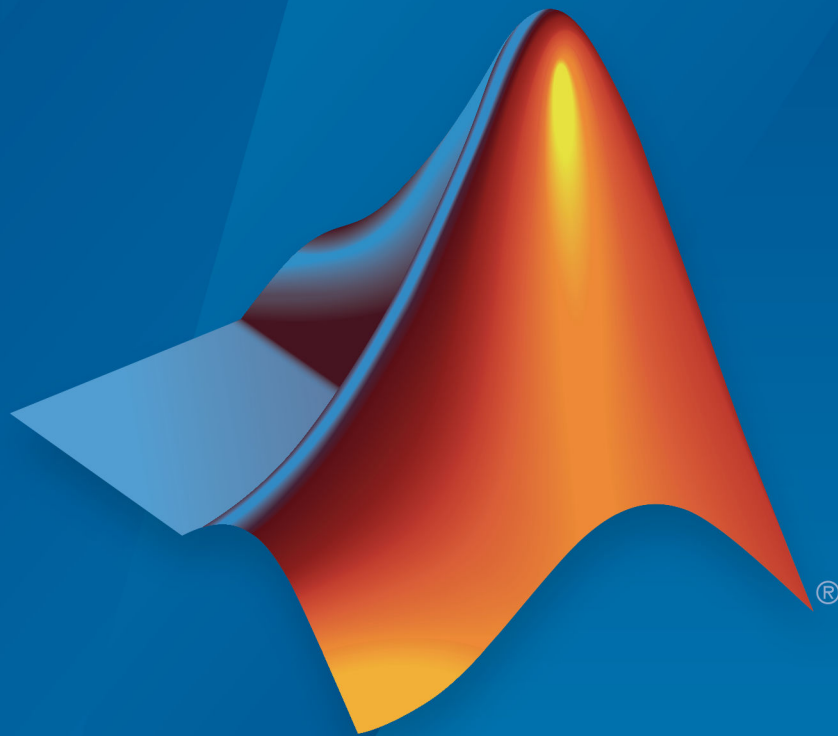


Sensor Fusion and Tracking Toolbox™

User's Guide



MATLAB® & SIMULINK®

R2018b



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

Sensor Fusion and Tracking Toolbox™ User's Guide

© COPYRIGHT 2018 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

September 2018 Online only

New for Version 1.0 (Release 2018b)

Tracking Scenarios

1

Tracking Simulation Overview	1-2
Creating a tracking scenario	1-4
Create Tracking Scenario with Two Platforms	1-9

Radar Detections

2

Simulate Radar Detections	2-2
Create Radar Sensor	2-2
Detector Input	2-16
Radar Sensor Coordinate Systems	2-18
INS	2-20
Detections	2-20

Multi-Object Tracking

3

Tracking and Tracking Filters	3-2
Multi-Object Tracking	3-2
Multi-Object Tracker Properties	3-3
Linear Kalman Filters	3-10
State Equations	3-10
Measurement Models	3-12

Linear Kalman Filter Equations	3-12
Filter Loop	3-13
Constant Velocity Model	3-15
Constant Acceleration Model	3-16
Extended Kalman Filters	3-18
State Update Model	3-18
Measurement Model	3-19
Extended Kalman Filter Loop	3-19
Predefined Extended Kalman Filter Functions	3-20

Data Structures

4

Target Pose	4-2
Platform Pose	4-4
Platform Profiles	4-6
Object Detections	4-7
Measurements and Measurement Noise	4-7
Measurement Parameters	4-8
Object Attributes	4-10
Signal Structure	4-11
INS	4-12
Sensor Configuration	4-13
Emitter Configuration	4-14

Tracking Scenarios

Tracking Simulation Overview

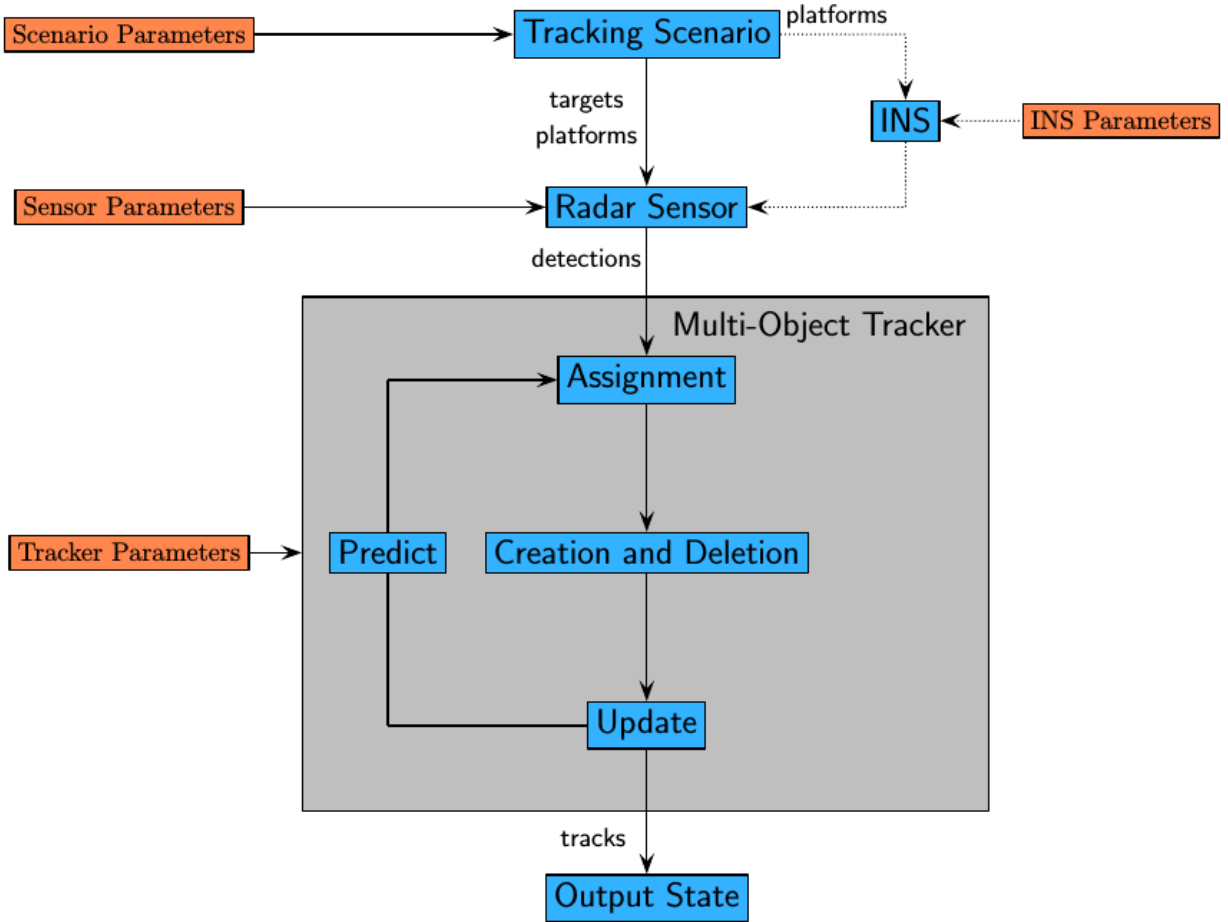
You can build a complete tracking simulation using the functions and objects supplied in this toolbox. The workflow for sensor fusion and tracking simulation consists of three (and optionally four) components. These components are

- 1 Use the tracking scenario generator to create ground truth for all moving and stationary radar platforms and all target platforms (planes, ships, cars, drones). The `trackingScenario` class models the motion of all platforms in a global coordinate system called scenario coordinates. These objects can represent ships, ground vehicles, airframes, or any object that the radar detects. See “Orientation, Position, and Coordinate Systems” for a discussion of coordinate systems.
- 2 Optionally, simulate an inertial navigation system (INS) that provides radar sensor platform position, velocity, and orientation relative to scenario coordinates.
- 3 Create models for each radar sensor with specifications and parameters using the `monostaticRadarSensor`, `radarSensor`, or `radarEmitter` objects. Using target platform pose and profile information, generate synthetic radar detections for each radar-target combination. Methods belonging to `trackingScenario` retrieve the pose and profile of any target platform. The `trackingScenario` generator does not have knowledge of scenario coordinates. It knows the relative positions of the target platforms with respect to the body platform of the radar. Therefore, the detector can only generate detections relative to the radar location and orientation.

If there is an INS attached to a radar platform, then the radar can transform detections to the scenario coordinate system. The INS allows multiple radars to report detections in a common coordinate system.

- 4 Process radar detections with a multi-object tracker to associate detections to existing tracks or create tracks. Multi-object tracks include `trackerGNN` and `trackerTOMHT`. If there is no INS, the tracker can only generate tracks specific to one radar. If an INS is present, the tracker can create tracks using measurements from all radars.

The flow diagram shows the progression of information in a tracking simulation.



Creating a tracking scenario

You can define a tracking simulation by using the `trackingScenario` object. This object creates an empty scenario which you can then populate with platforms by calling the `platform` method as many times as needed. A platform is an object (moving or stationary) which can either be a sensor, a target, or other entity. After creating a platform, you can use the `platform path` method to create paths that specify how the platform moves. You construct paths by specifying three-dimensional waypoints and the arrival time of the platform at the waypoint. `path` creates a three-dimensional piecewise-clothoid curve that the platform follows. Then, run the simulation by calling the `advance` method in a loop, or by calling the `record` method to run the simulation all at once. You can set the simulation update interval using the `SampleTime` property.

When you create a platform, you set its properties or leave them to their default value. You can set them all except for `PlatformID`. The orientation of a platform can be specified in three equivalent ways. The `Roll`, `Pitch`, and `Yaw`, the `OrientationQuaternion`, and `OrientationMatrix` can all be used to specify the orientation of the platform. They must all be consistent. If you change one, the others are automatically changed. If you specify more than one when creating a platform, the last value entered is used. Platform properties and their default values are listed here.

