array

```
allMsgs = readMessages(bagMsgs);
```

Return First Ten Messages

```
firstMsgs = readMessages(bagMsgs,1:10);
```

Input Arguments

`bag` — Message of a rosbag

`BagSelection` object

All the messages contained within a rosbag, specified as a `BagSelection` object.

`rows` — Rows of `BagSelection` object

n-by-2 matrix

Rows of `BagSelection` object, specified as an *n*-by-2 matrix, where *n* is the number of rows to retrieve messages from. The maximum range of the rows is [1, `bag.NumMessage`].

Output Arguments

`msgs` — ROS message object handle
handle | cell array

ROS message object handle, returned as a handle or cell array. ROS messages are retrieved from the `BagSelection` object.

See Also

`rosbag` | `select` | `timeseries`

Introduced in R2015a

readRGB

Extract RGB values from point cloud data

Syntax

```
rgb = readXYZ(pcloud)
```

Description

`rgb = readXYZ(pcloud)` extracts the `[r g b]` values from all points in the point cloud object, `pcloud` and returns them as an n-by-3 of n 3-D point coordinates. If the point cloud does not contain the RGB field, this function will display an error. To preserve the structure of the point cloud data, see "Preserving Point Cloud Structure" on page 2-96.

Examples

Read RGB Values from Point Cloud Object

```
rgb = readRGB(pcloud);
```

Input Arguments

`pcloud` — Point cloud
`PointCloud2` object handle

Point cloud, specified as a `PointCloud2` object handle for a `'sensor_msgs/PointCloud2'` ROS message.

Output Arguments

`rgb` — List of RGBv alues from point cloud
matrix

List of RGB values from point cloud, returned as a matrix. By default, this is a *n*-by-3 matrix. If the point cloud object being read has the `PreserveStructureOnRead` property set to true, the points are returned as an *h*-by-*w*-by-3 matrix. For more information, see "Preserving Point Cloud Structure" on page 2-96.

More About

Tips

Point cloud data can be organized in either 1-D lists or in 2-D image styles. 2-D image styles usually come from depth sensors or stereo cameras. The input `PointCloud2` object contains a `PreserveStructureOnRead` property that is either `true` or `false` (default). Suppose you set the property to `true`.

```
pcloud.PreserveStructureOnRead = true;
```

Now calling any read functions (`readXYZ`,`readRGB`, or `readField`) preserves the organizational structure of the point cloud. When you preserve the structure, the output matrices are of size *m*-by-*n*-by-*d*, where *m* is the height, *n* is the width, and *d* is the number of return values for each point. Otherwise, all points are returned as a *x*-by-*d* list. This structure can only be preserved if the point cloud is organized.

See Also

`readField` | `readXYZ`

Introduced in R2015a

readScanAngles

Return scan angles for laser scan range readings

Syntax

```
angles = readScanAngles(scan)
```

Description

`angles = readScanAngles(scan)` calculates the scan angles, `angles`, corresponding to the range readings in the laser scan message, `scan`. Angles are measured counter-clockerwise around the positive *z*-axis, with the zero angle along the *x*-axis. `angles` is returned in radians and wrapped to the [–`pi`, `pi`] interval.

Examples

Return Laser Scan Angles from Range Data

```
angles = readScanAngles(scan);
```

Input Arguments

`scan` — Laser scan message
`LaserScan` object handle

`'sensor_msgs/LaserScan'` ROS message, specified as a `LaserScan` object handle.

Output Arguments

`angles` — Scan angles for laser scan data
n–by–1 matrix in radians

Scan angles for laser scan data, returned as an n-by-1 matrix in radians. Angles are measured counter-clockerwise around the positive z-axis, with the zero angle along the x-axis. `angles` is returned in radians and wrapped to the [–`pi`, `pi`] interval.

See Also

`plot` | `readCartesian`

Introduced in R2015a

readXYZ

Extract XYZ coordinates from point cloud data

Syntax

```
xyz = readXYZ(pcloud)
```

Description

`xyz = readXYZ(pcloud)` extracts the `[x y z]` coordinates from all points in the point cloud object, `pcloud`, and returns them as an *n*-by-3 matrix of *n* 3-D point coordinates. If the point cloud does not contain the *x*, *y*, and *z* fields, this function returns an error. Points that contain `NaN` are preserved in the output. To preserve the structure of the point cloud data, see “Preserving Point Cloud Structure” on page 2-100.

Examples

Read XYZ Coordinates from Point Cloud

```
xyz = readXYZ(pcloud);
```

Input Arguments

`pcloud` — Point cloud

`PointCloud2` object handle

Point cloud, specified as a `PointCloud2` object handle for a `'sensor_msgs/PointCloud2'` ROS message.

Output Arguments

`xyz` — List of XYZ values from point cloud

matrix

List of XYZ values from point cloud, returned as a matrix. By default, this is a *n*-by-3 matrix. If the point cloud object being read has the `PreserveStructureOnRead` property set to true, the points are returned as an *h*-by-*w*-by-3 matrix. For more information, see "Preserving Point Cloud Structure" on page 2-100.

More About

Tips

Point cloud data can be organized in either 1-D lists or in 2-D image styles. 2-D image styles usually come from depth sensors or stereo cameras. The input `PointCloud2` object contains a `PreserveStructureOnRead` property that is either `true` or `false` (default). Suppose you set the property to `true`.

```
pcloud.PreserveStructureOnRead = true;
```

Now calling any read functions (`readXYZ`,`readRGB`, or `readField`) preserves the organizational structure of the point cloud. When you preserve the structure, the output matrices are of size *m*-by-*n*-by-*d*, where *m* is the height, *n* is the width, and *d* is the number of return values for each point. Otherwise, all points are returned as a *x*-by-*d* list. This structure can only be preserved if the point cloud is organized.

See Also

`readField` | `readRGB`

Introduced in R2015a

receive

Wait for new ROS message

Syntax

```
msg = receive(sub)
msg = receive(sub,timeout)
```

Description

`msg = receive(sub)` waits for MATLAB to receive a topic message from the specified subscriber, `sub`, and returns it as `msg`.

`msg = receive(sub,timeout)` specifies in `timeout` the number of seconds to wait for a message. If a message is not received within the timeout limit, the software throws an error.

Examples

Create Subscriber and Receive Data

```
laser = rossubscriber('/scan', rostype.sensor_msgs_LaserScan);
scan = receive(laser);
```

Receive Data with a Two Second Timeout

```
scan = receive(sub,2);
```

Input Arguments

`sub` — ROS subscriber

`Subscriber` object handle

ROS subscriber, specified as a `Subscriber` object handle. You can create the subscriber using `rossubscriber`.

`timeout` — Timeout for receiving a message
scalar in seconds

Timeout for receiving a message, specified as a scalar in seconds.

Output Arguments

`msg` — ROS message
`Message` object handle

ROS message, returned as a `Message` object handle.

See Also

`rosmessage` | `rossubscriber` | `rostopic`

Introduced in R2015a

roboticsSupportPackages

Download and install support packages for Robotics System Toolbox

Syntax

```
roboticsSupportPackages
```

Description

`roboticsSupportPackages` opens the Support Package Installer to download and install support packages for Robotics System Toolbox™. For more details, see “Install Robotics System Toolbox Support Packages”

Examples

Open Robotics System Toolbox Support Package Installer

```
roboticsSupportPackages
```

Introduced in R2015a

rosbag

Open and parse rosbag log file

Syntax

```
bag = rosbag(filename)
```

Description

`bag = rosbag(filename)` creates an indexable `BagSelection` object, `bag`, that contains all the message indexes from the rosbag located at path `filename`. To access the data, you can call `readMessages` or `timeseries` to extract relevant data.

A rosbag, or bag, is a file format for storing ROS message data. They are used primarily to log messages within the ROS network. You can use these bags for offline analysis, visualization, and storage.

This function supports version 2.0 of the rosbag file format. It also supports only uncompressed rosbags. See the ROS Wiki page for more information about rosbags and Bag version 2.0.

Examples

Retrieve Information from rosbag

Set the path to a rosbag file.

```
filePath = 'path/to/logfile.bag';
```

Retrieve information from the rosbag.

```
bagselect = rosbag(filePath)
```

Select a subset of the messages, filtered by time and topic

```
bagselect2 = select(bagselect, 'Time', ...
```

```
[bagselect.StartTime bagselect.StartTime + 1], 'Topic', '/odom')
```

Input Arguments

`filename` — Name of rosbag file and its path

string

Name of file and its path, for the rosbag you want to access, specified as a string. This path can be relative or absolute.

Output Arguments

`bag` — Selection of rosbag messages

`BagSelection` object handle

Selection of rosbag messages, returned as a `BagSelection` object handle.

See Also

`readMessages` | `select` | `timeseries`

Introduced in R2015a

rosgenmsg

Generate custom messages from ROS definitions

Syntax

`rosgenmsg(folderpath)`

Description

`rosgenmsg(folderpath)` generates ROS custom messages in MATLAB by reading ROS custom message and service definitions in the specified folder path. The function expects ROS package folders inside the folder path. These packages contain the message definitions in `.msg` files and the service definitions in `.srv` files. Also, the packages require a `package.xml` file to define its contents.

After calling this function, you can send and receive your custom messages in MATLAB like all other supported messages. You can create these messages using `rosmessage` or view the list of messages by calling `rosmsg list`.

Examples

Generate MATLAB Code for ROS Custom Messages

```
folderpath = 'C:/Users/user1/Documents/robot_custom_msg/';
rosgenmsg(folderpath)
```

- “Create Custom Messages from ROS Package”

Input Arguments

`folderpath` — Path to ROS package folders

string

Path to package folders, specified as a string. These folders contain message definitions in `.msg` files and the service definitions in `.srv` files. Also, the packages require a `package.xml` file to define its contents.

More About

- “ROS Custom Message Support”
- ROS Tutorials: Defining Custom Messages
- ROS Tutorials: Creating a ROS msg and srv

See Also

`roboticsSupportPackages`

Introduced in R2015a

rosinit

Connect to ROS network

Syntax

```
rosinit
rosinit(hostname)
rosinit(hostname,port)
rosinit(URI)
rosinit(___,Name,Value)
```

Description

`rosinit` starts the global ROS node with a default MATLAB name and tries to connect to a ROS master running on `localhost` and port `11311`. If the global ROS node cannot connect to the ROS master, `rosinit` also starts a ROS core in MATLAB, which consists of a ROS master, a ROS parameter server, and a rosout logging node.

`rosinit(hostname)` tries to connect to the ROS master at the host name or IP address specified by `hostname`. This syntax uses `11311` as the default port number.

`rosinit(hostname,port)` tries to connect to the host name or IP address specified by `hostname` and the port number specified by `port`.

`rosinit(URI)` tries to connect to the ROS master at the given resource identifier, `URI`, for example, `'http://192.168.1.1:11311'`.

`rosinit(___,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments. `Name` must appear inside single quotes (`' '`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Using `rosinit` is a prerequisite for most ROS-related tasks in MATLAB because:

- Communicating with a ROS network requires a ROS node connected to a ROS master.

- By default, ROS functions in MATLAB operate on the global ROS node, or they operate on objects that depend on the global ROS node.

For example, after creating a global ROS node with `rosinit`, you can subscribe to a topic on the global ROS node. When another node on the ROS network publishes messages on that topic, the global ROS node receives the messages.

If a global ROS node already exists, then `rosinit` restarts the global ROS node based on the new set of arguments.

Examples

Start ROS Core and Global Node

```
rosinit
```

```
Initializing ROS master on http://hostname.mathworks.com:11311/.
Initializing global node /matlab_global_node_9152 with NodeURI http://hostname:54194/
```

Start Node and Connect to ROS Master at Specified IP Address

```
rosinit('192.168.1.10')
```

```
Initializing global node /matlab_tped50a5c2_4448_4d11_a523_9829a6b3b5af with NodeURI ht
```

Start Global Node at Given IP and Node Name

```
rosinit('192.168.1.10', 'NodeHost','192.168.1.1','NodeName','/test_node')
```

```
Initializing global node /test_node with NodeURI http://192.168.1.1:64053/
```

Input Arguments

`hostname` — Host name or IP address

string

Host name or IP address, specified as a string.

`port` — Port number
scalar

Port number used to connect to the ROS master, specified as a scalar.

`URI` — URI for ROS master
string

URI for ROS master, specified as a string. Standard format for URIs is either `http://ipaddress:port` or `http://hostname:port`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'NodeHost','192.168.1.1'`

`'NodeHost'` — Host name or IP address
string

Host name or IP address under which the node advertises itself to the ROS network, specified as the comma-separated pair consisting of' 'NodeHost' and a string.

Example: 'comp-home'

`'NodeName'` — Global node name
string

Global node name, specified as the comma-separated pair consisting of 'NodeName' and a string. The node that is created through `rosinit` is registered on the ROS network with this name.

Example: 'NodeName','/test_node'

See Also
`rosshutdown`

Introduced in R2015a

rosmessage

Create ROS messages

Syntax

```
msg = rosmessage(messagetype)

msg = rosmessage(pub)
msg = rosmessage(sub)
msg = rosmessage(client)
msg = rosmessage(server)
```

Description

`msg = rosmessage(messagetype)` creates an empty ROS message object with message type. The `messagetype` string is case-sensitive and no partial matches are allowed. It must match a message on the list given by calling `rosmsg('list')`. To avoid errors in entering the message type, you can use `rostype` with tab completion to browse the list of all available types.

`msg = rosmessage(pub)` creates an empty message determined by the topic published by `pub`.

`msg = rosmessage(sub)` creates an empty message determined by the subscribed topic of `sub`.

`msg = rosmessage(client)` creates an empty message determined by the service associated with `client`.

`msg = rosmessage(server)` creates an empty message determined by the service type of `server`.

Examples

Create Empty String Message

```
strMsg = rosmessage('std_msgs/String');
```

Create Laser Scan Message using `rostype`

```
scan = rosmessage(rostype.sensor_msgs_LaserScan);
```

Create Message to Publish using ROS Publisher

```
chatpub = rospublisher('/chatter','std_msgs/String');
chatmsg = rosmessage(chatpub);
```

Input Arguments

`messagetype` — Message type

string

Message type, specified as a string. The string is case-sensitive and no partial matches are allowed. It must match a message on the list given by calling `rosmsg('list')`. To avoid errors in entering the message type, you can use `rostype` with tab completion to browse the list of all available types.

`pub` — ROS publisher

`Publisher` object handle

ROS publisher, specified as a `Publisher` object handle. You can create the object using `rospublisher`.

`sub` — ROS subscriber

`Subscriber` object handle

ROS subscriber, specified as a `Subscriber` object handle. You can create the object using `rossubscriber`.

`client` — ROS service client

`ServiceClient` object handle

ROS service client, specified as a `ServiceClient` object handle. You can create the object using `rossvcclient`.

`server` — ROS service server
`ServiceServer` object handle

ROS service server, specified as a `ServiceServer` object handle. You can create the object using `rossvcserver`.

Output Arguments

`msg` — ROS message
`Message` object handle

ROS message, returned as a `Message` object handle.

More About

- "Built-In Message Support"

See Also

`roboticsSupportPackages` | `rosmsg` | `rostype`

Introduced in R2015a

rosmsg

Retrieve information about ROS messages and message types

Syntax

```
rosmsg show msgtype
rosmsg md5 msgtype
rosmsg list
```

```
msginfo = rosmsg('show', msgtype)
msgmd5 = rosmsg('md5', msgtype)
msglist = rosmsg('list')
```

Description

`rosmsg show msgtype` returns the definition of the `msgtype` message.

`rosmsg md5 msgtype` returns the MD5 checksum of the `msgtype` message.

`rosmsg list` returns all available message types that you can use in MATLAB.

