









**help trackerGNN**

trackerGNN Tracking object using GNN assignment

tracker = trackerGNN creates a multi-sensor, multi-object tracker that

uses a global nearest neighbor (GNN) assignment to maintain a single

hypothesis about the objects it tracks. The trackerGNN initializes,

confirms, corrects, predicts (performs coasting), and deletes tracks.

A track is created with a 'Tentative' status, meaning that there is not

enough evidence for the trackerGNN to determine that the track is of a

physical object. If enough additional detections are assigned to the

tentative track, its status will change to 'Confirmed' (see

ConfirmationThreshold). Alternatively, a track will be confirmed if a

detection with a nonzero ObjectClassID value is assigned to it, as it

means that the sensor is able to classify the physical object.

tracker = trackerGNN('Name', value) creates a trackerGNN object by

specifying its properties as name-value pair arguments. Unspecified

properties have default values. See the list of properties below.

Step method syntax: click on step for more details.

The step method is responsible for managing all the tracks:

1. The method attempts to assign the detections to existing tracks.

2. New tracks are created based on unassigned detections.

3. Tracks that are assigned to detections are updated and confirmed.

4. Tracks that are not assigned to detections are coasted (predicted)

and eventually deleted.

System objects may be called directly like a function instead of using

the step method. For example, y = step(obj) and y = obj() are

equivalent.

trackerGNN properties:

TrackerIndex - Unique identifier of the tracker

FilterInitializationFcn - A handle to a function that initializes

a tracking filter based on a detection

Assignment - The name of the assignment algorithm

CustomAssignmentFcn - Name of 'Custom' assignment function

AssignmentThreshold - The threshold that controls the

assignment of detections to tracks

AssignmentClustering - Clustering of detections and tracks

for assignment

TrackLogic - Choose track logic: 'History' or 'Score'

ConfirmationThreshold - Specify the threshold for the

confirmation logic

DeletionThreshold - Specify the threshold for the deletion

logic

DetectionProbability - Probability of detecting a target\*

FalseAlarmRate - Rate of false positive detections\*

Beta - The rate of new tracks per unit volume\*

Volume - The volume of the sensor detection bin\*

MaxNumTracks - Define the maximum number of tracks

MaxNumDetections - Define the maximum number of detections

MaxNumSensors - Define the maximum number of sensors

OOSMHandling - Handle out-of-sequence measurement (OOSM)

MaxNumOOSMSteps - Maximum number of OOSM steps

StateParameters - Parameters defining the track state

HasDetectableTrackIDsInput - Provide detectable track IDs as an input

HasCostMatrixInput - Provide cost matrix as an input

NumTracks - Number of tracks (Read-only)

NumConfirmedTracks - Number of confirmed tracks (Read-only)

EnableMemoryManagement - Enable memory management properties

MaxNumDetectionsPerSensor - Define the maximum number of detections

per sensor

MaxNumDetectionsPerCluster - Define the maximum number of detections

per cluster

MaxNumTracksPerCluster - Define the maximum number of tracks

per assignment cluster

ClusterViolationHandling - Handle run-time violation of cluster

bounds

ClassFusionMethod - Choice of class fusion method

InitialClassProbabilities - Define the initial class probabilities \*\*

ClassFusionWeight - Define the weight of class fusion \*\*

\* Properties with an asterisk are only used for track score logic.

\*\* Properties with two asterisks are only used for 'Bayes' class fusion.

trackerGNN methods:

step - Creates, updates, and deletes the tracks

predictTracksToTime - Predicts the tracks to a time stamp

getTrackFilterProperties - Returns the values of filter properties

setTrackFilterProperties - Sets values to filter properties

initializeTrack - Initialize a new track

confirmTrack - Confirm a track

deleteTrack - Delete an existing track

exportToSimulink - Export the tracker to a Simulink model

release - Allows property value and input characteristics changes

clone - Creates a copy of the trackerGNN

isLocked - Locked status (logical)

reset - Resets states of the trackerGNN

% EXAMPLE: Construct a tracker and use it to track two objects

% Construct a trackerGNN object with the default constant

% velocity Kalman filter initialization function, initcvkf

tracker = trackerGNN('FilterInitializationFcn', @initcvkf, ...