Platform Properties

PlatformID	Scenario-defined platform ID.
Position	Position of platform, specified as a real-valued 1-by-3 vector representing the (x,y,z) coordinates of the platform referenced to the scenario coordinate system. The default is $[0, 0, 0]$. Units are in meters.
Velocity	Velocity of platform, specified as a real-valued 1-by-3 vector representing the (vx,vy,vz) velocity components referenced to the scenario coordinate system. The default is $[0, 0, 0]$. Units are in m/s.
Roll	Roll helps define the orientation of the platform body with respect to the scenario coordinate system. Roll is the clockwise rotation about the x body axes. If you do not enter a value for roll, roll has the default value of zero. You specify roll in degrees.
Pitch	Pitch helps define the orientation of the platform body with respect to the scenario coordinate system. Pitch is the clockwise rotation about the y body axes. If you do not enter a value for pitch, pitch has the default value of zero. You specify pitch in degrees.
Yaw	Yaw helps define the orientation of the platform body with respect to the scenario coordinate system. Yaw is the clockwise rotation about the z body axes. If you do not enter a value for yaw, yaw has the default value of zero. You specify yaw in degrees.
OrientationQuaternion	The orientation of the body expressed in quaternions.

OrientationMatrix	3-by-3 matrix specifying the platforms body coordinate vector with respect to scenario coordinates. The columns represent the directions of the carried x, y, and z coordinates of the body.
AngularVelocity	Angular velocity of platform, specified as a real-valued 1-by-3 vector specifying the x,y, and z components of the angular velocity referenced to scenario coordinates. Units are in degrees per second.
RCSPattern	The radar cross-section of a platform defines the response of the platform to radar signals. The values are specified at discrete azimuth and elevation angles. These angles are referenced to the platform body coordinates.
RCSAzimuthAngles	Azimuth angles of the entries in RCSPattern.
RCSAzimuthAngles	Elevation angles of the entries in RCSPattern.
ClassID	User-specified platform classification ID.

The input to the path method is the set of name-value parameters listed here.

Path Parameters

Waypoints	An M -by-3 matrix containing the x, y, and z positions in scenario coordinates through which the platform must pass. Units are in meters.
TimeOfArrival	An M -element vector that specifies the time at which the target crosses the corresponding waypoint. Units are in seconds.
Velocities	An M -by-3 matrix containing the x, y, and z components of the velocity in scenario coordinates at the corresponding waypoint. Units are in m/s.
Groundspeed	An M -element vector that specifies the ground speed at the corresponding waypoint. Units are in m/s.
ClimbRate	An M -element vector that specifies the rate of climb (in the z dimension). Units are in m/s.
Course	An M -element vector that specifies to the direction in which the target is moving in the horizontal plane. Course is specified as an angle measured from the x-axis to the y-axis in the xy-plane.
Orientation	An M -element vector of quaternions or a 3-by-3-by- M matrix of rotation matrices at each waypoint. These quantities correspond to the rotational orientation of the platform at that waypoint. If unspecified, the platform yaw and pitch angles are aligned in the direction of travel. Then, the Banking parameter roll determines the roll angle.

Banking	<p>Specified as either 'flat' or 'frictionless'.</p> <ul style="list-style-type: none">• If 'flat', the platform roll angle is zero.• If 'frictionless', the roll angle of the platform is chosen to balance the accelerations due to the trajectory path and the acceleration due to gravity. <p>The banking Banking property cannot be used together with Orientation. If you do not specify Banking or Orientation, the platform roll angle is zero.</p>
---------	--

At any time during the simulation, you can retrieve the current values of platform properties using the `platformPoses` and `platformProfiles` methods of the `trackingScenario` class. You can also use the `targetPoses` method of the `Platform` class. Both the `platformPoses` and `platformProfiles` methods return properties of all platforms with respect to scenario coordinates. The `targetPoses` method, while similar, returns properties of other platforms with respect to a specified platform.

Create Tracking Scenario with Two Platforms

Construct a tracking scenario with two platforms following different trajectories.

```
sc = trackingScenario('UpdateRate',100.0,'StopTime',1.2);
```

Create two platforms.

```
platfm1 = platform(sc);
platfm2 = platform(sc);
```

Platform 1 follows a circular path of radius 10 m for one second. This is accomplished by placing waypoints in a circular shape, ensuring that the first and last waypoint are the same.

```
wpts1 = [0 10 0; 10 0 0; 0 -10 0; -10 0 0; 0 10 0];
time1 = [0; 0.25; .5; .75; 1.0];
platfm1.Trajectory = waypointTrajectory(wpts1, time1);
```

Platform 2 follows a straight path for one second.

```
wpts2 = [-8 -8 0; 10 10 0];
time2 = [0; 1.0];
platfm2.Trajectory = waypointTrajectory(wpts2,time2);
```

Verify the number of platforms in the scenario.

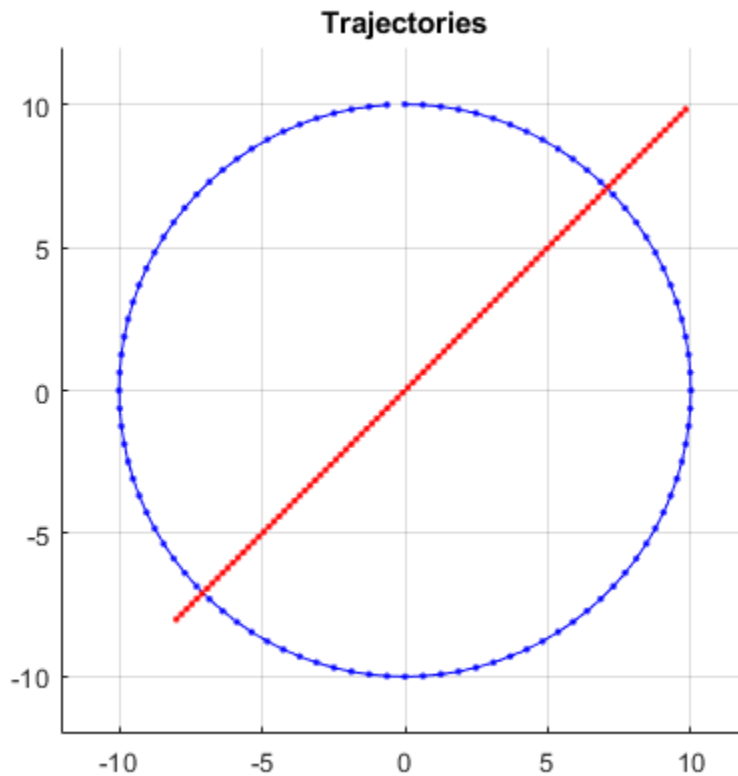
```
disp(sc.Platforms)

    [1x1 fusion.scenario.Platform]    [1x1 fusion.scenario.Platform]
```

Run the simulation and plot the current position of each platform. Use an animated line to plot the position of each platform

```
figure
grid
axis equal
axis([-12 12 -12 12])
line1 = animatedline('DisplayName','Trajectory 1','Color','b','Marker','.');
line2 = animatedline('DisplayName','Trajectory 2','Color','r','Marker','.');
title('Trajectories')
p1 = pose(platfm1);
p2 = pose(platfm2);
addpoints(line1,p1.Position(1),p1.Position(2));
```

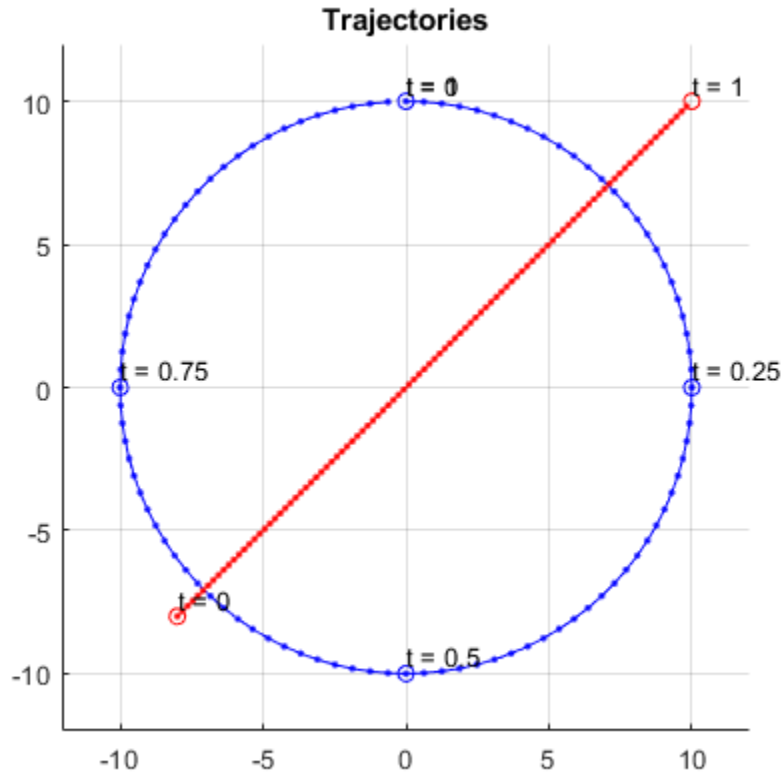
```
addpoints(line2,p2.Position(2),p2.Position(2));  
  
while advance(sc)  
    p1 = pose(platfm1);  
    p2 = pose(platfm2);  
    addpoints(line1,p1.Position(1),p1.Position(2));  
    addpoints(line2,p2.Position(2),p2.Position(2));  
    pause(0.1)  
end
```



Plot the waypoints for both platforms.

```
hold on  
plot(wpts1(:,1),wpts1(:,2),'ob')
```

```
text(wpts1(:,1),wpts1(:,2),"t = " + string(time1),'HorizontalAlignment','left','VerticalAlignment','top')
plot(wpts2(:,1),wpts2(:,2),'or')
text(wpts2(:,1),wpts2(:,2),"t = " + string(time2),'HorizontalAlignment','left','VerticalAlignment','top')
hold off
```



Radar Detections

The radar detectors `monostaticRadarSensor` and `radarSensor` generate measurements from target poses.