`msginfo = rosmsg('show', msgtype)` returns the definition of the `msgtype` message as a string.

`msgmd5 = rosmsg('md5', msgtype)` returns the 'MD5' checksum of the `msgtype` message as a string.

`msglist = rosmsg('list')` returns a cell array containing all available message types that you can use in MATLAB.

Examples

Retrieve Message Type Definition

```
msgInfo = rosmsg('show','geometry_msgs/Point')
```

```
msgInfo =
```

```
% This contains the position of a point in free space
double X
double Y
double Z
```

Get the MD5 Checksum of Message Type

```
msgMd5 = rosmsg('md5','geometry_msgs/Point')
```

```
msgMd5 =

4a842b65f413084dc2b10fb484ea7f17
```

Input Arguments

`msgtype` — ROS message type
string

ROS message type, specified as a string. `msgType` must be a valid ROS message type from ROS that MATLAB supports.

Example: 'std_msgs/Int8'

Output Arguments

`msginfo` — Details of message definition
string

Details of the information inside the ROS message definition, returned as a string.

`msgmd5` — MD5 checksum hash value
string

MD5 checksum hash value, returned as a string. The MD5 output is a string representation of the 16-byte hash value that follows the MD5 standard.

`msglist` — List of all message types available in MATLAB
cell array of strings

List of all message types available in MATLAB, returned as a cell array of strings.

Introduced in R2015a

rosnode

Retrieve information about ROS network nodes

Syntax

```
rosnode list
rosnode info nodename
rosnode ping nodename
```

```
nodelist = rosnode('list')
nodeinfo = rosnode('info',nodename)
rosnode('ping',nodename)
```

Description

`rosnode list` returns a list of all nodes registered on the ROS network. Use these nodes to exchange data between MATLAB and the ROS network.

`rosnode info nodename` returns a structure containing the name, URI, publications, subscriptions, and services of a specific ROS node,`nodename`.

`rosnode ping nodename` pings a specific node, `nodename`, and displays the response time.

`nodelist = rosnode('list')` returns a cell array of strings containing the nodes registered on the ROS network.

`nodeinfo = rosnode('info',nodename)` returns a structure containing the name, URI, publications, subscriptions, and services of a specific ROS node, `nodename`.

`rosnode('ping',nodename)` pings a specific node, `nodename` and displays the response time.

Examples

Retrieve List of ROS Nodes

```
rosnode list
```

```
/bumper2pointcloud
/cmd_vel_mux
/depthimage_to_laserscan
/gazebo
/laserscan_nodelet_manager
/matlab_tp8cc35a0e_35fd_4f70_9886_9e489b95b611
/mobile_base_nodelet_manager
/robot_state_publisher
/rosout
```

Retrieve ROS Node Info

```
nodeinfo = rosnode('info','/robot_state_publisher')
```

```
nodeinfo = 

         NodeName: '/robot_state_publisher'
              URI: 'http://192.168.154.132:58140/'
     Publications: [2x1 struct]
    Subscriptions: [2x1 struct]
         Services: [2x1 struct]
```

Ping ROS Node

```
rosnode('ping','/robot_state_publisher')
```

```
Pinging the /robot_state_publisher node with a timeout of 3 seconds.
Ping reply from http://192.168.154.132:58140/, response time = 2.920 ms.
Ping reply from http://192.168.154.132:58140/, response time = 2.138 ms.
Ping reply from http://192.168.154.132:58140/, response time = 2.194 ms.
Ping reply from http://192.168.154.132:58140/, response time = 4.607 ms.
Ping average time: 2.965 ms
```

Input Arguments

`nodename` — Name of node

string

Name of node, specified as a string. The name of the node must match the name given in ROS.

Output Arguments

`nodeinfo` — Information about ROS node

structure

Information about ROS node, returned as a structure containing these properties: `'NodeName'`, `'URI'`, `'Publications'`, `'Subscriptions'`, and `'Services'`. Access these properties using dot syntax, for example, `nodeinfo.NodeName`.

`nodelist` — List of node names available

cell array of strings

List of node names available, returned as a cell array of strings.

See Also

`rosinit` | `rostopic`

Introduced in R2015a

rosparam

Access ROS parameter server values

Syntax

`ptree = rosparam`

Description

`ptree = rosparam` creates a parameter tree object, `ptree`. Once `ptree` is created, the connection to the parameter server remains persistent until the object is deleted or the ROS master becomes unavailable.

A ROS parameter tree communicates with the ROS parameter server. The ROS parameter server can store strings, integers, doubles, booleans and cell arrays. The parameters are accessible by every node in the ROS network. Use the parameters to store static data such as configuration parameters. Use the `get`, `set`, `has`, `search`, and `del` functions to manipulate and view parameter values.

Examples

Create Parameter Tree Object and View Parameters

```
ptree = rosparam
```

```
ptree = 

  ParameterTree with properties:

    AvailableParameters: {40x1 cell}
```

```
ptree.AvailableParameters
```

```
ans = 

    '/bumper2pointcloud/pointcloud_radius'
    '/camera/imager_rate'
```

```
'/camera/rgb/image_raw/compressed/format'
...
```

Output Arguments

ptree — Parameter tree
ParameterTree object handle

Parameter tree, returned as a ParameterTree object handle. Use this object to reference parameter information, for example, ptree.AvailableFrames.

See Also

del | get | has | search | set

Introduced in R2015a

rospublisher

Publish messages on a topic

Syntax

```
pub = rospublisher(topicname)
pub = rospublisher(topicname,msgtype)
pub = rospublisher( ___ ,Name,Value)
[pub,msg] = rospublisher( ___ )
```

```
rospublisher(topicname,msg)
```

Description

`pub = rospublisher(topicname)` creates a publisher, `pub`, for a topic, `topicname`, that already exists on the ROS master topic list. The publisher gets the topic message type from the topic list on the ROS master. When the MATLAB global node publishes messages on that topic, ROS nodes that subscribe to that topic receive those messages. If the topic is not on the ROS master topic list, this function displays an error message. To see a list of available topic names, at the MATLAB command prompt, type `rostopic list`/

`pub = rospublisher(topicname,msgtype)` creates a publisher for a topic and adds that topic to the ROS master topic list. If the ROS master topic list already contains a matching topic, the ROS master adds the MATLAB global node to the list of publishers for that topic. If `msgtype` differs from the topic type on the ROS master topic list, the function displays an error message.

`pub = rospublisher( ___ ,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments using any of the argument from previous syntaxes. `Name` is the property name and `Value` is the corresponding value. `Name` must appear inside single quotes `('')`. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`). Properties not specified retain their default values.

`[pub,msg] = rospublisher( ___ )` returns a message, `msg`, that you can send with the publisher, `pub`. The message is initialized with default values.

`rospublisher(topicname,msg)` publishes a message, `msg`, to the specified topic without creating a publisher.

Examples

Create a Publisher with Specified Message Type and Send String Data

```
chatpub = rospublisher('/chatter','std_msgs/String');
msg = rosmessage(chatpub);
msg.Data = 'Some test string';
send(chatpub,msg);
```

Send Single Message Without Creating a Publisher

```
rospublisher('/chatter',msg)
```

Input Arguments

`topicname` — ROS topic name
string

ROS topic name, specified as a string.

`msgtype` — Message type for ROS topic
string

ROS message type, specified as a string.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IsLatching',false`

`'IsLatching'` — Latch property
`true` (default) | logical

Latch property, specified as the comma-seperated pair consisting of `'isLatching` and a logical. If enabled, latch mode saves the last message sent by the publisher and resends it to new subscribers. By default, latch mode is disabled (`false`). To enable latch mode, set `'IsLatching'` to `true`.

Output Arguments

`pub` — ROS publisher
`Publisher` object handle

ROS publisher, returned as a `Publisher` object handle.

Properties: When you call `rospublisher`, `pub` is returned as a `Publisher` object with the following properties:

- `TopicName` (read-only): Name of the published topic
- `MessageType` (read-only): Message type of published messages
- `IsLatching`: Indicates if publisher is latching
- `NumSubscribers` (read-only): Number of current subscribers for the published topic

To access these properties, use `pub.TopicName`, `pub.MessageType`, `pub.IsLatching`, or `pub.NumSubscribers`.

`msg` — ROS message
`Message` object handle

ROS message, returned as a `Message` object handle.

See Also

`rosmessage` | `rossubscriber`

Introduced in R2015a

rosservice

Retrieve information about services in ROS network

Syntax

```
rosservice list
rosservice info svcname
rosservice type svcname
rosservice uri svcname

svclist = rosservice('list')
svcinfo = rosservice('info',svcname)
svctype = rosservice('type',svcname)
svcuri = rosservice('uri',svcname)
```

Description

`rosservice list` returns a list of service names for all of the active service servers on the ROS network.

`rosservice info svcname` returns information about the specified service, `svcname`.

`rosservice type svcname` returns the service type.

`rosservice uri svcname` returns the URI of the service.

`svclist = rosservice('list')` returns a list of service names for all of the active service servers on the ROS network. `svclist` contains a cell array of service names.

`svcinfo = rosservice('info',svcname)` returns a structure of information, `svcinfo`, about the service, `svcname`.

`svctype = rosservice('type',svcname)` returns the service type of the service as a string.

`svcuri = rosservice('uri',svcname)` returns the URI of the service as a string.

Examples

View List of ROS Services

```
rosservice list
```

```
/bumper2pointcloud/get_loggers
/bumper2pointcloud/set_logger_level
/camera/rgb/image_raw/compressed/set_parameters
...
```

Get Information, Type and URI for ROS Service

Get the service information.

```
svcinfo = rosservice('info','gazebo/pause_physics')
```

```
svcinfo = 

    Node: '/gazebo'
     URI: 'rosrpc://192.168.154.132:33953'
    Type: 'std_srvs/Empty'
    Args: {}
```

Get the service type.

```
svctype = rosservice('type','gazebo/pause_physics')
```

```
svctype =

std_srvs/Empty
```

Get the service URI.

```
svcuri = rosservice('uri','gazebo/pause_physics')
```

```
svcuri =

rosrpc://192.168.154.132:33953
```

Input Arguments

`svcname` — Name of service

string

Name of service, specified as a string. The service name must match its name in the ROS network.

Output Arguments

`svcinfo` — Information about a ROS service
string

Information about a ROS service, returned as a string.

`svclist` — List of available ROS services
cell array of strings

List of available ROS services, returned as a cell array of strings.

`svctype` — Type of ROS service
string

Type of ROS service, returned as a string.

`svcuri` — URI for accessing service
string

URI for accessing service, returned as a string.

See Also
`rosinit` | `rosparam`

Introduced in R2015a

rosshutdown

Shut down ROS system

Syntax

```
rosshutdown
```

Description

`rosshutdown` shuts down the global node and, if it is running, the ROS master. When you finish working with the ROS network, use `rosshutdown` to shut down the global ROS entities created by `rosinit`. If the global node and ROS master are not running, this function has no effect. After calling `rosshutdown`, any ROS entities that depend on the global node, for example, subscribers created with `rossubscriber`, are deleted and become unstable.

Examples

Shut Down Global ROS Node

```
rosshutdown
```

```
Shutting down global node /matlab_global_node_9220 with NodeURI http://hostname:54335/
Shutting down ROS master on http://hostname.mathworks.com:11311/.
```

See Also

`rosinit`

Introduced in R2015a

rossubscriber

Subscribe to messages on a topic

Syntax

```
sub = rossubscriber(topicname)
sub = rossubscriber(topicname,msgtype)

sub = rossubscriber(topicname,callback)
sub = rossubscriber(topicname, msgtype,callback)

sub = rossubscriber( ___ ,Name,Value)
```

Description

`sub = rossubscriber(topicname)` subscribes to a topic with name `topicname`. If the ROS master topic list includes `topicname`, this syntax returns a subscriber object handle, `sub`. If the ROS master topic list does not include the topic, this syntax displays an error. `rossubscriber` enables you to transfer data by subscribing to messages. When ROS nodes publish messages on that topic, MATLAB receives those messages through this subscriber.

`sub = rossubscriber(topicname,msgtype)` subscribes to a topic that has the specified name, `topicname`, and type, `msgtype`. If the topic list on the ROS master does not include a topic with that specified name and type, a topic with the specific name and type is added to the topic list. Use this syntax to avoid errors when it is possible for the subscriber to subscribe to a topic before a publisher has added the topic to the topic list on the ROS master.

`sub = rossubscriber(topicname,callback)` specifies a callback function, `callback` that runs when the subscriber object handle receives a topic message. Use this syntax to avoid the blocking receive function. `callback` can be a single function handle or a cell array. The first element of the cell array must be be a function handle or a string containing the name of a function. The remaining elements of the cell array can be arbitrary user data that is passed to the callback function.

`sub = rossubscriber(topicname, msgtype,callback)` specifies a callback function and subscribes to a topic that has the specified name, `topicname`, and type, `msgtype`.

`sub = rossubscriber( ___ ,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments using any of the argument from previous syntaxes. `Name` is the property name and `Value` is the corresponding value. `Name` must appear inside single quotes `('')`. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`). Properties not specified retain their default values.

Examples

Create Subscriber

```
sub = rossubscriber('/scan');
```

Create Subscriber Using `rostype` for Message Type

Create the subscriber.

```
sub = rossubscriber('/scan', rostype.sensor_msgs_LaserScan);
```

Get the last message from the topic.

```
scan = sub.LatestMessage;
```

Wait to receive the next message and store in `scan`.

```
scan = receive(sub);
```

Create Subscriber Using Callback Function

Create the publisher and subscriber.

```
chatpub = rospublisher('/chatter', rostype.std_msgs_String);
chatsub = rossubscriber('/chatter', @testCallback);
```

Change the Callback Function of Existing Subscriber

```
chatsub = rossubscriber('/chatter', @testCallback);
userData = [5 1; 1 5];
```

```
chatsub.NewMessageFcn = {@func1, userData};
```

Create Subscriber with Specified Buffer Size

```
chatbuf = rossubscriber('/chatter', 'BufferSize', 5);
```

Input Arguments

`topicname` — ROS topic name
string

ROS topic name, specified as a string.

`msgtype` — Message type for ROS topic
string

Message type for ROS topic, specified as a string.

`callback` — Callback function
function handle | cell array

Callback function, specified as a function handle or cell array. In the first element of the cell array, specify either a function handle or a string representing a function name. In subsequent elements, specify user data.

The subscriber callback function requires at least two input arguments. The first argument, `src`, is the associated subscriber object. The second argument, `msg`, is the received message object. The function header for the callback is:

```
function subCallback(src,msg)
```

When setting the callback, you pass additional parameters to the callback function by including both the callback function and the parameters as elements of a cell array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Buffersize',25`

`'BufferSize'` — Buffer size
1 (default) | scalar

Buffer size, specified as the comma-separated pair consisting of `'BufferSize'` and a scalar. If messages arrive faster and than your callback can process them, they will be deleted once the incoming queue is full.

`'NewMessageFcn'` — Callback property
function handle | cell array

Callback property, specified as a function handle or cell array. In the first element of the cell array, specify either a function handle or a string representing a function name. In subsequent elements, specify user data.

The subscriber callback function requires at least two input arguments. The first argument, `src`, is the associated subscriber object. The second argument, `msg`, is the received message object. The function header for the callback is:

```
function subCallback(src,msg)
```

When setting the callback, you pass additional parameters to the callback function by including both the callback function and the parameters as elements of a cell array.

Output Arguments

`sub` — ROS subscriber
`Subscriber` object handle

ROS subscriber, returned as a `Subscriber` object handle. You can create the object using `rossubscriber`.