'ConfirmationThreshold', [4 5], ...

'DeletionThreshold', 10);

% Update the tracker with two detections with a nonzero ObjectClassID.

% These detections will create confirmed tracks.

detections = {objectDetection(1, [10;0], 'SensorIndex', 1, ...

'ObjectClassID', 5, 'ObjectAttributes', {struct('ID', 1)}); ...

objectDetection(1, [0;10], 'SensorIndex', 1, ...

'ObjectClassID', 2, 'ObjectAttributes', {struct('ID', 2)})};

time = 2;

tracks = tracker(detections, time);

% In this example, the initcvkf filter initialization function is used.

% The function initializes a 2-D constant velocity model, which assumes

% that the state is: [x; vx; y; vy].

% Because there were two detections, two tracks will be initialized.

% They are confirmed on initialization because their ObjectClassID is

% nonzero, indicating that the sensor was able to classify them.

% To find the position and velocity we use:

positionSelector = [1 0 0 0; 0 0 1 0]; % [x, y]

velocitySelector = [0 1 0 0; 0 0 0 1]; % [vx, vy]

positions = getTrackPositions(tracks, positionSelector)

velocities = getTrackVelocities(tracks, velocitySelector)

See also: objectDetection, objectTrack, trackerJPDA, trackerTOMHT,

trackerPHD

Documentation for trackerGNN

**help for trackerJPDA ---**

trackerJPDA Joint Probabilistic Data Association tracker

tracker = trackerJPDA creates a single-scan, multi-sensor,

multi-object tracker that uses Joint Probabilistic Data Association

to assign detections to each track. trackerJPDA applies a

probabilistic assignment where multiple detections from the same

sensor can be assigned to each track. trackerJPDA initializes,

confirms, corrects, predicts (performs coasting) and deletes

tracks. A track is created with a 'Tentative' status, meaning that

there is not enough evidence for the trackerJPDA to determine that

the track is of a physical object. If enough additional detections

are assigned to the tentative track, its status will change to

'Confirmed' (see ConfirmationThreshold). Alternatively, a track

will be confirmed if a detection with a nonzero ObjectClassID value

is assigned to it, as it means that the sensor is able to classify

the physical object.

tracker = trackerJPDA('Name', value) creates a trackerJPDA object

by specifying its properties as name-value pair arguments.

Unspecified properties have default values. See the list of

properties below.

Step method syntax: click on step

System objects may be called directly like a function instead of

using the step method. For example, y = step(obj) and y = obj() are

equivalent.

trackerJPDA properties:

TrackerIndex - Unique identifier of the tracker

FilterInitializationFcn - A handle to a function that initializes

a tracking filter based on a detection

MaxNumEvents - Value of k for k-best JPDA

EventGenerationFcn - A handle to a function that generates

the feasible joint event matrices

and their probabilities from a

likelihood matrix.

MaxNumTracks - Define the maximum number of tracks

MaxNumDetections - Define the maximum number of detections

MaxNumSensors - Define the maximum number of sensors

OOSMHandling - Handle out-of-sequence measurement (OOSM)

MaxNumOOSMSteps - Maximum number of OOSM steps

StateParameters - Parameters defining the track state

AssignmentThreshold - The threshold that controls the

assignment of detections to tracks

DetectionProbability - Probability of detection

InitializationThreshold - The probability threshold that allows

assigned detections to initialize a new track

TrackLogic - Choose track logic: 'History' or

'Integrated'

ConfirmationThreshold - The required threshold for confirmation

DeletionThreshold - The threshold below which a track is

deleted

HitMissThreshold - The required threshold to 'hit' the

TrackLogic or register a 'miss'(\*)

ClutterDensity - Spatial density of clutter measurements

NewTargetDensity - Spatial density of new targets (\*\*)

DeathRate - Time rate of target deaths (\*\*)

InitialExistenceProbability - Initial probability of track existence

(read only)(\*\*)

HasCostMatrixInput - Provide cost matrix as an input

HasDetectableTrackIDsInput - Provide detectable track IDs as an

input

NumTracks - Number of tracks (read only)

NumConfirmedTracks - Number of confirmed tracks (read only)