Simulate Radar Detections

The `monostaticRadarSensor` object simulates the detection of targets by a scanning radar. You can use the object to model many properties of real radar sensors. For example, you can

- simulate real detections with added random noise
- generate false alarms
- simulate mechanically scanned antennas and electronically scanned phased arrays
- specify angular, range, and range-rate resolution and limits

The radar sensor is assumed to be mounted on a platform and carried by the platform as it maneuvers. A platform can carry multiple sensors. When you create a sensor, you specify sensor positions and orientations with respect to the body coordinate system of a platform. Each call to `monostaticRadarSensor` creates a sensor. The output of `monostaticRadarSensor` generates the detection input to the multi-object tracker, `gnnTracker`, or any simple one-object tracker such as `trackingKF`, `trackingEKF`, `trackingUKF`, and `trackingCKF`.

The radar platform does not maintain any information about the radar sensors that are mounted on it. (The sensor itself contains its position and orientation with respect to the platform on which it is mounted but not which platform). You must create the association between radar sensors and platforms. A way to do this association is to put the platform and its associated sensors into a cell array. When you call a particular sensor, pass in the platform-centric target pose and target profile information. The sensor converts this information to sensor-centric poses. Target poses are outputs of `trackingScenario` methods.

Create Radar Sensor

You can create a radar sensor using the `monostaticRadarSensor` object. Set the radar properties using name-value pairs and then execute the simulator. For example,

```
radar1 = monostaticRadarSensor( ...  
    'UpdateRate',updaterate, ...           % Hz  
    'ReferenceRange', 111.0e3, ...         % m  
    'ReferenceRCS', 0.0, ...               % dBsm  
    'HasMechanicalScan',true, ...  
    'MaxMechanicalScanRate',scanrate, ... % deg/s  
    'HasElectronicScan',false, ...
```

```

    'FieldOfView', fov, ...           % [az;el] deg
    'HasElevation', false, ...
    'HasRangeRate', false, ...
    'AzimuthResolution', 1.4, ...    % deg
    'RangeResolution', 135.0)        % m
dets = radar1(targets, simtime);

```

Convenience Syntaxes

There are several syntaxes of `monostaticRadarSensor` that make it easier to specify the properties of commonly implemented radar scan modes. These syntaxes set combinations of these properties: `ScanMode`, `FieldOfView`, `MaxMechanicalScanRate`, `MechanicalScanLimits`, and `ElectronicScanLimits`.

- `sensor = monostaticRadarSensor('Rotator')` creates a `monostaticRadarSensor` object that mechanically scans 360° in azimuth. Setting `HasElevation` to `true` points the radar antenna towards the center of the elevation field of view.
- `sensor = monostaticRadarSensor('Sector')` creates a `monostaticRadarSensor` object that mechanically scans a 90° azimuth sector. Setting `HasElevation` to `true`, points the radar antenna towards the center of the elevation field of view. You can change the `ScanMode` to `'Electronic'` to electronically scan the same azimuth sector. In this case, the antenna is not mechanically tilted in an electronic sector scan. Instead, beams are stacked electronically to process the entire elevation spanned by the scan limits in a single dwell.
- `sensor = monostaticRadarSensor('Raster')` returns a `monostaticRadarSensor` object that mechanically scans a raster pattern spanning 90° in azimuth and 10° in elevation upwards from the horizon. You can change the `ScanMode` property to `'Electronic'` to perform an electronic raster scan in the same volume.
- `sensor = monostaticRadarSensor('No scanning')` returns a `monostaticRadarSensor` object that stares along the radar antenna boresight direction. No mechanical or electronic scanning is performed.

You can set other radar properties when you use these syntaxes. For example,

```
sensor = monostaticRadarSensor('Raster', 'ScanMode', 'Electronic')
```

Radar Sensor Parameters

The properties specific to the `monostaticRadarSensor` object are listed here. For more detailed information, type

```
help monostaticRadarSensor
```

at the command line.

Sensor location parameters.

Sensor Location

SensorIndex	A unique identifier for each sensor.
UpdateRate	Rate at which sensor updates are generated, specified as a positive scalar. The reciprocal of this property must be an integer multiple of the simulation time interval. Updates requested between sensor update intervals do not return detections.
MountingLocation	Sensor (x,y,z) defining the offset of the sensor origin from the origin of its platform. The default value positions the sensor origin at the platform origin.
Yaw	Angle specifying the rotation around the platform z-axis to align the platform coordinate system with the sensor coordinate system. Positive yaw angles correspond to a clockwise rotation when looking along the positive direction of the z-axis of the platform coordinate system. Rotations are applied using the ZYX convention.
Pitch	Angle specifying the rotation around the platform y-axis to align the platform coordinate system with the sensor coordinate system. Positive pitch angles correspond to a clockwise rotation when looking along the positive direction of the y-axis of the platform coordinate system. Rotations are applied using the ZYX convention.

Roll	Angle specifying the rotation around the platform x-axis to align the platform coordinate system with the sensor coordinate system. Positive pitch angles correspond to a clockwise rotation when looking along the positive direction of the x-axis of the platform coordinate system. Rotations are applied using the ZYX convention.
DetectionCoordinates	<p>Specifies the coordinate system for detections reported in the "Detections" on page 2-20 output struct. The coordinate system can be one of:</p> <ul style="list-style-type: none"> • 'Scenario' -- detections are reported in the scenario coordinate frame in rectangular coordinates. This option can only be selected when the sensor HasINS property is set to true. • 'Body' -- detections are reported in the body frame of the sensor platform in rectangular coordinates. • 'Sensor rectangular' -- detections are reported in the radar sensor coordinate frame in rectangular coordinates aligned with the sensor frame axes. • 'Sensor spherical' -- detections are reported in the radar sensor coordinate frame in spherical coordinates based on the sensor frame axes.

Sensitivity parameters.

Sensitivity Parameters

DetectionProbability	Probability of detecting a target with radar cross section, ReferenceRCS, at the range of ReferenceRange.
FalseAlarmRate	The probability of a false detection within each resolution cell of the radar. Resolution cells are determined from the AzimuthResolution and RangeResolution properties and when enabled the ElevationResolution and RangeRateResolution properties.
ReferenceRange	Range at which a target with radar cross section, ReferenceRCS, is detected with the probability specified in DetectionProbability.
ReferenceRCS	The target radar cross section (RCS) in dB at which the target is detected at the range specified by ReferenceRange with a detection probability specified by DetectionProbability.

Sensor resolution and bias parameters.

Resolution Parameters

AzimuthResolution	The radar azimuthal resolution defines the minimum separation in azimuth angle at which the radar can distinguish two targets.
ElevationResolution	The radar elevation resolution defines the minimum separation in elevation angle at which the radar can distinguish two targets. This property only applies when the HasElevation property is set to true.
RangeResolution	The radar range resolution defines the minimum separation in range at which the radar can distinguish two targets.
RangeRateResolution	The radar range rate resolution defines the minimum separation in range rate at which the radar can distinguish two targets. This property only applies when the HasRangeRate property is set to true.
AzimuthBiasFraction	This property defines the azimuthal bias component of the radar as a fraction of the radar azimuthal resolution specified by the AzimuthResolution property. This property sets a lower bound on the azimuthal accuracy of the radar.
ElevationBiasFraction	This property defines the elevation bias component of the radar as a fraction of the radar elevation resolution specified by the ElevationResolution property. This property sets a lower bound on the elevation accuracy of the radar. This property only applies when the HasElevation property is set to true.

RangeBiasFraction	This property defines the range bias component of the radar as a fraction of the radar range resolution specified by the RangeResolution property. This property sets a lower bound on the range accuracy of the radar.
RangeRateBiasFraction	This property defines the range rate bias component of the radar as a fraction of the radar range resolution specified by the RangeRateResolution property. This property sets a lower bound on the range rate accuracy of the radar. This property only applies when you set the HasRangeRate property to true.

Enabling parameters.

Enabling Parameters

HasElevation	This property allows the radar sensor to scan in elevation and estimate elevation from target detections.
HasRangeRate	This property allows the radar sensor to estimate range rate.
HasFalseAlarms	This property allows the radar sensor to generate false alarm detection reports.
HasRangeAmbiguities	When true, the radar does not resolve range ambiguities. When a radar sensor cannot resolve range ambiguities, targets at ranges beyond the <code>MaxUnambiguousRange</code> property value are wrapped into the interval <code>[0, MaxUnambiguousRange]</code> . When false, targets are reported at their unwrapped range.
HasRangeRateAmbiguities	When true, the radar does not resolve range rate ambiguities. When a radar sensor cannot resolve range rate ambiguities, targets at range rates above the <code>MaxUnambiguousRadialSpeed</code> property value are wrapped into the interval <code>[0, MaxUnambiguousRadialSpeed]</code> . When false, targets are reported at their unwrapped range rates. This property only applies when the <code>HasRangeRate</code> property is set to true.

HasNoise	Specifies if noise is added to the sensor measurements. Set this property to <code>true</code> to report measurements with noise. Set this property to <code>false</code> to report measurements without noise. The reported measurement noise covariance matrix contained in the output <code>objectDetection</code> struct is always computed regardless of the setting of this property.
HasINS	Set this property to <code>true</code> to enable an optional input argument to pass the current estimate of the sensor platform pose to the sensor. This pose information is added to the <code>MeasurementParameters</code> field of the reported detections. Then, the tracking and fusion algorithms can estimate the state of the target detections in scenario coordinates.

Scan parameters.