Properties: When you call `rossubscriber`, `sub` is returned as a `Subscriber` object with the following properties:

- `TopicName` (read-only): Name of the published topic
- `MessageType` (read-only): Message type of published messages
- `LatestMessage` (read-only): Latest message received
- `BufferSize` (read-only): Buffer size of the incoming queue

- `NewMessageFcn`: Callback property for subscriber callbacks

To access these properties, use `sub.TopicName`, `sub.MessageType`, `sub.LatestMessage`, `sub.BufferSize`, or `sub.NewMessageFcn`.

See Also

`rosmessage` | `rospublisher`

Introduced in R2015a

rossvcclient

Create ROS service client

Syntax

```
client = rossvcclient(servicename)
client = rossvcclient(servicename,Name,Value)

[client,reqmsg] = rossvcclient( ___ )
```

Description

`client = rossvcclient(servicename)` creates a service client that connects to, and gets its service type from, a service server. This command syntax blocks the current MATLAB program from running until it can connect to the service server.

Use `rossvcclient` to create a ROS service client. This service client uses a persistent connection to send requests to, and receive responses from, a ROS service server. The connection persists until the service client is deleted or the service server becomes unavailable.

`client = rossvcclient(servicename,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments. Name must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`[client,reqmsg] = rossvcclient( ___ )` returns a new service request message in `reqmsg`, using any of the arguments from previous syntaxes. The message type of `reqmsg` is determined by the service that `client` is connected to. The message is initialized with default values.

Examples

Create Service Client and Wait to Connect to Service

```
client = rossvcclient('/gazebo/get_model_state');
```

Connect to Service Server with Timeout

```
client = rossvcclient('/gazebo/get_model_state', 'Timeout', 3);
```

Create Service Request Message and Call for Response

Create the service request message.

```
request = rosmessage(client);
```

Send the service request and wait for a response.

```
request.ModelName = 'SomeModel';
response = call(client, request);
```

Create a Service Client and Get a Request Message

```
[client,reqmsg] = rossvcclient('/gazebo/get_model_state');
```

Input Arguments

`servicename` — Service name

string

Service name, specified as a string. To access information about active services, such as the service name, use the `rosservice` function.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Timeout',10`

`'Timeout'` — Timeout period in seconds

`inf` (default) | scalar

Timeout period in seconds, specified as a scalar. If the service client does not connect to the service server by the end of the timeout period, `rossvcclient` displays an error message, and MATLAB keeps running the current program. The default value of `inf` blocks MATLAB from running the current program until the service client is connected to the service server.

Output Arguments

`client` — ROS service client

`ServiceClient` object handle

ROS service client, returned as a `ServiceClient` object handle. This service client uses a persistent connection to send requests to, and receive responses from, a ROS service server.

ROS message, returned as a `Message` object handle that matches the request type of the service.

Properties: When you call `rossubscriber`, `client` is returned as a `ServiceClient` object with the following properties:

- `ServerName` (read-only): Name of the service
- `ServiceType` (read-only): Type of the service

To access these properties, use `client.ServerName` or `client.ServerType`.

`reqmsg` — ROS message

`Message` object handle

See Also

`call` | `rosservice` | `rossvcserver`

Introduced in R2015a

rossvcserver

Create ROS service server

Syntax

```
server = rossvcserver(servicename,svctype)
server = rossvcserver(servicename,svctype,callback)

servicename = rossvcserver(servicename,svctype,Name,Value)
```

Description

`server = rossvcserver(servicename,svctype)` creates a service server object of type `svctype` available in the ROS network under the name `servicename`. The service object cannot respond to service requests until you specify a function handle callback.

Use `rossvcserver` to create a ROS service server that can receive requests from, and send responses to, a ROS service client. The service server must exist before creating the service client. When you create the client, it establishes a connection to the server. The connection persists while both client and server exist and can reach each other.

`server = rossvcserver(servicename,svctype,callback)` specifies the function handle callback, `callback`, that constructs a response when the server receives a request. `callback` can be a single function handle or a cell array. The first element of the cell array must be a function handle or a string containing the name of a function. The remaining elements of the cell array can be arbitrary user data that is passed to the callback function.

`servicename = rossvcserver(servicename,svctype,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments using any of the argument from previous syntaxes. `Name` is the property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`). Properties not specified retain their default values.

Examples

Create Service Server

```
server = rossvcserver('/gazebo/get_model_state', rostype.gazebo_msgs_GetModelState)
```

Create Service Server with Callback Function and User Data

Create user data.

```
userData = randi(20);
```

Create a service server.

```
server = rossvcserver('/gazebo/get_model_state2', rostype.gazebo_msgs_GetModelState
   {@func1, userData});
```

Change the callback for a incoming service calls.

```
server.NewRequestFcn = @func2;
```

Input Arguments

`servicename` — Service name

string

Service name, specified as a string. You can access information about active services, such as the service name, using `rosservice`.

`svctype` — Service message type

string

Service message type, specified as a string. You can access information about service message types using `rostype`. Use tab completion to select the message.

`callback` — Callback function and inputs

function handle | cell array

Callback function and inputs, specified as a function handle or a cell array. The first element of the cell array must be a function handle or a string containing the name of a function. The remaining elements of the cell array can be arbitrary user data that is passed to the callback function. The service server callback function requires at least three input arguments and one output. The first argument, `server`, is the associated service server object. The second argument, `reqmsg`, is the request message object sent by the service client. The third argument is the default response message object, `defaultrespmsg`. Use `defaultrespmsg` as a starting point for constructing the function output `response`, which is sent back to the service client.

```
function response = serviceCallback(server,reqmsg,defaultrespmsg)
  response = defaultrespmsg;
  % Build the response message here
end
```

While setting the callback, to construct a callback that accepts additional parameters, use a cell array that includes the function handle callback and the parameters.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'NewMessageFcn',{@func1,userDate}`

`'NewMessageFcn'` — Callback property

function handle | cell array

Callback property, specified as a function handle or a cell array. The first element of the cell array must be a function handle or a string containing the name of a function. The remaining elements of the cell array can be arbitrary user data that is passed to the callback function. The service server callback function requires at least three input arguments and one output. The first argument, `server`, is the associated service server object. The second argument, `reqmsg`, is the request message object sent by the service client. The third argument is the default response message object, `defaultrespmsg`. Use `defaultrespmsg` as a starting point for constructing the function output `response`, which is sent back to the service client.

```
function response = serviceCallback(server,reqmsg,defaultrespmsg)
  response = defaultrespmsg;
  % Build the response message here
end
```

While setting the callback, to construct a callback that accepts additional parameters, use a cell array that includes the function handle callback and the parameters.

Output Arguments

`server` — Service server
`ServiceServer` object handle

Service server, returned as a `ServiceServer` object handle. This service server registers with the ROS master, which enables service clients to send it requests.

Properties: When you call `rossubscriber`, `server` is returned as a `ServiceServer` object with the following properties:

- `ServerName` (read-only): Name of the service
- `ServiceType` (read-only): Type of the service
- `NewRequestFcn`: Callback property for service request callbacks

To access these properties, use `client.ServerName`, `client.ServerType`, or `client.NewRequestFcn`.

See Also
`rossvcclient`

Introduced in R2015a

rostf

Access ROS transformations

Syntax

```
tfTree = rostf
```

Description

`tfTree = rostf` creates a ROS transformation tree object. The object allows you to access the tf coordinate transformations that are shared on the ROS network. You can receive transformations and apply them to different entities. You can also send transformations and share them with the rest of the ROS network.

ROS uses the tf transform library to keep track of the relationship between multiple coordinate frames. The relative transformations between these coordinate frames is maintained in a tree structure. Querying this tree lets you transform entities like poses and points between any two coordinate frames. To access available frames use the syntax:

```
tfTree.AvailableFrames
```

MATLAB can only keep track of the most current information between different frames. ROS tf allows for “time-traveling” or retrieving transformations from specific time instances.

Examples

Create Transformation Tree

```
tree = rostf;
```

Output Arguments

`tfTree` — ROS transformation tree
`TransformationTree` object handle

ROS transformation tree, returned as a `TransformationTree` object handle.

See Also

`getTransform` | `transform`

Introduced in R2015a

rostime

Access ROS time functionality

Syntax

```
time = rostime('now')
[time,issimtime] = rostime('now')
time = rostime('now','system')
```

Description

`time = rostime('now')` returns the current ROS time. If the `use_sim_time` ROS parameter is set to `true`, the `rostime` returns the simulation time published on the `clock` topic. Otherwise, the function returns your machine's system time. `time` is a ROS `Time` object. If no output argument is given, the current time (in seconds) is printed to the screen.

`rostime` can be used to timestamp messages or to measure time in the ROS network.

`[time,issimtime] = rostime('now')` also returns a Boolean that indicates if `time` is in simulation time (`true`) or system time (`false`).

`time = rostime('now','system')` always returns your machine's system time, even if ROS publishes simulation time on the `clock` topic. If no output argument is given, the system time (in seconds) is printed to the screen.

The system time in ROS follows the Unix or POSIX time standard. POSIX time is defined as the time that has elapsed since 00:00:00 Coordinated Universal Time (UTC), 1 January 1970, not counting leap seconds.

Examples

Show Current ROS Time

```
t = rostime('now')
```

```
t =

  ROS Time with properties:

     Sec: 1417812065
    Nsec: 368000000
```

Indicate Whether Time is System Time

```
[t,issim] = rostime('now');
```

```
t =

  ROS Time with properties:

     Sec: 1417812173
    Nsec: 171000000


issim =

     0
```

Timestamp Message Data

```
point = rosmessage('geometry_msgs/PointStamped');
point.Header.Stamp = rostime('now','system');
```

ROS Time to MATLAB Time Example

This example shows how to convert current ROS time into a MATLAB® standard time. The ROS Time object is first converted to a double in seconds, then to the specified MATLAB time.

```
% Sets up ROS network and stores ROS time
rosinit
t = rostime('now');

% Converts ROS time to a double in seconds
secondtime = double(t.Sec)+double(t.Nsec)*10^-9;

% Sets time to a specified MATLAB format
time = datetime(secondtime, 'ConvertFrom','posixtime')

% Shuts down ROS network
```

```
rosshutdown

Initializing ROS master on http://bat5236win64:11311/.
Initializing global node /matlab_global_node_23846 with NodeURI http://bat5236win64:565

time =

   24-Aug-2015 21:12:02

Shutting down global node /matlab_global_node_23846 with NodeURI http://bat5236win64:56
Shutting down ROS master on http://bat5236win64:11311/.
```

Output Arguments

`time` — Current ROS or system time

`Time` object handle

Current ROS or system time, returned as a `Time` object handle. By default, `time` is the ROS simulation time published on the `clock` topic. If the system time if the `use_sim_time` ROS parameter is set to `true`, `time` returns the system time..

`issimtime` — System time indicator

boolean

System time indicator, returned as a boolean. This indicates whether the `time` argument is in simulation time (`true`) or system time (`false`), returned as a Boolean.

See Also

`rosmessage`

Introduced in R2015a

rostopic

Retrieve information about ROS topics

Syntax

```
rostopic list
rostopic echo topicname
rostopic info topicname
rostopic type topicname
```

```
topiclist = rostopic('list')
msg = rostopic('echo', topicname)
topicinfo = rostopic('info', topicname)
msgtype = rostopic('type', topicname)
```

Description

`rostopic list` returns a list of ROS topics from the ROS master.

`rostopic echo topicname` returns the messages being sent from the ROS master about a specific topic, `topicname`. To stop returning messages, press **Ctrl+C**.

`rostopic info topicname` returns the message type, publishers, and subscribers for a specific topic, `topicname`.

`rostopic type topicname` returns the message type for a specific topic.

`topiclist = rostopic('list')` returns a cell array containing the ROS topics from the ROS master. If you do not define the output argument, the list is returned in the MATLAB Command Window.

`msg = rostopic('echo', topicname)` returns the messages being sent from the ROS master about a specific topic, `topicname`. To stop returning messages, press **Ctrl+C**. If the output argument is defined, then `rostopic` returns the first message that arrives on that topic.

`topicinfo = rostopic('info', topicname)` returns a structure containing the message type, publishers, and subscribers for a specific topic, `topicname`.

`msgtype = rostopic('type', topicname)` returns a string containing the message type for the specified topic, `topicname`.

Examples

Get Llist of Topics Available on ROS Master

```
rostopic list
```

```
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
...
```

Get Topic Info for Specified ROS Topic

```
topicinfo = rostopic('info','camera/depth/points')
```

```
topicinfo =

    MessageType: 'sensor_msgs/PointCloud2'
     Publishers: [1x1 struct]
    Subscribers: [0x0 struct]
```

Get Message Type for Specified ROS Topic

```
msgtype = rostopic('type','camera/depth/points')
```

```
msgtype =

sensor_msgs/PointCloud2
```

Input Arguments

`topicname` — ROS topic name

string

ROS topic name, specified as a string. The topic name must match one of the topics that`rostopic('list')` outputs.

Output Arguments

`topiclist` — List of topics from the ROS master
cell array of strings

List of topics from ROS master, returned as a cell array of strings.

`msg` — ROS message for a given topic
object handle

ROS message for a given topic, returned as an object handle.

`topicinfo` — Information about a given ROS topic
structure

Information about a ROS topic, returned as a structure. `topicinfo` included the message type, publishers, and subscribers associated with that topic.

`msgtype` — Message type for a ROS topic
string

Message type for a ROS topic, returned as a string.

Introduced in R2015a

rostype

Access available ROS message types

Syntax

`rostype`

Description

`rostype` creates a blank message of a certain type by browsing the list of available message types. You can use tab completion and do not have to rely on typing error-free message type strings. By typing `rostype.partialstring` , and pressing **Tab**, a list of matching message types appears in a list. By setting the message type equal to a variable, you can create a string of that message type. Alternatively, you can create the message by supplying the message type directly into `rosmessage` as an input argument.

Examples

Create ROS Message Type and ROS Message

```
t = rostype.std_msgs_String
msg = rosmessage(rostype.sensor_msgs_PointCloud2);
```

Introduced in R2015a

rotm2axang

Convert rotation matrix to axis-angle rotation

Syntax

```
axang = rotm2axang(rotm)
```

Description

`axang = rotm2axang(rotm)` converts a rotation given as an orthonormal rotation matrix, `rotm`, to the corresponding axis-angle representation, `axang`. The input rotation matrix must be in the premultiply form for rotations.

Examples

Convert Rotation Matrix to Axis-Angle Rotation

```
rotm = [1 0 0 ; 0 -1 0; 0 0 -1];
axang = rotm2axang(rotm)
```

```
axang =

    1.0000         0         0    3.1416
```

Input Arguments

`rotm` — Rotation matrix
3-by-3-by-*n* matrix

Rotation matrix, specified as a 3-by-3-by-*n* matrix containing *n* rotation matrices. Each rotation matrix has a size of 3-by-3 and is orthonormal. The input rotation matrix must be in the premultiply form for rotations.

Example: `[0 0 1; 0 1 0; -1 0 0]`

Output Arguments

`axang` — Rotation given in axis-angle form

n-by-4 matrix

Rotation given in axis-angle form, returned as an *n*-by-4 matrix of *n* axis-angle rotations. The first three elements of every row specify the rotation axis, and the last element defines the rotation angle (in radians).

Example: `[1 0 0 pi/2]`

See Also

`axang2rotm`

Introduced in R2015a

rotm2eul

Convert rotation matrix to Euler angles

Syntax

```
eul = rotm2eul(rotm)
eul = rotm2eul(rotm,sequence)
```

Description

`eul = rotm2eul(rotm)` converts a rotation matrix, `rotm`, to the corresponding Euler angles, `eul`. The input rotation matrix must be in the premultiply form for rotations. The default order for Euler angle rotations is `'ZYX'`.