TimeTolerance - Absolute tolerance between time

stamps of cluster detections (seconds)

EnableMemoryManagement - Enable memory management properties

MaxNumDetectionsPerSensor - Define the maximum number of detections

per sensor

MaxNumDetectionsPerCluster - Define the maximum number of detections

per cluster

MaxNumTracksPerCluster - Define the maximum number of tracks

per assignment cluster

ClusterViolationHandling - Handle run-time violation of cluster

bounds

ClassFusionMethod - Choice of class fusion method

InitialClassProbabilities - Define the initial class probabilities \*\*\*

ClassFusionWeight - Define the weight of class fusion \*\*\*

\* Properties are only used for track history logic.

\*\* Properties are only used for track integrated logic.

\*\*\* Properties with three asterisks are only used for 'Bayes' class fusion.

trackerJPDA methods:

step - Creates, updates, and deletes the tracks

predictTracksToTime - Predicts the tracks to a time stamp

getTrackFilterProperties - Returns the values of filter properties

setTrackFilterProperties - Sets values to filter properties

initializeTrack - Initialize a new track

confirmTrack - Confirm a track

deleteTrack - Delete a track

exportToSimulink - Export the tracker to a Simulink model

release - Allows property value and input characteristics changes

clone - Creates a copy of the trackerJPDA

isLocked - Locked status (logical)

reset - Resets states of the trackerJPDA

% EXAMPLE: Construct a tracker and use it to track two objects

% Construct a trackerJPDA object with a default constant

% velocity Extended Kalman Filter and 'History' track logic.

% Increase AssignmentThreshold to 100 to let tracks be jointly

% associated

tracker = trackerJPDA('TrackLogic','History', 'AssignmentThreshold',100,...

'ConfirmationThreshold', [4 5], ...

'DeletionThreshold', 10);

% Create noisy detections of two objects moving along the y = 10,

% and the y = -10 axis with constant velocity in the x direction

pos\_true = [0 0 ; 40 -40 ; 0 0];

V\_true = 5\*[cosd(-30) cosd(30) ; sind(-30) sind(30) ;0 0];

% Create a theater plot to visualize tracks and detections

tp = theaterPlot('XLimits',[-1 150],'YLimits',[-50 50]);

trackP = trackPlotter(tp,'DisplayName','Tracks','MarkerFaceColor','g','HistoryDepth',0);

detectionP = detectionPlotter(tp,'DisplayName','Detections','MarkerFaceColor','r');

% To find the position and velocity, use:

positionSelector = [1 0 0 0 0 0; 0 0 1 0 0 0; 0 0 0 0 0 0]; % [x, y, 0]

velocitySelector = [0 1 0 0 0 0; 0 0 0 1 0 0; 0 0 0 0 0 0 ]; % [vx, vy, 0]

dt = 0.2;

for time = 0:dt:30

% update the true position of objects

pos\_true = pos\_true + V\_true\*dt;

% create noisy detections of the two objects

detection(1) = objectDetection(time,pos\_true(:,1)+1\*randn(3,1));

detection(2) = objectDetection(time,pos\_true(:,2)+1\*randn(3,1));

% step the tracker through time with the detections

[confirmed,tentative,alltracks,info] = tracker(detection,time);

% Get positions, velocity and label info for the plot

[pos,cov] = getTrackPositions(confirmed,positionSelector);

vel = getTrackVelocities(confirmed,velocitySelector);

meas = cat(2,detection.Measurement);

measCov = cat(3,detection.MeasurementNoise);

% Update the track plot only if there are any tracks

if numel(confirmed)>0

labels = arrayfun(@(x)num2str([x.TrackID]),confirmed,'UniformOutput',false);

trackP.plotTrack(pos,vel,cov,labels);

end

detectionP.plotDetection(meas',measCov);

drawnow;

% Look at the information output every 5 seconds:

if time>0 && mod(time,8) == 0

disp(['At time t = ' num2str(time) 'seconds,']);

disp('The cost of assignment was: ')

disp(info.CostMatrix);

disp(['Number of clusters: ' num2str(numel(info.Clusters))]);

if numel(info.Clusters) == 1

disp('-----------------------------')

disp('The two tracks were in the same cluster.')