Scan Parameters

<p>ScanMode</p>	<p>This property specifies the scan mode used by the radar as one of:</p> <ul style="list-style-type: none"> • 'No scanning' -- the radar does not scan. The radar beam points along the antenna boresight. • 'Mechanical' -- the radar mechanically scans between the azimuth and elevation limits specified by the MechanicalScanLimits property. • 'Electronic' -- the radar electronically scans between the azimuth and elevation limits specified by the ElectronicScanLimits property. • 'Mechanical and electronic' -- the radar mechanically scans the antenna boresight between the mechanical scan limits and electronically scans beams relative to the antenna boresight between the electronic scan limits. The total field of regard scanned in this mode is the combination of the mechanical and electronic scan limits. <p>In all scan modes except 'No scanning', the scan proceeds at angular intervals specified by the radar field of view specified in FieldOfView.</p>

MaxMechanicalScanRate	This property sets the magnitude of the maximum mechanical scan rate of the radar. When <code>HasElevation</code> is <code>true</code> , the scan rate is a vector consisting of separate azimuthal and elevation scan rates. When <code>HasElevation</code> is <code>false</code> , the scan rate is a scalar representing the azimuthal scan rate. The radar sets its scan rate to step the radar mechanical angle by the radar field of regard. When the required scan rate exceeds the maximum scan rate, the maximum scan rate is used.
MechanicalScanLimits	This property specifies the mechanical scan limits of the radar with respect to its mounted orientation. When <code>HasElevation</code> is <code>true</code> , the limits are specified by minimum and maximum azimuth and by minimum and maximum elevation. When <code>HasElevation</code> is <code>false</code> , limits are specified by minimum and maximum azimuth. Azimuthal scan limits cannot span more than 360 degrees and elevation scan limits must lie in the closed interval <code>[-90 90]</code> .
ElectronicScanLimits	This property specifies the electronic scan limits of the radar with respect to the current mechanical angle. When <code>HasElevation</code> is <code>true</code> , the limits are specified by minimum and maximum azimuth and by minimum and maximum elevation. When <code>HasElevation</code> is <code>false</code> , limits are specified by minimum and maximum azimuth. Both azimuthal and elevation scan limits must lie in the closed interval <code>[-90 90]</code> .

FieldOfView	This property specifies the sensor azimuthal and elevation fields of view. The field of view defines the total angular extent observed by the sensor during a sensor update. The field of view must lie in the interval $(0,180]$. Targets outside of the sensor angular field of view during a sensor update are not detected.
-------------	--

Range and range rate parameters.

Range and Range Rate Parameters

MaxUnambiguousRange	<p>This property specifies the range at which the radar can unambiguously resolve the range of a target. Targets detected at ranges beyond the unambiguous range are wrapped into the range interval $[0, \text{MaxUnambiguousRange}]$. This property only applies to true target detections when you set <code>HasRangeAmbiguities</code> property to <code>true</code>.</p> <p>This property also defines the maximum range at which false alarms are generated. This property only applies to false target detections when you set <code>HasFalseAlarms</code> property to <code>true</code>.</p>
MaxUnambiguousRadialSpeed	<p>This property specifies the maximum magnitude value of the radial speed at which the radar can unambiguously resolve the range rate of a target. Targets detected at range rates whose magnitude is greater than the maximum unambiguous radial speed are wrapped into the range rate interval $[-\text{MaxUnambiguousRadialSpeed}, \text{MaxUnambiguousRadialSpeed}]$. This property only applies to true target detections when you set both the <code>HasRangeRate</code> and <code>HasRangeRateAmbiguities</code> properties to <code>true</code>.</p> <p>This property also defines the range rate interval over which false target detections are generated. This property only applies to false target detections when you set both the <code>HasFalseAlarms</code> and <code>HasRangeRate</code> properties to <code>true</code>.</p>

Detector Input

Each sensor created by `monostaticRadarSensor` accepts as input an array of target structures. This structure serves as the interface between the `trackingScenario` and the sensors. You create the target struct from target poses and profile information produced by `trackingScenario` or equivalent software.

The structure contains these fields.

Field	Description
PlatformID	Unique identifier for the platform, specified as a scalar positive integer. This is a required field with no default value.
ClassID	User-defined integer used to classify the type of target, specified as a nonnegative integer. Zero is reserved for unclassified platform types and is the default value.
Position	Position of target in platform coordinates, specified as a real-valued, 1-by-3 vector. This is a required field with no default value. Units are in meters.
Velocity	Velocity of target in platform coordinates, specified as a real-valued, 1-by-3 vector. Units are in meters per second. The default is <code>[0 0 0]</code> .
Acceleration	Acceleration of target in platform coordinates specified as a 1-by-3 row vector. Units are in meters per second-squared. The default is <code>[0 0 0]</code> .
Orientation	Orientation of the target with respect to platform coordinates, specified as a scalar quaternion or a 3-by-3 rotation matrix. Orientation defines the frame rotation from the platform coordinate system to the current target body coordinate system. Units are dimensionless. The default is <code>quaternion(1,0,0,0)</code> .

Field	Description
AngularVelocity	Angular velocity of target in platform coordinates, specified as a real-valued, 1-by-3 vector. The magnitude of the vector defines the angular speed. The direction defines the axis of clockwise rotation. Units are in degrees per second. The default is [0 0 0].

You can create a target pose structure by merging information from the platform information output from the `targetProfiles` method of `trackingScenario` and target pose information output from the `targetPoses` method on the platform carrying the sensors. You can merge them by extracting for each `PlatformID` in the target poses array, the profile information in platform profiles array for the same `PlatformID`.

The platform `targetPoses` method returns this structure for each target other than the platform.

Target Poses

platformID
ClassID
Position
Velocity
Yaw
Pitch
Roll
AngularVelocity

The `platformProfiles` method returns this structure for all platforms in the scenario.

Platform Profiles

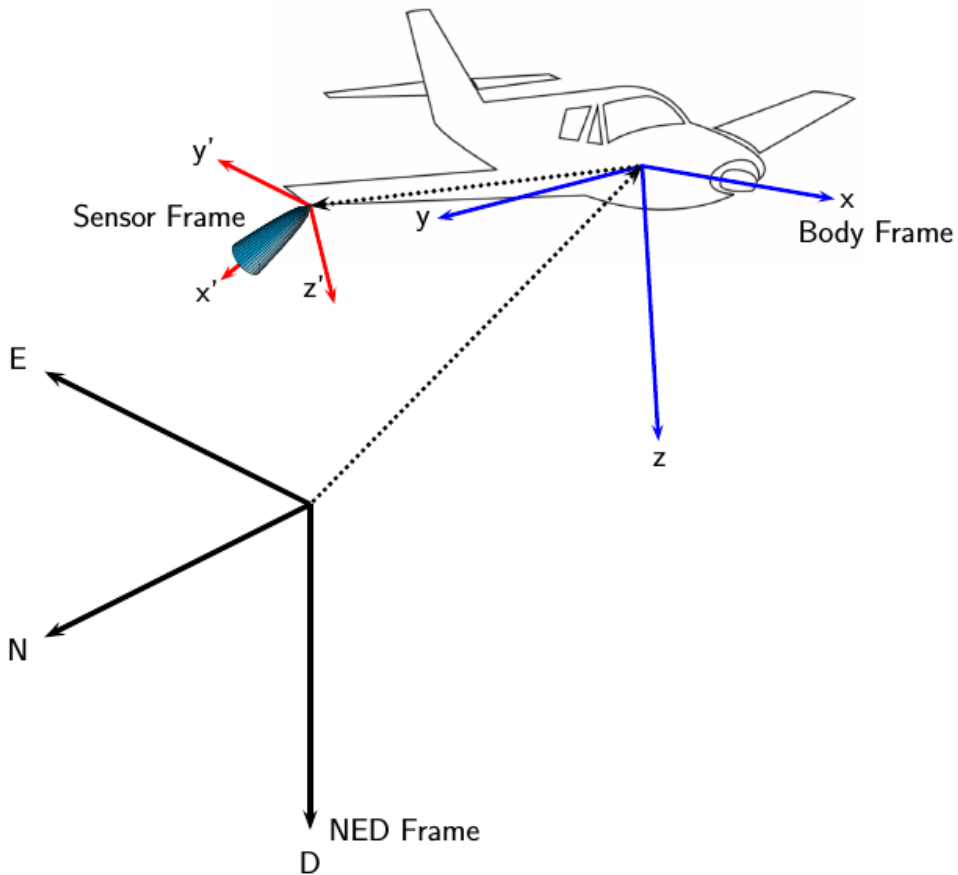
PlatformID
ClassID
RCSPattern
RCSAzimuthAngles
RCSElevationAngles

Radar Sensor Coordinate Systems

Detections consist of measurements of positions and velocities of targets and their covariance matrices. Detections are constructed with respect to sensor coordinates but can be output in one of several coordinates. Multiple coordinate frames are used to represent the positions and orientations of the various platforms and sensors in a scenario.

In a radar simulation, there is always a top-level global coordinate system which is usually the North-East-Down (NED) Cartesian coordinate system defined by a tangent plane at any point on the surface of the Earth. The `trackingScenario` object models the motion of platforms in the global coordinate system. When you create a platform, you specify its location and orientation relative to the global frame. These quantities define the body axes of the platform. Each radar sensor is mounted on the body of a platform. When you create a sensor, you specify its location and orientation with respect to the platform body coordinates. These quantities define the sensor axes. The body and radar axes can change

over time, however, global axes do not change.



Additional coordinate frames can be required. For example, often tracks are not maintained in NED (or ENU) coordinates, as this coordinate frame changes based on the latitude and longitude where it is defined. For scenarios that cover large areas (over 100 kilometers in each dimension), earth-centered earth-fixed (ECEF) can be a more appropriate global frame to use.

A radar sensor generates measurements in spherical coordinates relative to its sensor frame. However, the locations of the objects in the radar scenario are maintained in a top-level frame. A radar sensor is mounted on a platform and will, by default, only be aware of its position and orientation relative to the platform on which it is mounted. In other words, the radar expects all target objects to be reported relative to the platform body axes. The radar reports the required transformations (position and orientation) to relate the reported detections to the platform body axes. These transformations are used by consumers of the radar detections (e.g. trackers) to maintain tracks in the platform body axes. Maintaining tracks in the platform body axes enables the fusion of measurement or track information across multiple sensors mounted on the same platform.