`eul = rotm2eul(rotm,sequence)` converts a rotation matrix to Euler angles. The Euler angles are specified in the axis rotation sequence, `sequence`. The default order for Euler angle rotations is `'ZYX'`.

Examples

Convert Rotation Matrix to Euler Angles

```
rotm = [0 0 1; 0 1 0; -1 0 0];
eulZYX = rotm2eul(rotm)
```

```
eulZYX =

         0    1.5708         0
```

Convert Rotation Matrix to Euler Angles Using ZYZ Axis Order

```
rotm = [0 0 1; 0 -1 0; -1 0 0];
eulZYZ = rotm2eul(rotm,'ZYZ')
```

```
eulZYZ =

   -3.1416   -1.5708   -3.1416
```

Input Arguments

`rotm` — Rotation matrix

3-by-3-by-n matrix

Rotation matrix, specified as a 3-by-3-by-n matrix containing n rotation matrices. Each rotation matrix has a size of 3-by-3 and is orthonormal. The input rotation matrix must be in the premultiply form for rotations.

Example: `[0 0 1; 0 1 0; -1 0 0]`

`sequence` — Axis rotation sequence

`'ZYX'` (default) | `'ZYZ'`

Axis rotation sequence for the Euler angles, specified as one of these strings:

- `'ZYX'` (default) – The order of rotation angles is *z*-axis, *y*-axis, *x*-axis.
- `'ZYZ'` – The order of rotation angles is *z*-axis, *y*-axis, *z*-axis.

Output Arguments

`eul` — Euler rotation angles

n-by-3 matrix

Euler rotation angles in radians, returned as an n-by-3 array of Euler rotation angles. Each row represents one Euler angle set.

Example: `[0 0 1.5708]`

See Also

`eul2rotm`

Introduced in R2015a

rotm2quat

Convert rotation matrix to quaternion

Syntax

```
quat = rotm2quat(rotm)
```

Description

`quat = rotm2quat(rotm)` converts a rotation matrix, `rotm`, to the corresponding unit quaternion representation, `quat`. The input rotation matrix must be in the premultiply form for rotations.

Examples

Convert Rotation Matrix to Quaternion

```
rotm = [0 0 1; 0 1 0; -1 0 0];
quat = rotm2quat(rotm)
```

```
quat =

    0.7071         0    0.7071         0
```

Input Arguments

`rotm` — Rotation matrix
3-by-3-by-*n* matrix

Rotation matrix, specified as a 3-by-3-by-*n* matrix containing *n* rotation matrices. Each rotation matrix has a size of 3-by-3 and is orthonormal. The input rotation matrix must be in the premultiply form for rotations.

Example: `[0 0 1; 0 1 0; -1 0 0]`

Output Arguments

`quat` — Unit quaternion

n-by-4 matrix

Unit quaternion, returned as an *n*-by-4 matrix containing *n* quaternions. Each quaternion, one per row, is of the form $q = [w\ x\ y\ z]$, with *w* as the scalar number.

Example: `[0.7071 0.7071 0 0]`

See Also

`quat2rotm`

Introduced in R2015a

rotm2tform

Convert rotation matrix to homogeneous transformation

Syntax

```
tform = rotm2tform(rotm)
```

Description

`tform = rotm2tform(rotm)` converts the rotation matrix, `rotm`, into a homogeneous transformation matrix, `tform`. The input rotation matrix must be in the premultiply form for rotations. When using the transformation matrix, premultiply it with the coordinates to be transformed (as opposed to postmultiplying).

Examples

Convert Rotation Matrix to Homogeneous Transformation

```
rotm = [1 0 0 ; 0 -1 0; 0 0 -1];
tform = rotm2tform(rotm)
```

```
tform =

     1     0     0     0
     0    -1     0     0
     0     0    -1     0
     0     0     0     1
```

Input Arguments

`rotm` — Rotation matrix
3-by-3-by-n matrix

Rotation matrix, specified as a 3-by-3-by-*n* matrix containing *n* rotation matrices. Each rotation matrix has a size of 3-by-3 and is orthonormal. The input rotation matrix must be in the premultiply form for rotations.

Example: `[0 0 1; 0 1 0; -1 0 0]`

Output Arguments

`tform` — Homogeneous transformation
4-by-4-by-*n* matrix

Homogeneous transformation matrix, specified by a 4-by-4-by-*n* matrix of *n* homogeneous transformations. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

See Also

`tform2rotm`

Introduced in R2015a

scatter3

Display point cloud in scatter plot

Syntax

```
scatter3(pcloud)
scatter3(pcloud,Name,Value)
h = scatter3( ___ )
```

Description

`scatter3(pcloud)` plots the input `pcloud` point cloud as a 3-D scatter plot in the current axes handle. If the data contains RGB information for each point, the scatter plot is colored accordingly.

`scatter3(pcloud,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments. Name must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`)

`h = scatter3( ___ )` returns the scatter series object, using any of the arguments from previous syntaxes. Use `h` to modify properties of the scatter series after it is created.

When plotting ROS point cloud messages, MATLAB follows the standard ROS convention for axis orientation. This convention states that **positive *x* is forward, positive *y* is left, and positive *z* is up**. However, if cameras are used, a second frame is defined with an "_optical" suffix which changes the orientation of the axis. In this case, positive *z* is forward, positive *x* is right, and positive *y* is down. MATLAB looks for the "_optical" suffix and will adjust the axis orientation of the scatter plot accordingly. For more information, see Axis Orientation on the ROS Wiki.

Examples

Show 3-D Point Cloud

```
scatter3(pcloud);
```

Show 3-D Ppoint Cloud with Uniform Red Points

```
scatter3(pcloud,'MarkerEdgeColor',[1 0 0]);
```

Input Arguments

`pcloud` — Point cloud
`PointCloud2` object handle

Point cloud, specified as a `PointCloud2` object handle for a `'sensor_msgs/PointCloud2'` ROS message.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MarkerEdgeColor',[1 0 0]`

`'MarkerEdgeColor'` — Marker outline color
`'flat'` (default) | `'none'` | RGB triplet | color string

Marker outline color, specified as one of these values:

- `'flat'` — Colors defined by the `CData` property.
- `'none'` — No color, which makes unfilled markers invisible.
- RGB triplet or color string — Specify a custom color.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
'yellow'	'y'	[1 1 0]
'magenta'	'm'	[1 0 1]
'cyan'	'c'	[0 1 1]
'red'	'r'	[1 0 0]
'green'	'g'	[0 1 0]
'blue'	'b'	[0 0 1]
'white'	'w'	[1 1 1]
'black'	'k'	[0 0 0]

Example: [0.5 0.5 0.5]

Example: 'blue'

'Parent' — Parent of axes
axes object

Parent of axes, specified as the comma-separated pair consisting of 'Parent and an axes object in which to draw the point cloud. By default, the point cloud is plotted in the active axes.

Outputs

h — Scatter series object
scalar

Scatter series object, returned as a scalar. This value is a unique identifier, which you can use to query and modify the properties of the scatter object after it is created.

See Also

readRGB | readXYZ

Introduced in R2015a

search

Search ROS network for parameter names

Syntax

```
pnames = search(ptree,searchstr)
[pnames,pvalues] = search(ptree,searchstr)
```

Description

`pnames = search(ptree,searchstr)` searches within the parameter tree `ptree` and returns the parameter names that contain the string `searchstr`.

`[pnames,pvalues] = search(ptree,searchstr)` also returns the parameter values.

Examples

Search for Parameter Names and Values Using Partial String

```
[pnames,pvalues] = search(ptree,'gravity')
```

```
pnames =

    '/gazebo/gravity_x'    '/gazebo/gravity_y'    '/gazebo/gravity_z'


pvalues =

    [      0]
    [      0]
    [-9.8000]
```

Input Arguments

`ptree` — Parameter tree

`ParameterTree` object handle

Parameter tree, specified as a `ParameterTree` object handle. Create this object using the `rosparam` function.

`searchstr` — ROS parameter search string

string

ROS parameter search string. `search` returns all parameters that contain this string.

Output Arguments

`pnames` — Parameter values

cell array of strings

Parameter names, returned as a cell array of strings. These strings match the parameter names in the ROS master that contain the search string.

`pvalues` — Parameter values

cell array

Parameter values, returned as a cell array. These values vary, but it should match the value expected for each parameter name in the array. Supported values are

- `int32`
- `logical`
- `double`
- string
- cell array

Currently, Base64–encoded binary data and iso8601 data from ROS are not supported.

See Also

`get` | `rosparam`

Introduced in R2015a

select

Select subset of messages in rosbag

Syntax

```
bagsel = select(bag)
bagsel = select(bag,Name,Value)
```

Description

`bagsel = select(bag)` returns an object, `bagsel`, that contains all of the messages in the `BagSelection` object, `bag`

This function does not change the contents of the original `BagSelection` object. It returns a new object that contains the specified message selection.

`bagsel = select(bag,Name,Value)` provides additional options specified by one or more `Name,Value` pair arguments. `Name`must appear inside single quotes `('')`. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Examples

Create Copy of rosbag

Retrieve a rosbag file.

```
bag = rosbag(filepath);
```

Copy the bag using the `select` function.

```
bagCopy = select(bag);
```

Select Message Based on Time

Get the messages from the first full second of the rosbag.

```
bagMsgs = select(bagMsgs, 'Time', [bagMsgs.StartTime, ...
          bagMsgs.StartTime + 1])
```

Input Arguments

bag — Message of a rosbag
BagSelection object

All the messages contained within a rosbag, specified as a `BagSelection` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MessageType','/geometry_msgs/Point'`

'MessageType' — ROS message type
string | cell array

ROS message type, specified as a string or cell array. Multiple message types can be specified with a cell array of strings.

'Time' — Start and end times
n-by-2 matrix

Start and end times of the rosbag selection, specified as an n-by-2 vector.

'Topic' — ROS topic name
string | cell array

ROS topic name, specified as a string or cell array. Multiple topic names can be specified with a cell array of strings.

Output Arguments

bagsel — Copy or subset of rosbag messages
BagSelection object

Copy or subset of rosbag messages, returned as a `BagSelection` object

See Also

`readMessages` | `rosbag` | `timeseries`

Introduced in R2015a

send

Publish ROS message to topic

Syntax

```
send(pub,msg)
```

Description

`send(pub,msg)` publishes a message to the topic specified by the publisher, `pub`. This message can be received by all subscribers in the ROS network that are subscribed to the topic specified by `pub`

Examples

Publish Message Using send

```
send(pub,msg);
```

Create, Send and Receive Message

Set up a topic, publisher, and subscriber to share and receive a message.

Create a topic and publisher.

```
msgtype = rostype.geometry_msgs_Point;
pub = rospublisher('position', msgtype);
```

Create a message.

```
msg = rosmessage(msgtype);
msg.Y = 2
```

```
msg =

  ROS Point message with properties:

    MessageType: 'geometry_msgs/Point'
```

```
              X: 0
              Y: 2
              Z: 0

  Use showdetails to show the contents of the message
```

Send the message.

```
send(pub,msg)
```

Subscribe to the publisher.

```
sub = rossubscriber('position',msgtype)
```

```
sub = 

  Subscriber with properties:

        TopicName: '/position'
      MessageType: 'geometry_msgs/Point'
    LatestMessage: [1x1 Point]
       BufferSize: 25
    NewMessageFcn: []
```

Verify that the latest message received is correct.

```
sub.LatestMessage
```

```
ans = 

  ROS Point message with properties:

    MessageType: 'geometry_msgs/Point'
              X: 0
              Y: 2
              Z: 0

  Use showdetails to show the contents of the message
```

Input Arguments

pub — ROS publisher

`Publisher` object handle

ROS publisher, specified as a `Publisher` object handle. You can create the object using `rospublisher`.

`msg` — ROS message
`Message` object handle

ROS message, specified as a `Message` object handle.

See Also

`rospublisher` | `rostopic`

Introduced in R2015a

sendTransform

Send transformation to ROS network

Syntax

```
sendTransform(tftree,tf)
```

Description

`sendTransform(tftree,tf)` broadcasts a transform or array of transforms, `tf`, to the ROS network as a `TransformationStamped` ROS message.

Examples

Send Transformation to ROS Network

```
tftree = rostf
tf = gettransform(tftree,'/camera_depth_frame','/base_link');
sendTransform(tftree,tf)
```

Input Arguments

`tftree` — ROS transformation tree

`TransformationTree` object handle

ROS transformation tree, specified as a `TransformationTree` object handle. You can create a transformation tree by calling the `rostf` function.

`tf` — Transformations between coordinate frames

`TransformStamped` object handle | array of object handles

Transformations between coordinate frames, returned as a `TransformStamped` object handle or as an array of object handles. Transformations are structured as a 3-D translation (3-element vector) and a 3-D rotation (quaternion).

See Also

`getTransform` | `transform`

Introduced in R2015a

set

Set value of ROS parameter; add new parameter

Syntax

`set(ptree,paramname,pvalue)`

Description

`set(ptree,paramname,pvalue)` assigns the value `pvalue` to the parameter with the name `paramname`, which is contained in the parameter tree `ptree`.

Examples

Set and Get Parameter Value

```
ptree = rosparam;
set(ptree,'DoubleParam',1.0)
get(ptree,'DoubleParam')
```

```
ans =

     1
```

Input Arguments

`ptree` — Parameter tree
`ParameterTree` object handle

Parameter tree, specified as a `ParameterTree` object handle. Create this object using the `rosparam` function.

`paramname` — ROS parameter name
string

ROS parameter name, specified as a string. This string must match the parameter name exactly.

`pvalue` — Parameter value
int32 | logical | char | double | cell array

Parameter value, returned as either a `int32`, `logical`, `double`, `char`, or `cell array`. `pvalue` matches the value of the specified`paramname` and the supported data type in `ParameterTree`. Currently, Base64–encoded binary data and iso8601 data from ROS are not supported.

See Also
`get` | `rosparam`

Introduced in R2015a

showdetails

Display all ROS message contents

Syntax

```
details = showdetails(msg)
```

Description

`details = showdetails(msg)` gets all data contents of message object `msg`. The details are stored in `details` or displayed on the command line.

Examples

Create Message and View Details

Create a message.

```
msg = rosmessage(rostype.geometry_msgs_Point);
msg.X = 1;
msg.Y = 2;
msg.Z = 3;
```

View the message details.

```
showdetails(msg)
```

```
  X :  1
  Y :  2
  Z :  3
```

Input Arguments

`msg` — ROS message
`Message` object handle

ROS message, specified as a `Message` object handle.

Output Arguments

`details` — Details of ROS message
string

Details of ROS message, returned as a string.

See Also

`rosmessage`

Introduced in R2015a

tform2axang

Convert homogeneous transformation to axis-angle rotation

Syntax

`axang = tform2axang(tform)`

Description

`axang = tform2axang(tform)` converts the rotational component of a homogeneous transformation, `tform`, to an axis-angle rotation, `axang`. The translational components of `tform` are ignored. The input homogeneous transformation must be in the premultiply form for transformations.

Examples

Convert Homogeneous Transformation to Axis-Angle Rotation

```
tform = [1 0 0 0; 0 0 -1 0; 0 1 0 0; 0 0 0 1];
axang = tform2axang(tform)
```

```
axang =

    1.0000         0         0    1.5708
```

Input Arguments

`tform` — Homogeneous transformation

4-by-4-by-n matrix

Homogeneous transformation, specified by a 4-by-4-by-n matrix of n homogeneous transformations. The input homogeneous transformation must be in the premultiply form for transformations.

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

Output Arguments

`axang` — Rotation given in axis-angle form

n-by-4 matrix

Rotation given in axis-angle form, specified as an *n*-by-4 matrix of *n* axis-angle rotations. The first three elements of every row specify the rotation axes, and the last element defines the rotation angle (in radians).