disp(info.Clusters{1})

disp('Marginal Probabilities of association:')

disp(info.Clusters{1}.MarginalProbabilities)

end

end

end

See also: objectDetection, objectTrack, trackerGNN, trackerTOMHT,

trackerPHD, jpdaEvents

Documentation for trackerJPDA

**help for trackerTOMHT ---**

trackerTOMHT Track-oriented multi-hypothesis tracker

tracker = trackerTOMHT creates a multi-hypothesis, multi-sensor,

multi-object tracker that uses track-oriented multi-hypothesis. The

trackerTOMHT initializes, confirms, corrects, predicts (performs

coasting) and deletes tracks. A track is created with a 'Tentative'

status, meaning that there is not enough evidence for the trackerTOMHT

to determine that the track is of a physical object. If enough

additional detections are assigned to the tentative track, its status

will change to 'Confirmed' (see ConfirmationThreshold). Alternatively,

a track will be confirmed if a detection with a nonzero ObjectClassID

value is assigned to it, as it means that the sensor is able to

classify the physical object.

tracker = trackerTOMHT('Name', value) creates a trackerTOMHT object by

specifying its properties as name-value pair arguments. Unspecified

properties have default values. See the list of properties below.

Step method syntax: click on step for more details.

System objects may be called directly like a function instead of

using the step method. For example, y = step(obj) and y = obj() are

equivalent.

trackerTOMHT properties:

TrackerIndex - Unique identifier of the tracker

FilterInitializationFcn - A handle to a function that initializes

a tracking filter based on a detection

MaxNumTracks - Define the maximum number of tracks

MaxNumDetections - Define the maximum number of detections

MaxNumSensors - Define the maximum number of sensors

OOSMHandling - Handle out-of-sequence measurements (OOSM)

StateParameters - Parameters defining the track state

MaxNumHypotheses - Define the maximum number of hypotheses

MaxNumTrackBranches - Define the maximum number of branches

(track hypotheses) per track

MaxNumHistoryScans - Define the maximum number of scans kept

in the track history

AssignmentThreshold - The threshold that controls the

assignment of detections to tracks

ConfirmationThreshold - The required track score for confirmation

DeletionThreshold - The drop from track maximum score before

deletion

DetectionProbability - Probability of detection

FalseAlarmRate - Rate of false positive detections

Beta - The rate of new tracks per unit volume

Volume - The volume of the sensor detection bin

MinBranchProbability - Minimum global probability to avoid pruning

NScanPruning - Choose n-scan pruning method

HasCostMatrixInput - Provide cost matrix as an input

HasDetectableBranchIDsInput - Provide detectable branch IDs as an input

OutputRepresentation - Choice of track output method

HypothesesToOutput - An array of hypotheses to output

NumTracks - Number of tracks (read only)

NumConfirmedTracks - Number of confirmed tracks (read only)

trackerTOMHT methods:

step - Creates, updates, and deletes the tracks

getBranches - Returns the current list of track branches

getTrackFilterProperties - Returns the values of filter properties

setTrackFilterProperties - Sets values to filter properties

predictTracksToTime - Predicts the tracks to a time stamp

initializeTrack - Initialize a new track

deleteTrack - Delete a track

initializeBranch - Initialize a new track branch

confirmBranch - Confirm a track branch

deleteBranch - Delete a track branch

exportToSimulink - Export the tracker to a Simulink model

release - Allows property value and input characteristics changes

clone - Creates a copy of the trackerTOMHT

isLocked - Locked status (logical)

reset - Resets states of the trackerTOMHT

% EXAMPLE: Construct a tracker and use it to track two objects

% Construct a trackerTOMHT object with the default constant

% velocity Kalman filter initialization function, initcvkf

tracker = trackerTOMHT('FilterInitializationFcn', @initcvkf, ...

'ConfirmationThreshold', 20, ...

'DeletionThreshold', -7, ...

'MaxNumHypotheses', 10);

% Update the tracker with two detections with a nonzero ObjectClassID.

% These detections will create confirmed tracks.

detections = {objectDetection(1, [10;0], 'SensorIndex', 1, ...

'ObjectClassID', 5, 'ObjectAttributes', {struct('ID', 1)}); ...

objectDetection(1, [0;10], 'SensorIndex', 1, ...