If the platform is equipped with an inertial navigation system (INS) sensor, then the location and orientation of the platform relative to the top-level frame can be determined. This INS information can be used by the radar to reference all detections to scenario coordinates.

INS

When you specify `HasINS` as true, you must pass in an `INS` struct into the `step` method. This structure consists of the position, velocity, and orientation of the platform in scenario coordinates. These parameters let you express target poses in scenario coordinates by setting the `DetectionCoordinates` property.

Detections

Radar sensor detections are returned as a cell array of `objectDetection` objects. A detection contains these properties.

objectDetection Structure

Field	Definition
Time	Measurement time
Measurement	Measurements
MeasurementNoise	Measurement noise covariance matrix
SensorIndex	Unique ID of the sensor
ObjectClassID	Object classification
MeasurementParameters	Parameters used by initialization functions of any nonlinear Kalman tracking filters
ObjectAttributes	Additional information passed to tracker

Measurement and MeasurementNoise are reported in the coordinate system specified by the DetectionCoordinates property of the monostaticRadarSensor are reported in sensor Cartesian coordinates.

Measurement Coordinates

DetectionCoordinates	Measurement and Measurement Noise Coordinates		
'Scenario'	Coordinate Dependence on HasRangeRate		
'Body'			
'Sensor rectangular'	HasRangeRate		Coordinates
	true		[x;y;z;vx;vy;vz]
	false		[x;y;z]
'Sensor spherical'	Coordinate Dependence on HasRangeRate and HasElevation		
	HasRangeRate	HasElevation	Coordinates
	true	true	[az;el;rng;rr]
	true	false	[az;rng;rr]
	false	true	[az;el;rng]
	false	false	[az;rng]

The MeasurementParameters field consists of an array of structs describing a sequence of coordinate transformations from a child frame to a parent frame or the inverse transformations (see “Frame Rotation”). The longest possible sequence of transformations is: Sensor → Platform → Scenario. For example, if the detections are reported in sensor spherical coordinates and HasINS is set to false, then the sequence consists of one transformation from sensor to platform. If HasINS is true, the sequence of transformations consists of two transformations - first to platform coordinates then to scenario coordinates. Trivially, if the detections are reported in platform rectangular coordinates and HasINS is set to false, the transformation consists only of the identity.

Each struct takes the form:

MeasurementParameters

Parameter	Definition
Frame	Enumerated type indicating the frame used to report measurements. When detections are reported using a rectangular coordinate system, Frame is set to 'rectangular'. When detections are reported in spherical coordinates, Frame is set 'spherical' for the first struct.
OriginPosition	Position offset of the origin of frame(k) from the origin of frame(k+1) represented as a 3-by-1 vector.
OriginVelocity	Velocity offset of the origin of frame(k) from the origin of frame(k+1) represented as a 3-by-1 vector.
Orientation	A 3-by-3 real-valued orthonormal frame rotation matrix which rotates the axes of frame(k+1) into alignment with the axes of frame(k).
IsParentToChild	A logical scalar indicating if Orientation performs a frame rotation from the parent coordinate frame to the child coordinate frame. If false, Orientation performs a frame rotation from the child's coordinate frame to the parent's coordinate frame.
HasElevation	A logical scalar indicating if the frame has three-dimensional position. Only set to false for the first struct when detections are reported in spherical coordinates and HasElevation is false, otherwise it is true.
HasVelocity	A logical scalar indicating if the reported detections include velocity measurements. true when HasRangeRate is enabled, otherwise false.

ObjectAttributes

Attribute	Definition
TargetIndex	Identifier of the platform, PlatformID, that generated the detection. For false alarms, this value is negative.
SNR	Detection signal-to-noise ratio in dB.

Multi-Object Tracking

- “Tracking and Tracking Filters” on page 3-2
- “Linear Kalman Filters” on page 3-10
- “Extended Kalman Filters” on page 3-18

Tracking is the process of estimating the state of motion of an object based on measurements taken off the object. For an object moving in space, the state usually consists of position, velocity, and any other state parameters of objects at any given time. A state is the necessary information needed to predict future states of the system given the specified equations of motion. The estimates are derived from observations on the objects and are updated as new observations are taken. Observations are made using one or more sensors. Observations can only be used to update a track if it is likely that the observation is that of the object having that track. Observations need to be either associated with an existing track or used to create a new track. When several tracks are present, there are several ways observations are associated with one and only one track. The chosen track is based on the "closest" track to the observation.

Tracking and Tracking Filters

Multi-Object Tracking

You can use the multi-sensor, multi-target trackers, `trackerGNN`, and `trackerTOMHT`, to track many targets simultaneously. Tracks are initiated and updated using sensor detections of targets. There are several steps in the execution of the tracker when new detections are made.

- The tracker tries to assign a detection to an existing track.
- The tracker creates a track for each detection it cannot assign. When starting the tracker, all detections are used to create tracks.
- The tracker evaluates the status of each track. For new tracks, the status is judged to be tentative until enough detections are made to confirm the track. For existing tracks, newly assigned detections are used by the tracking filter to update the track state. When a track has no new added detections, it is held in a coasted state until new detections are assigned to it. If no new detections are added after a specified number of updates, the track is deleted.

on the command line.

When tracking multiple objects, there are several things to consider

- Decide which type of tracking filter to use. The choice of tracking filter depends on the expected dynamics of the object you want to track. The toolbox provides five Kalman filters for this purpose: the Linear Kalman filter, `trackingKF`, the Extended Kalman filter, `trackingEKF`, the Unscented Kalman filter, `trackingUKF`, the Cubature Kalman filter, `trackingCKF`, and the Interacting Multiple Model (IMM) Kalman filter, `trackingIMM`. The linear Kalman filter is used when the dynamics of the object follow a linear model and the measurements are linear functions of the state vector. The extended, unscented, and cubature Kalman filters are used when the dynamics are nonlinear or the measurement model is nonlinear or both.

The toolbox provides multi-object trackers, `trackerGNN`, and `trackerTOMHT`, which run the tracking filters and manage tracks. You can set the type of filter using an initialization function such as `initcvkf` which creates a constant-velocity linear Kalman filter from a single detection report. For any nonlinear Kalman filter, the initialization lets you specify a state transition function and a measurement function. For the extended Kalman filter, you can specify an optional state transition function

Jacobian and an optional measurement function Jacobian. For example, `initcaekf` creates a constant-acceleration extended Kalman filter.

- Choose which track assignment function to use. The assignment function determines whether a new detection belongs to an existing track or not. The toolbox contains three assignment algorithms, all of which use a cost matrix. Each column is assigned to a row in a way that minimizes the total cost. The algorithms are described in the help for each of these functions.
 - `assignmunkres` uses a Munkres assignment algorithm to find an optimal solution to the global nearest neighbor (GNN) assignment problem.
 - `assignauction` uses a forward/reverse auction assignment algorithm to find a suboptimal solution to the GNN assignment problem.
 - `assignjv` uses a Jonker-Volgenant assignment algorithm to find another type of optimal solution to the GNN assignment problem.
 - Other assignment algorithms include `assignkbest`, `assignkbestsd`, `assignsd`, and `assignTOMHT`.
- For the `trackerTOMHT` tracker, you can specify the conditions under which a track is confirmed or deleted by setting the `TrackLogic` property. Two algorithms are supported: 'History' and 'Score'.
 - 'History' -- track confirmation and deletion are based on the number of times the track has been assigned to a detection in the last several tracker updates.
 - 'Score' -- track confirmation and deletion are based on a log-likelihood computation. A high score means that the track is more likely to be valid. A low score means that the track is more likely to be false.

Multi-Object Tracker Properties

trackerGNN Properties

The `trackerGNN` object is a multi-sensor, multi-object tracker that uses global nearest neighbor association. Each detection can be assigned to only one track (single-hypothesis tracker) which can also be a new track that the detection initiates. At each step of the simulation, the tracker updates the track state. You can specify the behavior of the tracker by setting the following properties.

trackerGNN Properties

FilterInitializationFcn	A handle to a function that initializes a tracking filter based on a single detection. This function is called when a detection cannot be assigned to an existing track. For example, <code>initcaekf</code> creates an extended Kalman filter for an accelerating target. All tracks are initialized with the same type of filter.
Assignment	The name of the assignment algorithm. The tracker provides three built-in algorithms: 'Munkres', 'Jonker-Volgenant', and 'Auction' algorithms. You can also create your own custom assignment algorithm by specifying 'Custom'.
CustomAssignmentFcn	The name of the custom assignment algorithm function. This property is available on when the Assignment property is set to 'Custom'.
AssignmentThreshold	Specify the threshold that controls the assignment of a detection to a track. Detections can only be assigned to a track if their normalized distance from the track is less than the assignment threshold. Each tracking filter has a different method of computing the normalized distance. Increase the threshold if there are detections that can be assigned to tracks but are not. Decrease the threshold if there are detections that are erroneously assigned to tracks.

TrackLogic	Specify the track confirmation logic -- 'History' or 'Score'. For descriptions of these options, type help trackHistoryLogic or help trackScoreLogic at the command line.
ConfirmationThreshold	Specify the threshold for track confirmation. The threshold depends on the setting for TrackLogic <ul style="list-style-type: none">• 'History' -- specify the confirmation threshold as [M N]. If the track is detected at least M times in the last N updates, the track is confirmed.• 'Score' --- specify the confirmation threshold as a single number. If the score is greater than or equal to the threshold, this track is confirmed. .