Example: `[1 0 0 pi/2]`

See Also

`axang2tform`

Introduced in R2015a

tform2eul

Extract Euler angles from homogeneous transformation

Syntax

```
eul = tform2eul(tform)
eul = tform2eul(tform, sequence)
```

Description

`eul = tform2eul(tform)` extracts the rotational component from a homogeneous transformation, `tform`, and returns it as Euler angles, `eul`. The translational components of `tform` are ignored. The input homogeneous transformation must be in the premultiply form for transformations. The default order for Euler angle rotations is `'ZYX'`.

`eul = tform2eul(tform, sequence)` extracts the Euler angles, `eul`, from a homogeneous transformation, `tform`, using the specified rotation sequence, `sequence`. The default order for Euler angle rotations is `'ZYX'`.

Examples

Extract Euler Angles from Homogeneous Transformation Matrix

```
tform = [1 0 0 0.5; 0 -1 0 5; 0 0 -1 -1.2; 0 0 0 1];
eulZYX = tform2eul(tform)
```

```
eulZYX =

         0         0    3.1416
```

Extract Euler Angles from Homogeneous Transformation Matrix Using ZYZ Rotation

```
tform = [1 0 0 0.5; 0 -1 0 5; 0 0 -1 -1.2; 0 0 0 1];
eulZYZ = tform2eul(tform,'ZYZ')
```

```
eulZYZ =

         0   -3.1416    3.1416
```

Input Arguments

`tform` — Homogeneous transformation
4-by-4-by-n matrix

Homogeneous transformation, specified by a 4-by-4-by-n matrix of n homogeneous transformations. The input homogeneous transformation must be in the premultiply form for transformations.

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

`sequence` — Axis rotation sequence
`'ZYX'` (default) | `'ZYZ'`

Axis rotation sequence for the Euler angles, specified as one of these strings:

- `'ZYX'` (default) – The order of rotation angles is z-axis, y-axis, x-axis.
- `'ZYZ'` – The order of rotation angles is z-axis, y-axis, z-axis.

Output Arguments

`eul` — Euler rotation angles
n-by-3 matrix

Euler rotation angles in radians, returned as an n-by-3 array of Euler rotation angles. Each row represents one Euler angle set.

Example: `[0 0 1.5708]`

See Also

`eul2tform`

Introduced in R2015a

tform2quat

Extract quaternion from homogeneous transformation

Syntax

`quat = tform2quat(tform)`

Description

`quat = tform2quat(tform)` extracts the rotational component from a homogeneous transformation, `tform`, and returns it as a quaternion, `quat`. The translational components of `tform` are ignored. The input homogeneous transformation must be in the premultiply form for transformations.

Examples

Extract Quaternion from Homogeneous Transformation

```
tform = [1 0 0 0; 0 -1 0 0; 0 0 -1 0; 0 0 0 1];
quat = tform2quat(tform)
```

```
quat =

     0     1     0     0
```

Input Arguments

`tform` — Homogeneous transformation
4-by-4-by-*n* matrix

Homogeneous transformation, specified by a 4-by-4-by-*n* matrix of *n* homogeneous transformations. The input homogeneous transformation must be in the premultiply form for transformations.

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

Output Arguments

`quat` — Unit quaternion
n-by-4 matrix

Unit quaternion, returned as an *n*-by-4 matrix containing *n* quaternions. Each quaternion, one per row, is of the form $q = [w\ x\ y\ z]$, with *w* as the scalar number.

Example: `[0.7071 0.7071 0 0]`

See Also

`quat2tform`

Introduced in R2015a

tform2rotm

Extract rotation matrix from homogeneous transformation

Syntax

```
rotm = tform2rotm(tform)
```

Description

`rotm = tform2rotm(tform)` extracts the rotational component from a homogeneous transformation, `tform`, and returns it as an orthonormal rotation matrix, `rotm`. The translational components of `tform` are ignored. The input homogeneous transformation must be in the pre-multiply form for transformations. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Examples

Convert Homogeneous Transformation to Rotation Matrix

```
tform = [1 0 0 0; 0 -1 0 0; 0 0 -1 0; 0 0 0 1];
rotm = tform2rotm(tform)
```

```
rotm =

     1     0     0
     0    -1     0
     0     0    -1
```

Input Arguments

`tform` — Homogeneous transformation
4-by-4-by-*n* matrix

Homogeneous transformation matrix, specified by a 4-by-4-by-*n* matrix of *n* homogeneous transformations. The input homogeneous transformation must be in the pre-multiply form for transformations.

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

Output Arguments

`rotm` — Rotation matrix
3-by-3-by-*n* matrix

Rotation matrix, returned as a 3-by-3-by-*n* matrix containing *n* rotation matrices. Each rotation matrix has a size of 3-by-3 and is orthonormal. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Example: `[0 0 1; 0 1 0; -1 0 0]`

See Also

`rotm2tform`

Introduced in R2015a

tform2trvec

Extract translation vector from homogeneous transformation

Syntax

```
trvec = tform2trvec(tform)
```

Description

`trvec = tform2trvec(tform)` extracts the Cartesian representation of translation vector, `trvec` , from a homogeneous transformation, `tform`. The rotational components of `tform` are ignored. The input homogeneous transformation must be in the premultiply form for transformations.

Examples

Extract Translation Vector from Homogeneous Transformation

```
tform = [1 0 0 0.5; 0 -1 0 5; 0 0 -1 -1.2; 0 0 0 1];
trvec = tform2trvec(tform)
```

```
trvec =

    0.5000    5.0000   -1.2000
```

Input Arguments

`tform` — Homogeneous transformation
4-by-4-by-*n* matrix

Homogeneous transformation, specified by a 4-by-4-by-n matrix of n homogeneous transformations. The input homogeneous transformation must be in the premultiply form for transformations.

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

Output Arguments

`trvec` — Cartesian representation of a translation vector

n-by-3 matrix

Cartesian representation of a translation vector, returned as an *n*-by-3 matrix containing *n* translation vectors. Each vector is of the form $t = [x\ y\ z]$.

Example: `[0.5 6 100]`

See Also

`trvec2tform`

Introduced in R2015a

timeseries

Creates a time series object for selected message properties

Syntax

```
[ts,cols] = timeseries(bag)
[ts,cols] = timeseries(bag,property)
[ts,cols] = timeseries(bag,property,...,propertyN)
```

Description

`[ts,cols] = timeseries(bag)` creates a time series for all numeric and scalar message properties. The function evaluates each message in the current `BagSelection` object, `bag`, as `ts`. The `cols` output argument stores property names as a cell array of strings.

The returned time series object is memory-efficient because it stores only particular message properties instead of whole messages.

`[ts,cols] = timeseries(bag,property)` creates a time series for a specific message property, `property`. Property names can also be nested, for example, `'Pose.Pose.Position.X'` for the *x*-axis position of a robot.

`[ts,cols] = timeseries(bag,property,...,propertyN)` creates a time series for a range specific message properties. Each property is a different column in the time series object.

Examples

Create Time Series from Entire Bag Selection

```
ts = timeseries(bagMsgs);
```

Create Time Series with Single Property

```
ts = timeseries(bagMsgs, 'Pose.Pose.Position.X');
```

Create Time Series with Multiple Properties

```
ts = timeseries(bagMsgs, 'Twist.Twist.Angular.X', ...
          'Twist.Twist.Angular.Y', 'Twist.Twist.Angular.Z')
```

Input Arguments

`bag` — Bag selection

`BagSelection` object handle

Bag selection, specified as a `BagSelection` object handle. You can get a bag selection by calling `rosbag`.

`property` — Property names

string

Property names, specified as a string. Multiple properties can be specified. Each property name is a separate input and represents a different column in the time series object.

Output Arguments

`ts` — Time series

`Time` object handle

Time series, returned as a `Time` object handle.

`cols` — List of property names

cell array of strings

List of property names, returned as a cell array of strings.

More About

- “Time Series Basics”

See Also

`readMessages` | `rosbag` | `select`

Introduced in R2015a

transform

Transform message entities into target coordinate frame

Syntax

```
tfentity = transform(tftree,targetframe,entity)
```

Description

`tfentity = transform(tftree,targetframe,entity)` retrieves the transformation between `targetframe` and `entity` and applies it to `entity`, a ROS message of a specific type. `tftree` is the full transformation tree containing known transformations between entities. If the transformation from `entity` to `targetframe` does not exist, MATLAB thows an error.

Examples

Transform `PointStamped` Message

Define a point in the coordinate frame of a camera.

```
pt = rosmessage('geometry_msgs/PointStamped');
       pt.Header.FrameId = '/camera_depth_frame';
       pt.Point.X = 3;
       pt.Point.Y = 1.5;
       pt.Point.Z = 0.2;
```

Transform the point to the `base_link` frame.

```
tfpt = transform(tftree, '/base_link', pt)
```

Input Arguments

`tftree` — ROS transformation tree

`TransformationTree` object handle

ROS transformation tree, specified as a `TransformationTree` object handle. You can create a transformation tree by calling the `rostf` function.

`targetframe` — Target coordinate frame
string

Target coordinate frame that entity transforms into, specified as a string. You can view the available frames for transformation calling `tftree.AvailableFrames`.

`entity` — Initial message entity
`Message` object handle

Initial message entity, specified as a `Message` object handle.

Supported messages are:

- `geometry_msgs/PointStamped`
- `geometry_msgs/PoseStamped`
- `geometry_msgs/QuaternionStamped`
- `geometry_msgs/Vector3Stamped`
- `sensor_msgs/PointCloud2`

Output Arguments

`tfentity` — Transformed entity
`Message` object handle

Transformed entity, returned as a `Message` object handle.

See Also

`getTransform` | `waitForTransform`

Introduced in R2015a

trvec2tform

Convert translation vector to homogeneous transformation

Syntax

```
tform = trvec2tform(trvec)
```

Description

`tform = trvec2tform(trvec)` converts the Cartesian representation of a translation vector, `trvec`, to the corresponding homogeneous transformation, `tform`. When using the transformation matrix, premultiply it with the coordinates to be transformed (as opposed to postmultiplying).

Examples

Convert Translation Vector to Homogeneous Transformation

```
trvec = [0.5 6 100];
tform = trvec2tform(trvec)
```

```
tform =

    1.0000         0         0    0.5000
         0    1.0000         0    6.0000
         0         0    1.0000  100.0000
         0         0         0    1.0000
```

Input Arguments

`trvec` — Cartesian representation of a translation vector

n-by-3 matrix

Cartesian representation of a translation vector, specified as an n-by-3 matrix containing n translation vectors. Each vector is of the form $t = [x\ y\ z]$.

Example: `[0.5 6 100]`

Output Arguments

`tform` — Homogeneous transformation

4-by-4-by-n matrix

Homogeneous transformation matrix, returned as a 4-by-4-by-n matrix of n homogeneous transformations. When using the rotation matrix, premultiply it with the coordinates to be rotated (as opposed to postmultiplying).

Example: `[0 0 1 0; 0 1 0 0; -1 0 0 0; 0 0 0 1]`

See Also

`tform2trvec`

Introduced in R2015a

waitForTransform

Wait until a transformation is available

Syntax

```
waitForTransform(tftree,targetframe,sourceframe)
waitForTransform(tftree,targetframe,sourceframe,timeout)
```

Description

`waitForTransform(tftree,targetframe,sourceframe)` waits until the transformation between `targetframe` and `sourceframe` is available in the transformation tree, `tftree`. This functions disables the command prompt until a transformation becomes available on the ROS network.

`waitForTransform(tftree,targetframe,sourceframe,timeout)` specifies a timeout period in seconds. If the transformation does not become available, MATLAB displays an error, but continues running the current program.

Examples

Wait for Transform

```
waitForTransform(tftree,'/camera_depth_frame','/base_link');
```

Specify Timeout of Five Seconds to Wait for Transform

```
waitForTransform(tftree,'/camera_depth_frame','/base_link',5);
```

Input Arguments

`tftree` — ROS transformation tree
`TransformationTree` object handle

ROS transformation tree, specified as a `TransformationTree` object handle. You can create a transformation tree by calling the `rostf` function.

`targetframe` — Target coordinate frame
string

Target coordinate frame, specified as a string. You can view the available frames for transformation by calling `tftree.AvailableFrames`.

`sourceframe` — Initial coordinate frame
string

Initial coordinate frame, specified as a string. You can view the available frames for transformation using `tftree.AvailableFrames`.

`timeout` — Timeout period
scalar in seconds

Timeout period, specified as a scalar in seconds. If the transformation does not become available, MATLAB displays an error, but continues running the current program.

See Also

`getTransform` | `receive` | `transform`

Introduced in R2015a

writeBinaryOccupancyGrid

Write values from grid to ROS message

Syntax

```
writeBinaryOccupancyGrid(msg,map)
```

Description

`writeBinaryOccupancyGrid(msg,map)` writes occupancy values and other information to the ROS message, `msg`, from the binary occupancy grid, `map`.

Examples

Write Binary occupancy Grid Information to ROS Message

```
map = robotics.BinaryOccupancyGrid(randi([0,1], 10));
msg = rosmessage('nav_msgs/OccupancyGrid');
writeBinaryOccupancyGrid(msg, map);
```

Input Arguments

`map` — Binary occupancy grid
`BinaryOccupancyGrid` object handle

Binary occupancy grid, specified as a `BinaryOccupancyGrid` object handle. `map` is converted to a `'nav_msgs/OccupancyGrid'` message on the ROS network. `map` is an object with a grid of binary values, where `1` indicates an occupied location and `0` indications an unoccupied location.

`msg` — `'nav_msgs/OccupancyGrid'` ROS message
`OccupancyGrid` object handle

`'nav_msgs/OccupancyGrid'` ROS message, specified as a `OccupancyGrid` object handle.

See Also

robotics.BinaryOccupancyGrid | `readBinaryOccupancyGrid`

Introduced in R2015a

writeImage

Write MATLAB image to ROS image message

Syntax

```
writeImage(msg,img)
writeImage(msg,img,alpha)
```

Description

`writeImage(msg,img)` converts the MATLAB image, `img`, to a message object and stores the ROS compatible image data in the message object, `msg`. The message must be a `'sensor_msgs/Image'` message. `'sensor_msgs/CompressedImage'` messages are not supported.

`writeImage(msg,img,alpha)` converts the MATLAB image, `img` to a message object. If the image encoding supports an alpha channel (`rgba` or `bgra` family), specify this alpha channel in `alpha`. Alternatively, the input image can store the alpha channel as its fourth channel.

Examples

Write Image to Message

```
msg = rosmessage('sensor_msgs/Image')
writeImage(msg,img);
```

Write Message Using Alpha Channel

```
writeImage(msg,img,alpha);
```

Input Arguments

`msg` — ROS image message
`Image` object handle

'sensor_msgs/Image' ROS image message, specified as an Image object handle. 'sensor_msgs/Image' image messages are not supported.

img — Image
grayscale image matrix | RBG image matrix | *m*-by-*n*-by-3 array

Image, specified as a matrix representing a grayscale or RGB image or as a*m*-by-*n*-by-3 array, depending on the sensor image.

alpha — Alpha channel
uint8 grayscale image

Alpha channel, specified as a uint8 grayscale image. Alpha must be the same size and data type as img.

More About

Tips

You must specify encoding of the input image in the 'Encoding' property of the image message. If you do not specify the image encoding before calling the function, the default encoding, rgb8, is used (3-channel RGB image with uint8 values).

All encoding types supported for the readImage are also supported in this function. For more information on supported encoding types and their representations in MATLAB, see readImage.

Bayer-encoded images (bayer_rggb8, bayer_bggr8, bayer_gbrg8, bayer_grbg8 and their 16-bit equivalents) must be given as 8-bit or 16-bit single-channel images or they do not encode.