'ObjectClassID', 2, 'ObjectAttributes', {struct('ID', 2)})};

time = 2;

tracks = tracker(detections, time);

% In this example, the initcvkf filter initialization function is used.

% The function initializes a 2-D constant velocity model, which assumes

% that the state is: [x; vx; y; vy].

% Because there were two detections, two tracks will be initialized.

% They are confirmed on initialization because their ObjectClassID is

% nonzero, indicating that the sensor was able to classify them.

% To find the position and velocity we use:

positionSelector = [1 0 0 0; 0 0 1 0]; % [x, y]

velocitySelector = [0 1 0 0; 0 0 0 1]; % [vx, vy]

positions = getTrackPositions(tracks, positionSelector)

velocities = getTrackVelocities(tracks, velocitySelector)

See also: objectDetection, objectTrack, trackerGNN, trackerJPDA,

trackerPHD, assignTOMHT, trackBranchHistory, clusterTrackBranches,

compatibleTrackBranches, pruneTrackBranches

Documentation for trackerTOMHT

**help for trackerPHD ---**

trackerPHD Tracking objects using a multi-target PHD filter

tracker = trackerPHD('SensorConfigurations',configurations) creates a

multi-sensor, multi-object tracker which uses a multi-target

Probability Hypothesis Density (PHD) filter to estimate the states of

the targets. configurations is a cell array of trackingSensorConfiguration

objects. The trackerPHD maintains track estimate and identity by

assigning a unique label, TrackID, to each component of the PHD.

It uses an Iterated-Corrector approach for each sensor to update

the density sequentially.

A peak from the PHD is labeled as a track, when its weight exceeds a

certain threshold, ExtractionThreshold. The track is given a 'Tentative'

status, meaning there is not enough evidence yet to determine if it is

of a physical object. When the weight increases beyond the

ConfirmationThreshold, it's status is changed to 'Confirmed'.

tracker = trackerPHD('Name',value) creates a trackerPHD by specifying

its properties as name-value pairs. See the list of properties below.

Step method syntax: click on step for more details.

The step method of the tracker is responsible for managing all the

tracks. It performs the following steps:

1. It operates on each sensor sequentially.

2. It finds detections with low likelihood from each sensor and uses

them for initializing new components in the PHD.

3. If the sensor produces multiple detections per object, it generates

multiple partitions of the detections to incorporate the unknown origin

of the measurement.

4. It weighs each partition and the detection clusters belonging to it.

5. It assigns and maintains labels (TrackIDs) for each new initiated

track originating from the PHD.

6. It maintains the PHD within tractable limits by pruning and merging

it.

trackerPHD properties:

TrackerIndex - Unique identifier of the tracker

SensorConfigurations - Configurations of sensors

PartitioningFcn - Function to partition detections

into clusters

BirthRate - Rate at which new targets are

born in the scenario per unit

time

DeathRate - Rate at which targets die in the

scenario per unit time

AssignmentThreshold - The threshold that controls the

list of detection cells, which

are considered for generating new

components in the PHD

ExtractionThreshold - The minimum weight of a component

in PHD to be called a tentative

track

ConfirmationThreshold - The minimum weight of a tentative

track to be confirmed

DeletionThreshold - The weight below which a component

from the PHD is deleted

MergingThreshold - The threshold below which

components belonging to the same

target track are merged

LabelingThresholds - Threshold for Label management

HasSensorConfigurationsInput - Update sensor configurations via

input

StateParameters - Parameters defining the track state

NumTracks - Number of tracks. (read-only)

NumConfirmedTracks - Number of confirmed tracks (read-only)

MaxNumSensors - Define the maximum number of sensors

MaxNumTracks - Define the maximum number of tracks

MaxNumComponents - Define the maximum number of

components in the PHD filter

trackerPHD methods:

step - Initializes, deletes and manages tracks

predictTracksToTime - Predicts the tracks to a time stamp

initializeTrack - Initialize a new track

deleteTrack - Delete an existing track

getPHDFilter - Get PHD filter with current estimates

sensorIndices - Return the list of sensor indices

exportToSimulink - Export the tracker to a Simulink model

release - Allows property value and input characteristics changes

clone - Creates a copy of the trackerPHD

isLocked - Locked status (logical)

reset - Resets states of the trackerPHD

% EXAMPLE: Construct a tracker and use it to track two objects.