<p>DeletionThreshold</p>	<p>Specify the threshold for track deletion. The threshold depends on the setting of TrackLogic</p> <ul style="list-style-type: none"> • 'History' -- specify the deletion threshold as a pair of integers [P R]. A track is deleted if it is not assigned to a track at least P times in the last R updates. • 'Score' --- specify the deletion threshold as a single number. The track is deleted if its score decreases by at least this threshold from its maximum track score.
<p>DetectionProbability</p>	<p>Specify the probability of detection as a number in the range (0,1). The probability of detection is used to calculate the track score when initializing and updating a track. This property is used only when TrackLogic is set to 'Score'.</p>
<p>FalseAlarmRate</p>	<p>Specify the rate of false detection as a number in the range (0,1). The false alarm rate is used to calculate the track score when initializing and updating a track. This property is used only when TrackLogic is set to 'Score'.</p>
<p>Beta</p>	<p>Specify the rate of new tracks per unit volume as a positive number. This property is used only when TrackLogic is set to 'Score'. The rate of new tracks is used in calculating the track score during track initialization. This property is used only when TrackLogic is set to 'Score'.</p>

Volume	Specify the volume of the sensor measurement bin as a positive scalar. For example, a radar sensor that produces a 4-D measurement of azimuth, elevation, range, and range-rate creates a 4-D volume. The volume is a product of the radar angular beamwidth, the range bin width, and the range-rate bin width. The volume is used in calculating the track score when initializing and updating a track. This property is used only when <code>TrackLogic</code> is set to 'Score'.
MaxNumTracks	Specify the maximum number of tracks the tracker can maintain.
MaxNumSensors	Specify the maximum number of sensors sending detections to the tracker as a positive integer. This number must be greater than or equal to the largest <code>SensorIndex</code> value used in the <code>objectDetection</code> input to the <code>step</code> method. This property determines how many sets of <code>ObjectAttributes</code> each track can have.
HasDetectableTrackIDsInput	Set this property to <code>true</code> if you want to provide a list of detectable track IDs as input to the <code>step</code> method. This list contains all tracks that the sensors expect to detect and, optionally, the probability of detection for each track ID.
HasCostMatrixInput	Set this property to <code>true</code> if you want to provide an assignment cost matrix as input to the <code>step</code> method.

trackerGNN Input

The input to the `trackerGNN` consists of a list of detections, the update time, cost matrix, and other data. Detections are specified as a cell array of `objectDetection` objects (see "Detections" on page 2-20). The input arguments are listed here.

trackerGNN Input

tracker	A trackerGNN object.
detections	Cell array of objectDetection objects (see "Detections" on page 2-20).
time	Time to which all the tracks are to be updated and predicted. The time at this execution step must be greater than the value in the previous call.
costmatrix	Cost matrix for assigning detections to tracks. A real T -by- D matrix, where T is the number of tracks listed in the allTracks argument returned from the previous call to step. D is the number of detections that are input in the current call. A larger cost matrix entry means a lower likelihood of assignment.
detectableTrackIDs	IDs of tracks that the sensors expect to detect, specified as an M -by-1 or M -by-2 matrix. The first column consists of track IDs, as reported in the TrackID field of the tracker output. The second column is optional and allows you to add the detection probability for each track.

trackerGNN Output

The output of the tracker can consist of up to three struct arrays with track state information. You can retrieve just the confirmed tracks, the confirmed and tentative tracks, or these tracks plus a combined list of all tracks.

```
confirmedTracks = step(...)
```

```
[confirmedTracks, tentativeTracks] = step(...)
```

```
[confirmedTracks, tentativeTracks, allTracks] = step(...)
```

The fields contained in the struct are:

trackerGNN Output struct

TrackID	Unique integer that identifies the track.
UpdateTime	Time to which the track is updated.
Age	Number of updates since track initialization.
State	State vector at update time.
StateCovariance	State covariance matrix at update time.
IsConfirmed	True if the track is confirmed.
TrackLogic	The track logic used in confirming the track - 'History' or 'Score'.
TrackLogicState	The current state of the track logic. <ul style="list-style-type: none"> • For 'History' track logic, a 1-by-Q logical array, where Q is the larger of N specified in the confirmation threshold property, <code>ConfirmationThreshold</code>, and R specified in the deletion threshold property, <code>DeletionThreshold</code>. • For 'Score' track logic, a 1-by-2 numerical array in the form: <code>[currentScore, maxScore]</code>.
IsCoasted	True if the track has been updated without a detection. In this case, tracks are predicted to the current time.
ObjectClassID	An integer value representing the target classification. Zero is reserved for an "unknown" class.
ObjectAttributes	A cell array of cells. Each cell captures the object attributes reported by the corresponding sensor.

Linear Kalman Filters

In this section...

“State Equations” on page 3-10

“Measurement Models” on page 3-12

“Linear Kalman Filter Equations” on page 3-12

“Filter Loop” on page 3-13

“Constant Velocity Model” on page 3-15

“Constant Acceleration Model” on page 3-16

When you use a Kalman filter to track objects, you use a sequence of detections or measurements to construct a model of the object motion. Object motion is defined by the evolution of the state of the object. The Kalman filter is an optimal, recursive algorithm for estimating the track of an object. The filter is recursive because it updates the current state using the previous state, using measurements that may have been made in the interval. A Kalman filter incorporates these new measurements to keep the state estimate as accurate as possible. The filter is optimal because it minimizes the mean-square error of the state. You can use the filter to predict future states or estimate the current state or past state.

State Equations

For most types of objects tracked in Sensor Fusion and Tracking Toolbox, the state vector consists of one-, two- or three-dimensional positions and velocities.

Start with Newton equations for an object moving in the x -direction at constant acceleration and convert these equations to space-state form.

$$m\ddot{x} = f$$
$$\ddot{x} = \frac{f}{m} = a$$

If you define the state as

$$x_1 = x$$
$$x_2 = \dot{x},$$

you can write Newton's law in state-space form.

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} a$$

You use a linear dynamic model when you have confidence that the object follows this type of motion. Sometimes the model includes process noise to reflect uncertainty in the motion model. In this case, Newton's equations have an additional term.

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} a + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v_k$$

v_k and is the unknown noise perturbations of the acceleration. Only the statistics of the noise are known. It is assumed to be zero-mean Gaussian white noise.

You can extend this type of equation to more than one dimension. In two dimensions, the equation has the form

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ a_x \\ 0 \\ a_y \end{bmatrix} + \begin{bmatrix} 0 \\ v_x \\ 0 \\ v_y \end{bmatrix}$$

The 4-by-4 matrix on the right side is the state transition model matrix. For independent x - and y - motions, this matrix is block diagonal.

When you transition to discrete time, you integrate the equations of motion over the length of the time interval. In discrete form, for a sample interval of T , the state-representation becomes

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \begin{bmatrix} 0 \\ T \end{bmatrix} a + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tilde{v}$$

The quantity x_{k+1} is the state at discrete time $k+1$, and x_k is the state at the earlier discrete time, k . If you include noise, the equation becomes more complicated, because the integration of noise is not straightforward.

The state equation can be generalized to

$$x_{k+1} = F_k x_k + G_k u_k + v_k$$

F_k is the state transition matrix and G_k is the control matrix. The control matrix takes into account any known forces acting on the object. Both of these matrices are given. The last term represents noise-like random perturbations of the dynamic model. The noise is assumed to be zero-mean Gaussian white noise.

Continuous-time systems with input noise are described by linear stochastic differential equations. Discrete-time systems with input noise are described by linear stochastic difference equations. A state-space representation is a mathematical model of a physical system where the inputs, outputs, and state variables are related by first-order coupled equations.

Measurement Models

Measurements are what you observe about your system. Measurements depend on the state vector but are not always the same as the state vector. For instance, in a radar system, the measurements can be spherical coordinates such as range, azimuth, and elevation, while the state vector is the Cartesian position and velocity. For the linear Kalman filter, the measurements are always linear functions of the state vector, ruling out spherical coordinates. To use spherical coordinates, use the extended Kalman filter.

The measurement model assumes that the actual measurement at any time is related to the current state by

$$z_k = H_k x_k + w_k$$

w_k represents measurement noise at the current time step. The measurement noise is also zero-mean white Gaussian noise with covariance matrix Q described by $Q_k = E[n_k n_k^T]$.

Linear Kalman Filter Equations

Without noise, the dynamic equations are

$$x_{k+1} = F_k x_k + G_k u_k.$$

Likewise, the measurement model has no measurement noise contribution. At each instance, the process and measurement noises are not known. Only the noise statistics are known. The

$$z_k = H_k x_k$$

You can put these equations into a recursive loop to estimate how the state evolves and also how the uncertainties in the state components evolve.

Filter Loop

Start with a best estimate of the state, $x_{0/0}$, and the state covariance, $P_{0/0}$. The filter performs these steps in a continual loop.

- 1 Propagate the state to the next step using the motion equations.

$$x_{k+1|k} = F_k x_{k|k} + G_k u_k.$$

Propagate the covariance matrix as well.

$$P_{k+1|k} = F_k P_{k|k} F_k^T + Q_k.$$

The subscript notation $k+1|k$ indicates that the quantity is the optimum estimate at the $k+1$ step propagated from step k . This estimate is often called the *a priori* estimate.

Then predict the measurement at the updated time.

$$z_{k+1|k} = H_{k+1} x_{k+1|k}$$

- 2 Use the difference between the actual measurement and predicted measurement to correct the state at the updated time. The correction requires computing the Kalman gain. To do this, first compute the measurement prediction covariance (innovation)

$$S_{k+1} = H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1}$$

Then the Kalman gain is

$$K_{k+1} = P_{k+1|k} H_{k+1}^T S_{k+1}^{-1}$$

and is derived from using an optimality condition.