See Also
readImage

Introduced in R2015a

Methods — Alphabetical List

copy

Class: robotics.BinaryOccupancyGrid
Package: robotics

Copy array of handle objects

Syntax

`b = copy(a)`

Description

`b = copy(a)` copies each element in the array of handles, `a`, to the new array of handles, `b`.

The `copy` method does not copy dependent properties. MATLAB does not call `copy` recursively on any handles contained in property values. MATLAB does not call the class constructor or property set methods during the copy operation.

`b` has the same number of elements and is the same size and class of`a`. `b` is the same class as `a`. If `a` is empty, `b` is also empty. If `a` is heterogeneous, `b` is also heterogeneous. If `a` contains deleted handles, then `copy` creates deleted handles of the same class in `b`. Dynamic properties and listeners associated with objects in `a` are not copied to objects in `a`.

`copy` is a sealed and public method in class `matlab.mixin.Copyable`.

Input Arguments

a — Oject array
handle

Object array, specified as a handle.

Output Arguments

b — Object array containing copies of the objects in a
handle

Object array containing copies of the object in a, specified as a handle.

See Also

robotics.BinaryOccupancyGrid

Introduced in R2015a

getOccupancy

Class: robotics.BinaryOccupancyGrid
Package: robotics

Get occupancy value for one or more positions

Syntax

```
occval = getOccupancy(map,xy)
occval = getOccupancy(map,ij,'grid')
```

Description

`occval = getOccupancy(map,xy)` returns an array of occupancy values for an input array of world coordinates, `xy`. Each row of `xy` is a point in the world, represented as an `[x y]` coordinate pair. `occval` is the same length as `xy` and a single column array. An occupied location is represented as `true` (`1`), and a free location is represented as `false` (`0`).

`occval = getOccupancy(map,ij,'grid')` returns an array of occupancy values based on a `[rows cols]`input array of grid positions, `ij`.

Input Arguments

`map` — Map representation

`BinaryOccupancyGrid` object

Map representation, specified as a robotics.BinaryOccupancyGrid object. This object represents the environment of the robot. The object contains a matrix grid with binary values indicating obstacles as `true` (`1`) and free locations as `false` (`0`).

`xy` — World coordinates

n-by-2 vertical array

World coordiantes, specified as an n-by-2 vertical array of `[x y]` pairs, where n is the number of world coordinates.

Data Types: `double`

`ij` — Grid positions
n-by-2 vertical array

Grid positions, specified as an *n*-by-2 vertical array of `[i j]` pairs in `[rows cols]` format, where *n* is the number of grid positions.

Data Types: `double`

Output Arguments

`occval` — Occupancy values
n-by-1 vertical array

Occupancy values of the same length as either `xy` or `ij`, returned as an *n*-by-1 vertical array, where *n* is the same *n* in either `xy` or `ij`.

See Also

robotics.BinaryOccupancyGrid | robotics.BinaryOccupancyGrid.setOccupancy

Introduced in R2015a

grid2world

Class: robotics.BinaryOccupancyGrid
Package: robotics

Convert grid indices to world coordinates

Syntax

```
xy = grid2world(map,ij)
```

Description

`xy = grid2world(map,ij)` converts a `[row col]` array of grid indices, `ij`, to an array of world coordinates, `xy`.

Input Arguments

`map` — Map representation

`BinaryOccupancyGrid` object

Map representation, specified as a robotics.BinaryOccupancyGrid object. This object represents the environment of the robot. The object contains a matrix grid with binary values indicating obstacles as `true` (1) and free locations as `false` (0).

`ij` — Grid positions

n-by-2 vertical array

Grid positions, specified as an *n*-by-2 vertical array of `[i j]` pairs in `[rows cols]` format, where *n* is the number of grid positions.

Output Arguments

`xy` — World coordinates

n-by-2 vertical array

World coordiantes, specified as an *n*-by-2 vertical array of `[x y]` pairs, where *n* is the number of world coordinates.

See Also

robotics.BinaryOccupancyGrid | robotics.BinaryOccupancyGrid.world2grid

Introduced in R2015a

inflate

Class: robotics.BinaryOccupancyGrid
Package: robotics

Inflate each occupied grid location

Syntax

```
inflate(map,radius)
inflate(map,gridradius,'grid')
```

Description

`inflate(map,radius)` inflates each occupied position of the `map` by the radius given in meters. `radius` is rounded up to the nearest cell equivalent based on the resolution of the map. Every cell within the radius is set to `true` (`1`).

`inflate(map,gridradius,'grid')` inflates each occupied position by the radius given in number of cells.

Input Arguments

`map` — Map representation

`BinaryOccupancyGrid` object

Map representation, specified as a robotics.BinaryOccupancyGrid object. This object represents the environment of the robot. The object contains a matrix grid with binary values indicating obstacles as `true` (`1`) and free locations as `false` (`0`).

`radius` — Dimension the defines how much to inflate occupied locations

scalar

Dimension that defines how much to inflate occupied locations, specified as a scalar. `radius` is rounded up to the nearest cell value.

Data Types: `double`

`gridradius` — Dimension the defines how much to inflate occupied locations
positive scalar

Dimension that defines how much to inflate occupied locations, specified as a positive scalar. `gridradius` is the number of cells to inflate the occupied locations.

Data Types: `double`

See Also

robotics.BinaryOccupancyGrid | robotics.BinaryOccupancyGrid.setOccupancy

Introduced in R2015a

setOccupancy

Class: robotics.BinaryOccupancyGrid
Package: robotics

Set occupancy value for one or more positions

Syntax

```
setOccupancy(map,xy,occval)
setOccupancy(map,ij,occval,'grid')
```

Description

`setOccupancy(map,xy,occval)` assigns occupancy values, `occval`, to the input array of world coordinates, `xy` in the occupancy grid, `map`. Each row of the array, `xy`, is a point in the world and is represented as an `[x y]` coordinate pair. `occval` is either a scalar or a single column array of the same length as `xy` . An occupied location is represented as `true` (1), and a free location is represented as `false` (0).

`setOccupancy(map,ij,occval,'grid')` assigns occupancy values, `occval`, to the input array of grid indices, `ij`, as `[rows cols]`.

Input Arguments

`map` — Map representation

`BinaryOccupancyGrid` object

Map representation, specified as a robotics.BinaryOccupancyGrid object. This object represents the environment of the robot. The object contains a matrix grid with binary values indicating obstacles as `true` (1) and free locations as `false` (0).

`xy` — World coordinates

n-by-2 vertical array

World coordiantes, specified as an *n*-by-2 vertical array of `[x y]` pairs, where *n* is the number of world coordinates.

Data Types: `double`

`ij` — Grid positions
n-by-2 vertical array

Grid positions, specified as an *n*-by-2 vertical array of `[i j]` pairs in `[rows cols]` format, where *n* is the number of grid positions.

Data Types: `double`

`occval` — Occupancy values
n-by-1 vertical array

Occupancy values of the same length as either `xy` or `ij`, returned as an *n*-by-1 vertical array, where *n* is the same *n* in either `xy` or `ij`.

Examples

Set Occupancy Values

Set the occupancy of grid locations using `setOccupancy`.

Initialize an occupancy grid object using `BinaryOccupancyGrid`.

```
map = robotics.BinaryOccupancyGrid(10,10);
```

Set the occupancy of a specific location using `setOccupancy`.

```
setOccupancy(map, [8 8], 1);
```

Set the occupancy of an array of locations.

```
[x,y] = meshgrid(2:5);
setOccupancy(map, [x(:) y(:)],1);
```

See Also

robotics.BinaryOccupancyGrid | robotics.BinaryOccupancyGrid.getOccupancy

Introduced in R2015a

show

Class: robotics.BinaryOccupancyGrid
Package: robotics

Show occupancy grid values

Syntax

```
show(map)
show(map, 'grid')
```

```
show(___,'Parent',parent)
```

```
h = show(map,___)
```

Description

`show(map)` displays the binary occupancy grid `map` in the current axes, with the axes labels representing the world coordinates.

`show(map, 'grid')` displays the binary occupancy grid `map` in the current axes, with the axes labels representing the grid coordinates.

`show(___,'Parent',parent)` sets the specified axes handle `parent` to the axes, using any of the arguments from previous syntaxes.

`h = show(map,___)` returns the figure object handle created by `show`.

Input Arguments

`map` — Map representation
`BinaryOccupancyGrid` object

Map representation, specified as a robotics.BinaryOccupancyGrid object. This object represents the environment of the robot. The object contains a matrix grid with binary values indicating obstacles as `true` (1) and free locations as `false` (0).

`parent` — Axes properties

handle

Axes properties, specified as a handle.

See Also

robotics.BinaryOccupancyGrid

Introduced in R2015a

world2grid

Class: robotics.BinaryOccupancyGrid
Package: robotics

Convert world coordinates to grid indices

Syntax

`ij = world2grid(map,xy)`

Description

`ij = world2grid(map,xy)` converts an array of world coordinates, `xy`, to a `[rows cols]` array of grid indices, `ij`.

Input Arguments

`map` — Map representation

`BinaryOccupancyGrid` object

Map representation, specified as a robotics.BinaryOccupancyGrid object. This object represents the environment of the robot. The object contains a matrix grid with binary values indicating obstacles as `true` (`1`) and free locations as `false` (`0`).

`xy` — World coordinates

n-by-2 vertical array

World coordiantes, specified as an *n*-by-2 vertical array of `[x y]` pairs, where *n* is the number of world coordinates.

Output Arguments

`ij` — Grid positions

n-by-2 vertical array

Grid positions, specified as an n-by-2 vertical array of `[i j]` pairs in `[rows cols]` format, where n is the number of grid positions.

See Also

robotics.BinaryOccupancyGrid | robotics.BinaryOccupancyGrid.grid2world

Introduced in R2015a

findpath

Class: robotics.PRM
Package: robotics

Find path between start and goal points on roadmap

Syntax

```
xy = findpath(prm,start,goal)
```

Description

`xy = findpath(prm,start,goal)` finds an obstacle-free path between `start` and `goal` locations within `prm`, a roadmap object that contains a network of connected points.

If any properties of `prm` change, or if the roadmap is not created, `update` is called.

Input Arguments

`prm` — Roadmap path planner

PRM object

Roadmap path planner, specified as a robotics.PRM object.

`start` — Start location of path

2-by-1 vector

Start location of path, specified as a 2-by-1 vector representing an `[x y]` pair.

Example: [0 0]

`goal` — Final location of path

2-by-1 vector

Final location of path, specified as a 2-by-1 vector representing an `[x y]` pair.

Example: [10 10]

Output Arguments

`xy` — Waypoints for a path between `start` and `goal`

2-by-*n* column vector

Waypoints for a path between start and goal, specified as a 2-by-*n* column vector of `[x y]` pairs, where *n* is the number of waypoints. These pairs represent the solved path from the `start` and `goal` locations, given the roadmap from the `prm` input object.

See Also

robotics.PRM | robotics.PRM.show | robotics.PRM.update

Introduced in R2015a

show

Class: robotics.PRM
Package: robotics

Show map, roadmap, and path

Syntax

```
show(prm)
show(prm,Name,Value)
```

Description

`show(prm)` shows the map and the roadmap, specified as `prm` in a figure window. If no roadmap exists, `update` is called. If a path is computed before calling `show`, the path is also plotted on the figure.

`show(prm,Name,Value)` sets the specified `Value` to the property `Name`.

Input Arguments

`prm` — Roadmap path planner
PRM object

Roadmap path planner, specified as a robotics.PRM object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`'Parent'` — Axes handle
handle

Axes handle that specifies the parent of the figure object created by show, specified as the comma-separated pair consisting of 'Parent' and a handle.

`'Map'` — Map display option

'on' (default) | 'off'

Map display option, specified as the comma-separated pair consisting of `'Map'` and either 'on' or 'off'.

`'Roadmap'` — Roadmap display option

‘on’ (default) | 'off'

Roadmap display option, specified as the comma-separated pair consisting of `'Roadmap'` and either 'on' or 'off'.

`'Path'` — Path display option

‘on’ (default) | 'off'

A string to turn on or off the display of the path, whose value is 'on' or 'off'.

See Also

| |

Related Examples

- “Path Following for a Differential Drive Robot”

Introduced in R2015a

update

Class: robotics.PRM
Package: robotics

Create or update roadmap

Syntax

```
update(prm)
```

Description

`update(prm)` creates a roadmap if called for the first time after creating the `PRM` object, `prm`. Subsequent calls of `update` recreate the roadmap by resampling the map. `update` creates the new roadmap using the `Map`, `NumNodes`, and `ConnectionDistance` property values specified in `prm`.

Input Arguments

`prm` — Roadmap path planner

`PRM` object

Roadmap path planner, specified as a robotics.PRM object.

See Also

robotics.PRM | robotics.PRM.findpath | robotics.PRM.show

Introduced in R2015a

clone

Class: robotics.PurePursuit
Package: robotics

Create PurePursuit object with same property values

Syntax

```
copy = clone(controller)
```

Description

`copy = clone(controller)` creates another instance of the System object, `controller`, with the same property values. If the input object is locked, the `clone` method creates a copy that is also locked and has states initialized to the same values as the original. If the input object is not locked, the `clone` method creates a new unlocked object with uninitialized states.

Input Arguments

`controller` — Pure pursuit controller

`PurePursuit` object

Pure pursuit controller, specified as a `PurePursuit` object.

Output Arguments

`copy` — Pure pursuit controller

`PurePursuit` object

Copy of pure pursuit controller, returned as a `PurePursuit` object.

See Also

robotics.PurePursuit

Introduced in R2015a

isLocked

Class: robotics.PurePursuit
Package: robotics

Check locked states (logical)

Syntax

```
status = isLocked(controller)
```

Description

`status = isLocked(controller)` returns a logical value, `status`, which indicates whether input attributes and nontunable properties are locked for the object, `controller`. For the `PurePursuit` class, the only nontunable property is `Waypoints`.

Input Arguments

`controller` — Pure pursuit controller

`PurePursuit` object

Pure pursuit controller, specified as a `PurePursuit` object.

Output Arguments

`status` — Locked status of object

`Boolean`

Locked status of the object input attributes and nontunable properties, returned as a Boolean.

See Also

robotics.PurePursuit | robotics.PurePursuit.release | robotics.PurePursuit.step

Introduced in R2015a

release

Class: robotics.PurePursuit
Package: robotics

Allow property value changes

Syntax

```
release(controller)
```

Description

`release(controller)` resets the internal properties of the `controller` object and unlocks the object so that you can modify nontunable properties. For the `PurePursuit` class, the only nontunable property is `Waypoints`. After `release` is called, you can change the properties and input characteristics of `controller`.

Input Arguments

`controller` — Pure pursuit controller
`PurePursuit` object

Pure pursuit controller, specified as a `PurePursuit` object.

See Also

robotics.PurePursuit | robotics.PurePursuit.isLocked | robotics.PurePursuit.step

Introduced in R2015a

reset

Class: robotics.PurePursuit
Package: robotics

Reset internal states to default

Syntax

`reset(controller)`

Description

`reset(controller)` resets the internal system properties of the `controller` object. All properties specific to the `PurePursuit` object are kept the same and the locked status of the object does not change.

Input Arguments

`controller` — Pure pursuit controller
`PurePursuit` object

Pure pursuit controller, specified as a `PurePursuit` object.

See Also

robotics.PurePursuit | robotics.PurePursuit.release

Introduced in R2015a

step

Class: robotics.PurePursuit
Package: robotics

Compute linear and angular velocity control commands

Syntax

```
[vel, angvel] = step(controller,pose)
```

Description

`[vel, angvel] = step(controller,pose)` processes the robot's position and orientation, `pose`, as `[x y theta]`, and outputs the linear velocity, `vel`, and angular velocity, `angvel`, based on the specified `controller` .

Input Arguments

`controller` — Pure pursuit controller

`PurePursuit` object

Pure pursuit controller, specified as a `PurePursuit` object.

`pose` — Position and orientation of robot

3-by-1 vector in the form `[x y theta]`

Position and orientation of robot, specified as a 3-by-1 vector in the form `[x y theta]`. The robot's pose is an *x* and *y* position with angular orientation (in radians) measured from the *x*-axis.

Output Arguments

`vel` — Linear velocity

scalar in meters per second

Linear velocity, specifed as a scalar in meters per second.

Data Types: `double`

`angvel` — Angular velocity

scalar in radians per second

Angular velocity, specified as a scalar in radians per second.

Data Types: `double`

See Also

robotics.PurePursuit

Introduced in R2015a

robotics.VectorFieldHistogram.clone

Class: robotics.VectorFieldHistogram
Package: robotics

Create `VectorFieldHistogram` object with same property values

Syntax

```
vfhCopy = clone(vfh)
```

Description

`vfhCopy = clone(vfh)` creates another instance of the `VectorFieldHistogram` object with the same properties.

Input Arguments

`vfh` — Vector field histogram algorithm

`VectorFieldHistogram` object

Vector field histogram algorithm, specified as a `VectorFieldHistogram` object. This object contains all the parameters for tuning the VFH+ algorithm.

Output Arguments

`vfhCopy` — Vector field histogram algorithm

`VectorFieldHistogram` object

Vector field histogram algorithm, returned as a `VectorFieldHistogram` object. This object contains all the parameters for tuning the VFH+ algorithm.

Examples

Copy VectorFieldHistogram Object

Create a`VectorFieldHistogram` object.

```
vfh = robotics.VectorFieldHistogram
```

```
vfh = 

  System: robotics.VectorFieldHistogram

  Properties:
          NumAngularSectors: 180
             DistanceLimits: [0.05 2]
                RobotRadius: 0.1
             SafetyDistance: 0.1
           MinTurningRadius: 0.1
      TargetDirectionWeight: 5
     CurrentDirectionWeight: 2
    PreviousDirectionWeight: 2
        HistogramThresholds: [3 10]
```

Create a copy with the same properties.

```
vfh2 = clone(vfh)
```

```
vfh2 = 

  System: robotics.VectorFieldHistogram

  Properties:
          NumAngularSectors: 180
             DistanceLimits: [0.05 2]
                RobotRadius: 0.1
             SafetyDistance: 0.1
           MinTurningRadius: 0.1
      TargetDirectionWeight: 5
     CurrentDirectionWeight: 2
    PreviousDirectionWeight: 2
```

```
    HistogramThresholds: [3 10]
```

See Also

robotics.VectorFieldHistogram | robotics.VectorFieldHistogram.reset | robotics.VectorFieldHistogram.show

Introduced in R2015b

robotics.VectorFieldHistograms.reset

Reset internal states to default

Syntax

```
reset(vfh)
```

Description

`reset(vfh)` resets the internal states of the `VectorFieldHistogram` object to their initial values. All properties specific to the object are kept the same.

Input Arguments

`vfh` — Vector field histogram algorithm
`VectorFieldHistogram` object

Vector field histogram algorithm, specified as a `VectorFieldHistogram` object. This object contains all the parameters for tuning the VFH+ algorithm.

Examples

Reset `VectorFieldHistogram` Object

```
reset(vfh)
```

See Also

robotics.VectorFieldHistogram | robotics.VectorFieldHistogram.clone | robotics.VectorFieldHistogram.step

Introduced in R2015b

robotics.VectorFieldHistogram.show

Class: robotics.VectorFieldHistogram
Package: robotics

Display `VectorFieldHistogram` information in figure window

Syntax

`show(vfh)`

`show(vfh,'Parent',parent)`

`h = show( ___ )`

Description

`show(vfh)` shows histograms calculated by the VFH+ algorithm in a figure window. The figure also includes the parameters of the `VectorFieldHistrogram` object and range values from the last step input.

`show(vfh,'Parent',parent)` sets the specified axes handle, `parent`, to the axes.

`h = show( ___ )` returns the figure object handle created by `show` using any of the arguments from the previous syntaxes.

Examples

Show Sample Data from `VectorFieldHistogram` Object

Create a vector field histogram object.

```
vfh = robotics.VectorFieldHistogram;
targetDir = 0;
```

Create sample laser scan data input.

```
ranges = 10*ones(1,300);
```

```
ranges(1,110:150) = 1.0;
angles = linspace(-pi/2,pi/2,300);
```

Compute an obstacle-free steering direction.

```
steeringDir = step(vfh,ranges,angles,targetDir);
```

Show the data from the `VectorFieldHistogram` object.

```
show(vfh);
```

Input Arguments

`vfh` — Vector field histogram algorithm

`VectorFieldHistogram` object

Vector field histogram algorithm, specified as a `VectorFieldHistogram` object. This object contains all the parameters for tuning the VFH+ algorithm.

`parent` — Axes properties

handle

Axes properties, specified as a handle.

Output Arguments

`h` — Axes handles for VFH algorithm display

`Axes` array

Axes handles for VFH algorithm display, specified as an `Axes` array. The VFH histogram and `HistogramThresholds` are shown in the first axes. The binary histogram, range sensor readings, target direction, and steering directions are shown in the second axes.

See Also

robotics.VectorFieldHistogram | robotics.VectorFieldHistogram.step

Introduced in R2015b

robotics.VectorFieldHistogram.step

Class: robotics.VectorFieldHistogram
Package: robotics

Find obstacle-free steering direction

Syntax

```
steeringDir = step(vfh,ranges,angles,targetDir)
```

Description

`steeringDir = step(vfh,ranges,angles,targetDir)` finds an obstacle-free steering direction using the VFH+ algorithm for input vectors, `ranges` and `angles`. The algorithm parameters are defined in the `vfh` object. A target direction is given based on the target location.

Input Arguments

`vfh` — Vector field histogram algorithm

`VectorFieldHistogram` object

Vector field histogram algorithm, specified as a `VectorFieldHistogram` object. This object contains all the parameters for tuning the VFH+ algorithm.

`ranges` — Range values from scan data

vector

Range values from scan data, specified as a vector in meters. These range values are distances from a sensor at given `angles`. The vector must be the same length as the corresponding `angles` vector.

`angles` — Angle values from scan data

vector

Angle values from scan data, specified as a vector in radians. These angle values are the specific angles of the given `ranges`. The vector must be the same length as the corresponding `ranges` vector.

`targetDir` — Target direction for robot
scalar

Target direction for the robot, specified as a scalar in radians. The forward direction of the robot is considered zero radians, with positive angles measured counterclockwise.

Output Arguments

`steeringDir` — Steering direction for robot
scalar

Steering direction for the robot, specified as a scalar in radians. This obstacle-free direction is calculated based on the VFH+ algorithm. The forward direction of the robot is considered zero radians, with positive angles measured counterclockwise.

Examples

Find Steering Direction of Vector Field Histogram

Create a vector field histogram object.

```
vfh = robotics.VectorFieldHistogram;
targetDir = 0;
```

Create sample laser scan data input.

```
ranges = 10*ones(1, 300);
ranges(1, 70:150) = 1.0;
angles = linspace(-pi/2, pi/2, 300);
```

Compute an obstacle-free steering direction.

```
steeringDir = step(vfh,ranges,angles, targetDir);
```

Show the data from the `VectorFieldHistogram` object.

```
show(vfh);
```

See Also

robotics.VectorFieldHistogram | robotics.VectorFieldHistogram.reset | robotics.VectorFieldHistogram.show

Introduced in R2015b

Blocks — Alphabetical List

Blank Message

Create blank message using specified message type

Library

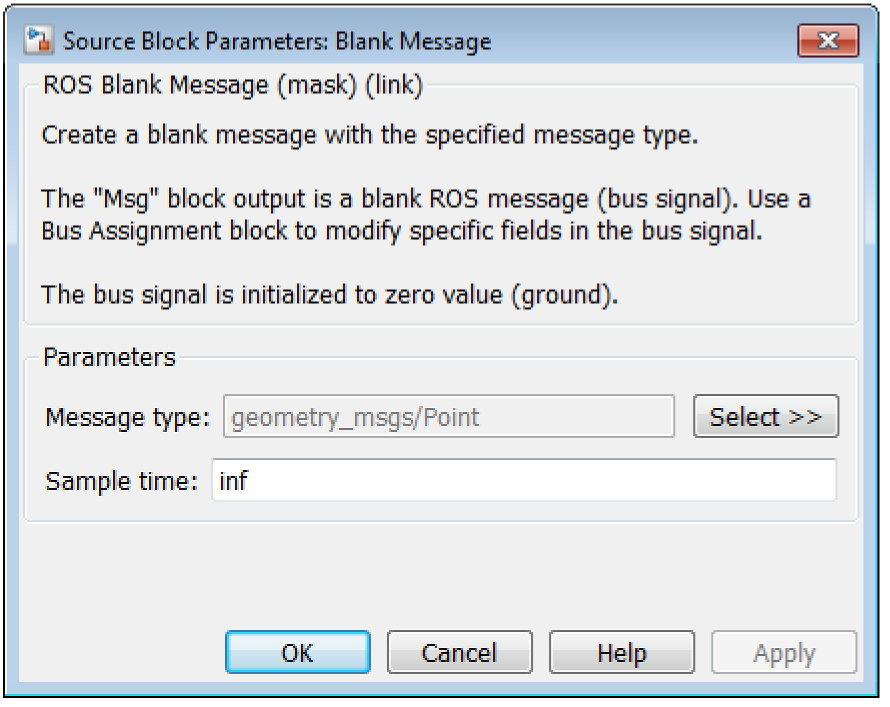
Robotics System Toolbox

`robotlib`

Description

The Blank Message block creates a Simulink® nonvirtual bus corresponding to the selected ROS message type. On each sample hit, the block emits a blank or "zero" signal for the designated message type. All elements of the bus are initialized to 0, and the length of the variable-length arrays are initialized to 0 as well.

Dialog Box



Message type

Message type for the blank message. Use the **Select** button to select from a full list of supported ROS messages. You can also use the `rostype` function in MATLAB to view the list of supported ROS messages.

Sample time

Interval between times that the Blank Message block output can change during simulation.

Default: `inf`

This default value indicates that the block output can never change. Using this value speeds simulation and code generation by eliminating the need to recompute the block output.

For more information, see “ Specify Sample Time”.

See Also

`Publish` | `Subscribe`

Related Examples

- “Virtual and Nonvirtual Buses”

Introduced in R2015a

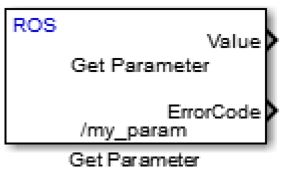
Get Parameter

Get values from ROS parameter server

Library

Robotics System Toolbox

`robotlib`



Description

The Get Parameter block outputs the value of the specified ROS parameter. The block uses the ROS node of the Simulink model to connect to the ROS network. This node is created when you run the model and is deleted when the model terminates. If the model does not have a node, the block creates one.

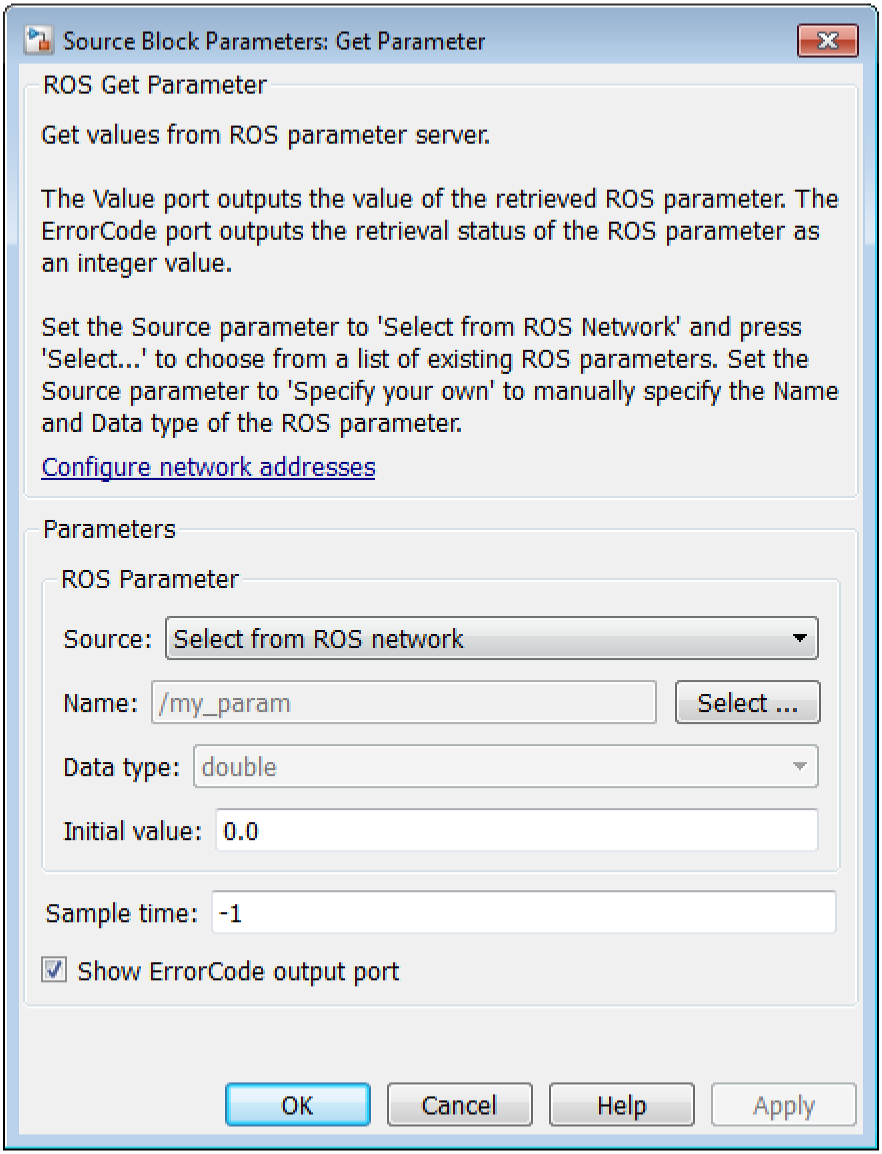
On each sample hit, the block checks the ROS parameter server for the specified ROS parameter and outputs its value.

The `Value` output shows the parameter value from the parameter server.

The `ErrorCode` output is a numerical value that indicates the status of the ROS parameter:

- **0** — ROS parameter retrieved successfully. The retrieved value is output in `Value`.
- **1** — No ROS parameter with specified name found. If there is no known value, `Value` is set to the last received value or to **Initial value**.
- **2** — ROS parameter retrieved, but its type is different than the specified **Data type**. If there is no known value, `Value` is set to the last received value or to **Initial value**.

Dialog Box



Source

Source of the parameter name that you want to access.

- `Select from ROS network` — Use **Select** to select a parameter name. The **Data type** parameter is set automatically. You must be connected to a ROS network.
- `Specify your own` — Enter a parameter name in **Name** and specify its data type in **Data type**. You must match a parameter name exactly.

Name

The name of the parameter to get from the ROS parameter server. When **Source** is set to `Select from ROS network`, use **Select** to select an existing parameter. You must be connected to a ROS network to get a list of parameters. Otherwise, specify the parameter and data type.