- 3 Correct the predicted estimate with the measurement. Assume that the estimate is a linear combination of the predicted state and the measurement. The estimate after correction uses the subscript notation, $k+1|k+1$. is computed from

$$x_{k+1|k+1} = x_{k+1|k} + K_{k+1}(z_{k+1} - z_{k+1|k})$$

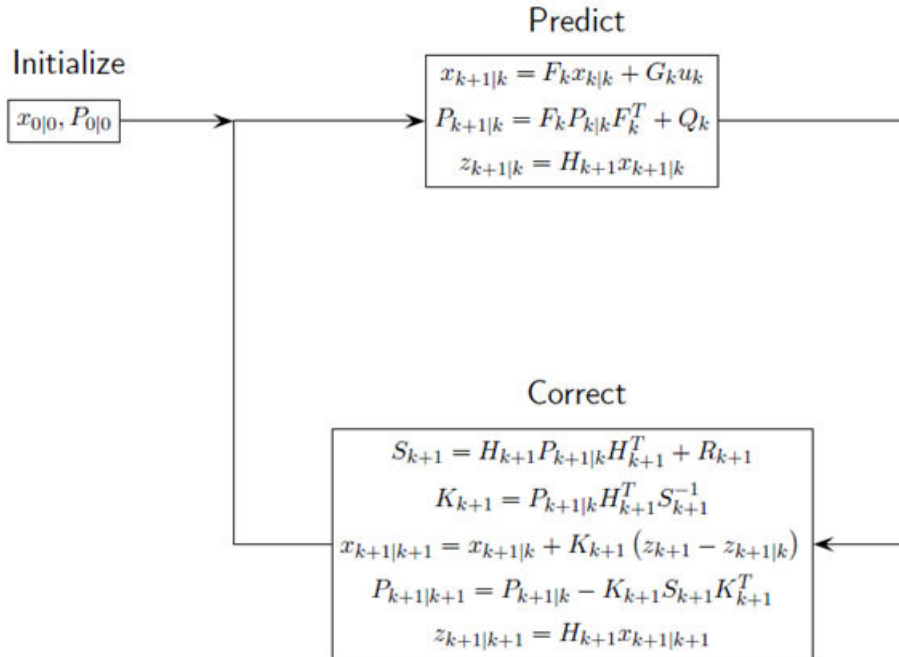
where K_{k+1} is the Kalman gain. The corrected state is often called the *a posteriori* estimate of the state because it is derived after the measurement is included.

Correct the state covariance matrix

$$P_{k+1|k+1} = P_{k+1|k} - K_{k+1}S_{k+1}K'_{k+1}$$

Finally, you can compute a measurement based upon the corrected state. This is not a correction to the measurement but is a best estimate of what the measurement would be based upon the best estimate of the state. Comparing this to the actual measurement gives you an indication of the performance of the filter.

This figure summarizes the Kalman loop operations.



Constant Velocity Model

The linear Kalman filter contains a built-in linear constant-velocity motion model. Alternatively, you can specify the transition matrix for linear motion. The state update at the next time step is a linear function of the state at the present time. In this filter, the measurements are also linear functions of the state described by a measurement matrix. For an object moving in 3-D space, the state is described by position and velocity in the x -, y -, and z -coordinates. The state transition model for the constant-velocity motion is

$$\begin{bmatrix} x_{k+1} \\ v_{x,k+1} \\ y_{k+1} \\ v_{y,k+1} \\ z_{k+1} \\ v_{z,k+1} \end{bmatrix} = \begin{bmatrix} 1 & T & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & T & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ v_{x,k} \\ y_k \\ v_{y,k} \\ z_k \\ v_{z,k} \end{bmatrix}$$

The measurement model is a linear function of the state vector. The simplest case is one where the measurements are the position components of the state.

$$\begin{bmatrix} m_{x,k} \\ m_{y,k} \\ m_{z,k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ v_{x,k} \\ y_k \\ v_{y,k} \\ z_k \\ v_{z,k} \end{bmatrix}$$

Constant Acceleration Model

The linear Kalman filter contains a built-in linear constant-acceleration motion model. Alternatively, you can specify the transition matrix for constant-acceleration linear motion. The transition model for linear acceleration is

$$\begin{bmatrix} x_{k+1} \\ v_{x,k+1} \\ a_{x,k+1} \\ y_{k+1} \\ v_{y,k+1} \\ a_{y,k+1} \\ z_{k+1} \\ v_{z,k+1} \\ a_{z,k+1} \end{bmatrix} = \begin{bmatrix} 1 & T & \frac{1}{2}T^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & T & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T & \frac{1}{2}T^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & T & \frac{1}{2}T^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ v_{x,k} \\ a_{x,k} \\ y_k \\ v_{y,k} \\ a_{y,k} \\ z_k \\ v_{z,k} \\ a_{z,k} \end{bmatrix}$$

The simplest case is one where the measurements are the position components of the state.

$$\begin{bmatrix} m_{x,k} \\ m_{y,k} \\ m_{z,k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ v_{x,k} \\ a_{x,k} \\ y_k \\ v_{y,k} \\ a_{y,k} \\ z_k \\ v_{z,k} \\ a_{z,k} \end{bmatrix}$$

Extended Kalman Filters

In this section...
“State Update Model” on page 3-18
“Measurement Model” on page 3-19
“Extended Kalman Filter Loop” on page 3-19
“Predefined Extended Kalman Filter Functions” on page 3-20

Use an extended Kalman filter when object motion follows a nonlinear state equation or when the measurements are nonlinear functions of the state. A simple example is when the state or measurements of the object are calculated in spherical coordinates, such as azimuth, elevation, and range.

State Update Model

The extended Kalman filter formulation linearizes the state equations. The updated state and covariance matrix remain linear functions of the previous state and covariance matrix. However, the state transition matrix in the linear Kalman filter is replaced by the Jacobian of the state equations. The Jacobian matrix is not constant but can depend on the state itself and time. To use the extended Kalman filter, you must specify both a state transition function and the Jacobian of the state transition function.

Assume there is a closed-form expression for the predicted state as a function of the previous state, controls, noise, and time.

$$x_{k+1} = f(x_k, u_k, w_k, t)$$

The Jacobian of the predicted state with respect to the previous state is

$$F^{(x)} = \frac{\partial f}{\partial x}$$

The Jacobian of the predicted state with respect to the noise is

$$F^{(w)} = \frac{\partial f}{\partial w_i}$$

These functions take simpler forms when the noise enters linearly into the state update equation:

$$x_{k+1} = f(x_k, u_k, t) + w_k$$

In this case, $F^{(w)} = 1_M$.

Measurement Model

In the extended Kalman filter, the measurement can be a nonlinear function of the state and the measurement noise.

$$z_k = h(x_k, v_k, t)$$

The Jacobian of the measurement with respect to the state is

$$H^{(x)} = \frac{\partial h}{\partial x}.$$

The Jacobian of the measurement with respect to the measurement noise is

$$H^{(v)} = \frac{\partial h}{\partial v}.$$

These functions take simpler forms when the noise enters linearly into the measurement equation:

$$z_k = h(x_k, t) + v_k$$

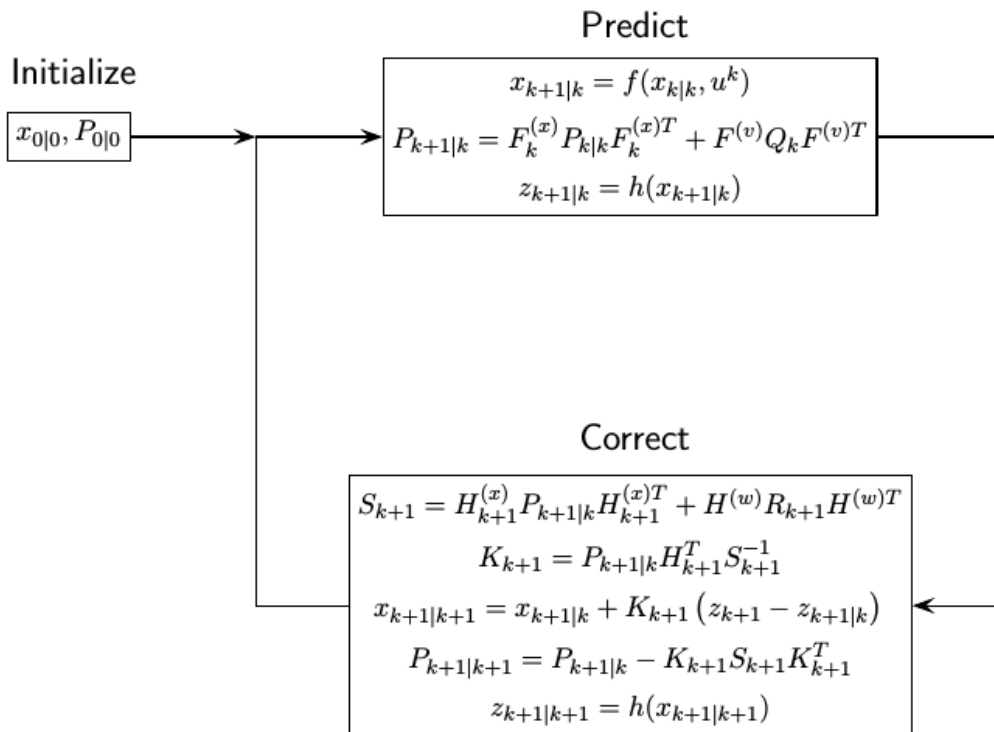
In this case, $H^{(v)} = 1_N$.

Extended Kalman Filter Loop

This extended Kalman filter loop is almost identical to the linear Kalman filter loop except that:

- The exact nonlinear state update and measurement functions are used whenever possible and the state transition matrix is replaced by the state Jacobian

- The measurement matrices are replaced by the appropriate Jacobians.



Predefined Extended Kalman Filter Functions

Sensor Fusion and Tracking Toolbox provides predefined state update and measurement functions to use in the extended Kalman filter.