Parameter name strings must follow the rules of ROS graph names. Valid names have these characteristics:

- The first character is an alpha character ([a-z|A-Z]), tilde (~), or forward slash (/).
- Subsequent characters are alphanumeric ([0-9|a-z|A-Z]), underscores(_), or forward slashes (/).

Data type

Data type of your parameter.

- `double`
- `int32`
- `boolean`

If you set **Source** to `Select from ROS network`, then **Data type** is disabled and the data type is selected automatically.

Initial value

Output of `Value` when an error occurs and no valid value has been received from the parameter server. The data type must match the specified **Data type**.

Sample time

Interval between times that the Get Parameter block output can change during simulation. In simulation, the sample time follows simulation time and not actual wall-block time.

Default: `-1`

This default value indicates that the block sample time is *inherited*. See “ Specify Sample Time”.

Show `ErrorCode` output port

Select this check box to enable `ErrorCode` output. When you clear this check box, the `ErrorCode` output is removed from the block.

See Also

`Set Parameter`

External Websites

- ROS Parameter Server
- ROS Graph Names

Introduced in R2015b

Publish

Send messages to ROS network

Library

Robotics System Toolbox

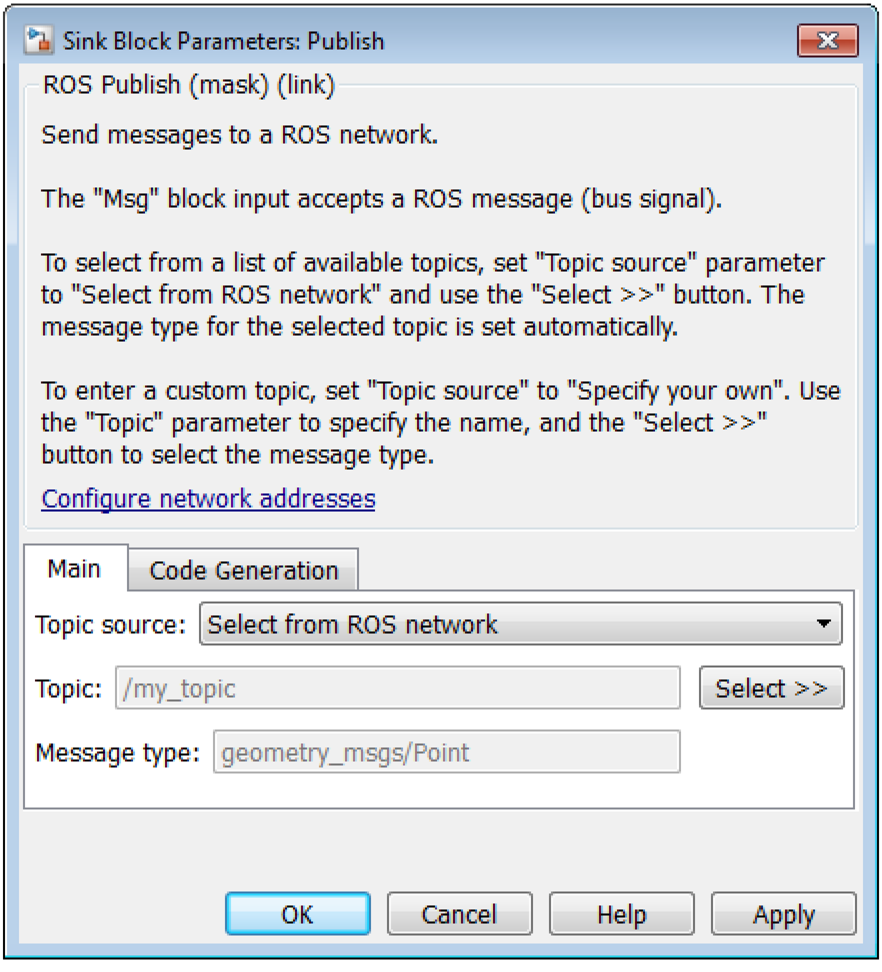
`robotlib`

Description

The Publish block takes in as its input a Simulink nonvirtual bus that corresponds to the specified ROS message type and publishes it to the ROS network. It uses the node of the Simulink model to create a ROS publisher for a specific topic. This node is created when the model runs and is deleted when the model terminates. If the model does not have a node, the block creates one.

On each sample hit, the block converts the `Msg` input from a Simulink bus signal to a ROS message and publishes it. The block does not distinguish whether the input is a new message but merely publishes it on every sample hit. For simulation, this input is a MATLAB ROS message and in code generation, it is a C++ ROS message.

Dialog Box



Topic source

This selector determines where you get the topic name that you want to subscribe to.

- `Select from ROS network` — Use the **Select** button to select a topic. You must be connected to a ROS network.

- `Specify your own` — Enter a topic name in **Topic**. You must match a topic name exactly.

Topic

The ROS topic to publish to, specified as a string. When **Topic source** is set to `Select from ROS network`, use the**Select** button to select from the ROS network. You must be connected to a ROS network to get a list of topics. Otherwise, specify the topic you want.

Topic name strings must follow the rules of ROS topic names. Valid names have the following characteristics:

- The first character is an alpha character ([a-z|A-Z]), tilde (~), or forward slash (/).
- Subsequent characters are alphanumeric ([a-z|A-Z]), underscores(_), or forward slashes (/).

Message type

Message type for the **Topic** specified. If you select a topic from the ROS network, the message type is selected for you. Otherwise, use **Select** button to select from a full list of supported ROS messages. You can also use the `rostype` function in MATLAB to view the list of messages.

Tips

You can also set the addresses for the ROS master and node host by clicking the **Configure network addresses** link in the dialog box.

See Also

`Blank Message` | `Subscribe`

Related Examples

- "Virtual and Nonvirtual Buses"

More About

- "Simulink and ROS Interaction"

Introduced in R2015a

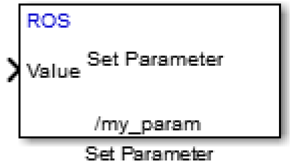
Set Parameter

Set values on ROS parameter server

Library

Robotics System Toolbox

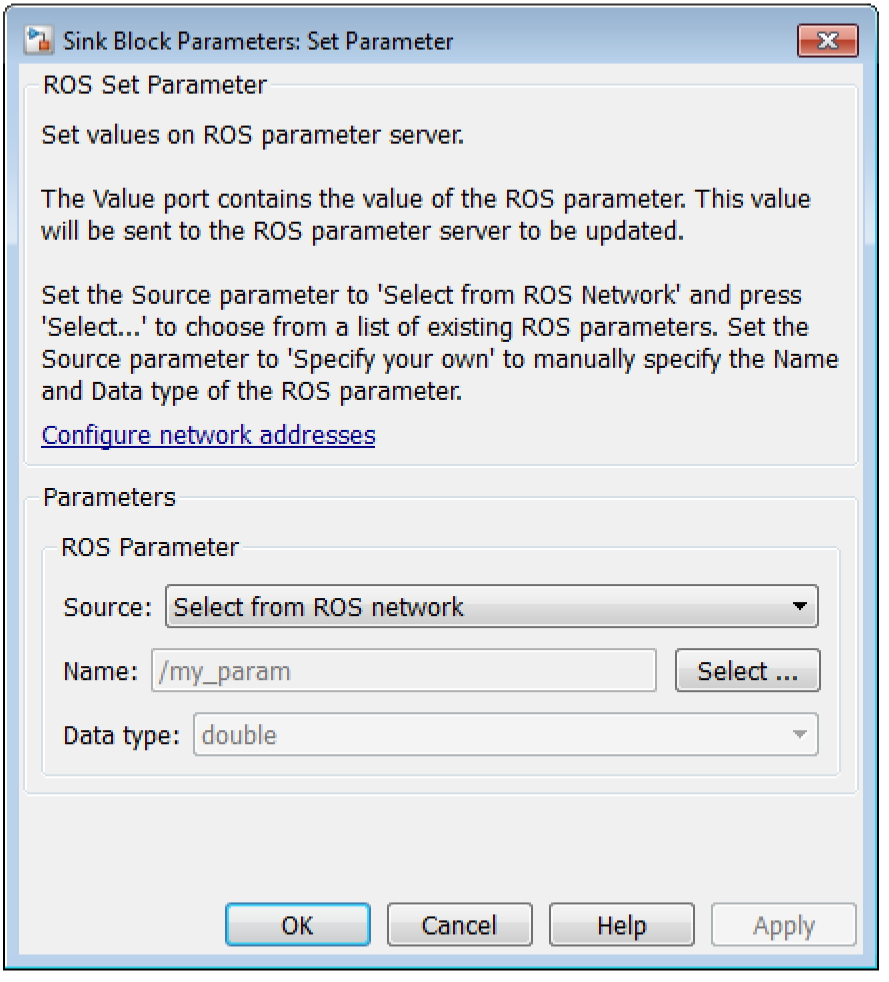
`robotlib`



Description

The Set Parameter block sets the `Value` input to the specified name on the ROS parameter server. The block uses the ROS node of the Simulink model to connect to the ROS network. This node is created when you run the model and is deleted when the model terminates. If the model does not have a node, the block creates one.

Dialog Box



Source

Source of the parameter name that you want to access.

- `Select from ROS network` — Use **Select** to select a parameter name. The **Data type** parameter is set automatically. You must be connected to a ROS network.

- `Specify your own` — Enter a parameter name in **Name** and specify its data type in **Data type**. You must match a parameter name exactly.

Name

The name of the parameter to get from the ROS parameter server. When **Source** is set to `Select from ROS network`, use **Select** to select an existing parameter. You must be connected to a ROS network to get a list of parameters. Otherwise, specify the parameter and data type.

Parameter name strings must follow the rules of ROS graph names. Valid names have these characteristics:

- The first character is an alpha character ([a-z|A-Z]), tilde (~), or forward slash (/).
- Subsequent characters are alphanumeric ([0-9|a-z|A-Z]), underscores(_), or forward slashes (/).

Data type

Data type of your parameter.

- `double`
- `int32`
- `boolean`

If you set **Source** to `Select from ROS network`, then **Data type** is disabled and the data type is selected automatically.

See Also

`Get Parameter`

External Websites

- ROS Parameter Servers
- ROS Graph Names

Introduced in R2015b

Subscribe

Receive messages from ROS network

Library

Robotics System Toolbox

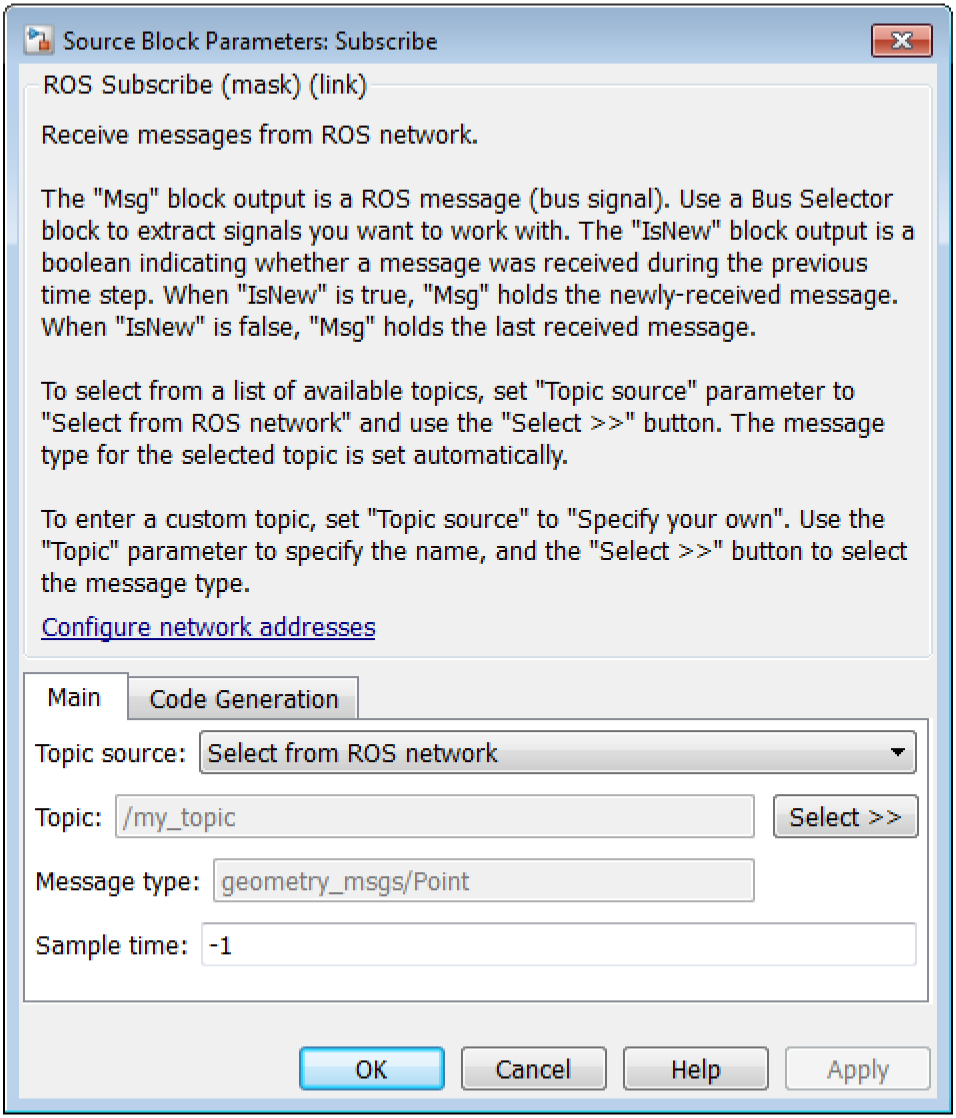
`robotlib`

Description

Subscribe creates a Simulink nonvirtual bus that corresponds to the specified ROS message type. The block uses the node of the Simulink model to create a ROS subscriber for a specific topic. This node is created when the model runs and is deleted when the model terminates. If the model does not have a node, the block creates one.

On each sample hit, the block checks if a new message available on the specific topic. If a new message is available, the block retrieves the message and converts it to a Simulink bus signal. The `Msg` outputs this new message. If a new message is not available, `Msg` outputs the last received ROS message. If there has not been a received message since the start of the simulation, `Msg` outputs a blank message.

Dialog Box



Topic source

This selector determines where you get the topic name that you want to subscribe to.

- `Select from ROS network` — Use the **Select** button to select a topic. You must be connected to a ROS network.
- `Specify your own` — Enter a topic name in **Topic**. You must match a topic name exactly.

Topic

The ROS topic to publish to, specified as a string. When **Topic source** is set to `Select from ROS network`, use the**Select** button to select from the ROS network. You must be connected to a ROS network to get a list of topics. Otherwise, specify the topic you want.

Topic name strings must follow the rules of ROS topic names. Valid names have the following characteristics:

- The first character is an alpha character ([a-z|A-Z]), tilde (~), or forward slash (/).
- Subsequent characters are alphanumeric ([a-z|A-Z]), underscores(_), or forward slashes (/).

Message type

Message type for the **Topic** specified. If you select a topic from the ROS network, the message type is selected for you. Otherwise, use **Select** button to select from a full list of supported ROS messages. You can also use the `rostype` function in MATLAB to view the list of messages.

Sample time

Interval between times that the Subscribe block output can change during simulation. In simulation, the sample time follows simulation time and not actual wall-block time.

Default: `-1`

This default value indicates that the block sample time is *inherited*.

For more information about the *inherited* sample time type, see " Specify Sample Time".

Tips

You can also *Configure Network Addresses* by clicking the link in the dialog box. This allows you to set the addresses for the 'ROS Master' and 'Node Host'.

See Also

`Blank Message` | `Publish`

Related Examples

- “Virtual and Nonvirtual Buses”

More About

- “Simulink and ROS Interaction”

Introduced in R2015a