Motion Model	Function Name	Function Purpose
Constant velocity	constvel	Constant-velocity state update model

Motion Model	Function Name	Function Purpose
	constveljac	Constant-velocity state update Jacobian
	cvmeas	Constant-velocity measurement model
	cvmeasjac	Constant-velocity measurement Jacobian
Constant acceleration	constacc	Constant-acceleration state update model
	constaccjac	Constant-acceleration state update Jacobian
	cameas	Constant-acceleration measurement model
	cameasjac	Constant-acceleration measurement Jacobian
Constant turn rate	constturn	Constant turn-rate state update model
	constturnjac	Constant turn-rate state update Jacobian
	ctmeas	Constant turn-rate measurement model
	ctmeasjac	Constant-turnrate measurement Jacobian

Data Structures

Target Pose

Target pose consists of the position, velocity, orientation, and signature of a target. All quantities are specified in the frame of a sensor platform. The target pose structure has these fields:

Field	Description
PlatformID	Unique identifier for the platform, specified as a scalar positive integer. This is a required field with no default value.
ClassID	User-defined integer used to classify the type of target, specified as a nonnegative integer. Zero is reserved for unclassified platform types and is the default value.
Position	Position of target in platform coordinates, specified as a real-valued, 1-by-3 vector. This is a required field with no default value. Units are in meters.
Velocity	Velocity of target in platform coordinates, specified as a real-valued, 1-by-3 vector. Units are in meters per second. The default is $[0 \ 0 \ 0]$.
Acceleration	Acceleration of target in platform coordinates specified as a 1-by-3 row vector. Units are in meters per second-squared. The default is $[0 \ 0 \ 0]$.
Orientation	Orientation of the target with respect to platform coordinates, specified as a scalar quaternion or a 3-by-3 rotation matrix. Orientation defines the frame rotation from the platform coordinate system to the current target body coordinate system. Units are dimensionless. The default is quaternion(1, 0, 0, 0).

Field	Description
AngularVelocity	Angular velocity of target in platform coordinates, specified as a real-valued, 1-by-3 vector. The magnitude of the vector defines the angular speed. The direction defines the axis of clockwise rotation. Units are in degrees per second. The default is [0 0 0].

Platform Pose

Platform pose consists of the position, velocity, orientation, and angular velocity of a platform with respect to scenario coordinates. The returned structure has these fields:

Field	Description
PlatformID	Unique identifier for the platform, specified as a scalar positive integer. This is a required field with no default value.
ClassID	User-defined integer used to classify the type of target, specified as a nonnegative integer. Zero is reserved for unclassified platform types and is the default value.
Position	Position of target in scenario coordinates, specified as a real-valued 1-by-3 vector. This is a required field with no default value. Units are in meters.
Velocity	Velocity of platform in scenario coordinates, specified as a real-valued 1-by-3 vector. Units are in meters per second. The default value is [0 0 0].
Acceleration	Acceleration of the platform in scenario coordinates, specified as a 1-by-3 row vector in meters per second-squared. The default value is [0 0 0].
Orientation	Orientation of the platform with respect to the local scenario NED coordinate frame, specified as a scalar quaternion or a 3-by-3 rotation matrix. The format is specified by the <code>fmt</code> input argument. Orientation defines the frame rotation from the local NED coordinate system to the current platform body coordinate system. Units are dimensionless. The default value is <code>quaternion(1,0,0,0)</code> .

Field	Description
AngularVelocity	Angular velocity of the platform in scenario coordinates, specified as a real-valued 1-by-3 vector. The magnitude of the vector defines the angular speed. The direction defines the axis of clockwise rotation. Units are in degrees per second. The default is value [0 0 0].

Platform Profiles

A profile contains the radar, IR, or sonar properties of a platform. The structure contains these fields:

Field	Description
PlatformID	Scenario-defined platform identifier, defined as a positive integer
ClassID	User-defined platform classification identifier, defined as a nonnegative integer
Signatures	Platform signatures defined as a cell array of radar cross-section (<code>rccSignature</code>), IR emission pattern (<code>irSignature</code>), and sonar target strength (<code>tsSignature</code>) objects.

Object Detections

Sensor detections are returned as a cell array of `objectDetection` objects. A detection contains these properties.

Property	Definition
Time	Measurement time
Measurement	Object measurements
MeasurementNoise	Measurement noise covariance matrix
SensorIndex	Unique ID of the sensor
ObjectClassID	Object classification
MeasurementParameters	Parameters used by initialization functions of nonlinear Kalman tracking filters
ObjectAttributes	Additional information passed to tracker

Measurements and Measurement Noise

The sensor measures the coordinates of the target. The `Measurement` and `MeasurementNoise` values are reported in the coordinate system specified by the `DetectionCoordinates` property of the sensor.

When the `DetectionCoordinates` property is `'Scenario'`, `'Body'`, or `'Sensor rectangular'`, the `Measurement` and `MeasurementNoise` values are reported in rectangular coordinates. Velocities are only reported when the range rate property, `HasRangeRate`, is true.

When the `DetectionCoordinates` property is `'Sensor spherical'`, the `Measurement` and `MeasurementNoise` values are reported in a spherical coordinate system derived from the sensor rectangular coordinate system. Elevation and range rate are only reported when `HasElevation` and `HasRangeRate` are true.

Measurements are ordered as [azimuth, elevation, range, range rate]. Reporting of elevation and range rate depends on the corresponding `HasElevation` and `HasRangeRate` property values. Angles are in degrees, range is in meters, and range rate is in meters per second.

Measurement Coordinates

DetectionCoordinates	Measurement and Measurement Noise Coordinates		
'Scenario'	Coordinate Dependence on HasRangeRate		
'Body'			
'Sensor rectangular'	HasRangeRate	Coordinates	
	true	[x; y; z; vx; vy; vz]	
	false	[x; y; z]	
'Sensor spherical'	Coordinate Dependence on HasRangeRate and HasElevation		
	HasRangeRate	HasElevation	Coordinates
	true	true	[az; el; rng; rr]
	true	false	[az; rng; rr]
	false	true	[az; el; rng]
	false	false	[az; rng]

Measurement Parameters

The MeasurementParameters field consists of an array of structures that describe a sequence of coordinate transformations from a child frame to a parent frame or the inverse transformations (see “Frame Rotation”). The longest possible sequence of transformations is Sensor → Platform → Scenario. For example, if the detections are reported in sensor spherical coordinates and HasINS is set to false, then the sequence consists of one transformation from sensor to platform. If HasINS is true, the sequence of transformations consists of two transformations - first to platform coordinates then to scenario coordinates. Trivially, if the detections are reported in platform rectangular coordinates and HasINS is set to false, the transformation consists only of the identity.

The structure fields are shown here. Not all fields have to be present in the structure. The set of fields and their default values can depend on the type of sensor.

Field	Description
-------	-------------

Frame	Enumerated type indicating the frame used to report measurements. When detections are reported using a rectangular coordinate system, Frame is set to 'rectangular'. When detections are reported in spherical coordinates, Frame is set 'spherical' for the first struct.
OriginPosition	Position offset of the origin of the frame relative to the parent frame, represented as a 3-by-1 vector.
OriginVelocity	Velocity offset of the origin of the frame relative to the parent frame, represented as a 3-by-1 vector.
Orientation	3-by-3 real-valued orthonormal frame rotation matrix.
IsParentToChild	A logical scalar indicating if Orientation performs a frame rotation from the parent coordinate frame to the child coordinate frame. If false, Orientation performs a frame rotation from the child coordinate frame to the parent coordinate frame.
HasElevation	A logical scalar indicating if elevation is included in the measurement. For measurements reported in a rectangular frame, and if HasElevation is false, the measurements are reported assuming 0 degrees of elevation.
HasAzimuth	A logical scalar indication if azimuth is included in the measurement.
HasRange	A logical scalar indication if range is included in the measurement.

HasVelocity	A logical scalar indicating if the reported detections include velocity measurements. For measurements reported in the rectangular frame, if HasVelocity is false, the measurements are reported as [x y z]. If HasVelocity is true, measurements are reported as [x y z vx vy vz].
Frame	Enumerated type indicating the frame used to report measurements. When detections are reported using a rectangular coordinate system, Frame is set to 'rectangular'. When detections are reported in spherical coordinates, Frame is set 'spherical' for the first struct.

Object Attributes

Object attributes contain additional information about a detection:

Attribute	Description
TargetIndex	Identifier of the platform, PlatformID, that generated the detection. For false alarms, this value is negative.
SNR	Detection signal-to-noise ratio in dB.

Signal Structure

Emitted signals have this structure:

Field	Description
PlatformID	1
EmitterIndex	1
OriginPosition	real-valued 3-by-1 vector
OriginVelocity	real-valued 3-by-1 vector
Orientation	1-by-1 quaternion
FieldOfView	[1 5]
EIRP	100
RCS	0
CenterFrequency	300e6
Bandwidth	3e6
WaveformType	0
ProcessingGain	0
PropagationRange	0
PropagationRangeRate	0
IsDirectPath	1

INS

INS is valid when the `HasINS` property is `true`.

Platform pose information from an inertial navigation system (INS) is a structure which has these fields:

Field	Definition
Position	Position of the GPS receiver in the local NED coordinate system specified as a real-valued 1-by-3 vector. Units are in meters.
Velocity	Velocity of the GPS receiver in the local NED coordinate system specified as a real-valued 1-by-3 vector. Units are in meters per second.
Orientation	Orientation of the INS with respect to the local NED coordinate system specified as a scalar quaternion or a 3-by-3 real-valued orthonormal frame rotation matrix. Defines the frame rotation from the local NED coordinate system to the current INS body coordinate system. This is also referred to as a "parent to child" rotation.

Sensor Configuration

Field	Description
SensorIndex	Unique sensor index
IsValidTime	Valid detection time, returned as 0 or 1. IsValidTime is 0 when detection updates are requested at times that are between update intervals specified by UpdateInterval.
IsScanDone	IsScanDone is true when the sensor has completed a scan.
FieldOfView	Field of view of sensor determines which objects fall within the sensor beam during object execution. The field of view is defined as a 2-by-1 vector of positive real values, [azfov;elfov].
MeasurementParameters	MeasurementParameters is an array of structures containing the coordinate frame transforms needed to transform positions and velocities in the top-level frame to the current sensor frame.

Emitter Configuration

Field	Description
EmitterIndex	Unique emitter index
IsValidTime	Valid emission time, returned as 0 or 1. IsValidTime is 0 when emitter updates are requested at times that are between update intervals specified by UpdateInterval.
IsScanDone	IsScanDone is true when the emitter has completed a scan.
FieldOfView	Field of view of emitter.
MeasurementParameters	MeasurementParameters is an array of structures containing the coordinate frame transforms needed to transform positions and velocities in the top-level frame to the current emitter frame.