% Construct a trackerPHD for tracking two objects using 1 sensor.

configuration = trackingSensorConfiguration(1);

% Set the IsValidTime flag of the sensor to true to let the tracker know

% that this sensor had the chance to report detections.

configuration.IsValidTime = true;

% Specify properties such as ClutterDensity for the sensor.

configuration.ClutterDensity = 1e-7;

tracker = trackerPHD('SensorConfigurations',configuration);

% Update the tracker with some detections from two objects

detections = cell(20,1);

for i = 1:10

detections{i} = objectDetection(0,[200;-30;0] + 0.1\*randn(3,1));

end

for j = 11:20

detections{j} = objectDetection(0,[100;5;0] + 0.1\*randn(3,1));

end

[confTracks,tentTracks,allTracks,analysisInfo] = tracker(detections,0);

% In the first-step, as no previous tracks are present all possible

% partitions of the detections are considered for birth.

analysisInfo.SensorAnalysisInfo{1}.IsBirthCell;

% Update the tracker again after 0.1 seconds by assuming that targets

% moved at a constant velocity

dT = 0.1;

for i = 1:20

detections{i}.Time = detections{i}.Time + dT;

detections{i}.Measurement = detections{i}.Measurement + [1;2;0]\*dT;

end

[confTracks,tentTracks,allTracks] = tracker(detections,dT);

% In this example, we used the default sensor configuration

% FilterInitializationFcn, initcvggiwphd, which uses a constant velocity

% model and defines the states as [x;vx;y;vy;z;vy]. To find the

% positions, we use

positionSelector = [1 0 0 0 0 0;0 0 1 0 0 0;0 0 0 0 1 0];

positions = getTrackPositions(confTracks,positionSelector)

% You can also use the property Extent and MeasurementRate of the track

% to check the other states of a Gamma Gaussian Inverse Wishart (GGIW)

% state.

confTracks(1).MeasurementRate

% When detections are not available, you can use the function

% predictTracksToTime to predict the tracks to a certain time stamp.

confPredictedTracks = predictTracksToTime(tracker,'confirmed',0.2);

% EXAMPLE2: Construct a tracker and use it to track single moving

% object. In this scenario you have a car that is moving at a constant

% speed which will be tracked with the help of a non scanning radar

% sensor fixed on the tower. In order to track the car you will feed

% the detections and sensor configurations generated from the radar

% sensor model to trackerPHD as an input.

% Create Scenario

scenario = trackingScenario;

scenario.StopTime = Inf;

scenario.UpdateRate = 0;

% Create platforms

Tower = platform(scenario,'ClassID',3);

Tower.Dimensions = struct( ...

'Length', 10, ...

'Width', 10, ...

'Height', 60, ...

'OriginOffset', [0 0 30]);

Car = platform(scenario,'ClassID',2);

Car.Dimensions = struct( ...

'Length', 4.7, ...

'Width', 1.8, ...

'Height', 1.4, ...

'OriginOffset', [-0.6 0 0.7]);

Car.Trajectory = waypointTrajectory( ...

[0 -15 -0.23;0.3 -29.5 -0.23;0.3 -42 -0.39;0.3 -56.5 -0.23; ...

-0.3 -78.2 -0.23;4.4 -96.4 -0.23], [0;1.4;2.7;4.1;6.3;8.2], ...

'Course', [-88;-89;-89;-92;-84;-71], ...

'GroundSpeed', [10;10;10;10;10;10], ...

'ClimbRate', [0;0;0;0;0;0], ...

'AutoPitch', true, ...

'AutoBank', true);

% Create sensors

NoScanning = fusionRadarSensor('SensorIndex', 1, ...

'UpdateRate', 10, ...

'MountingAngles', [-90 0 0], ...

'FieldOfView', [20 10], ...

'ScanMode', 'No scanning', ...

'HasINS', true, ...

'DetectionCoordinates', 'Scenario', ...

'TargetReportFormat','Detections','HasElevation',true);

% Assign sensors to platforms

Tower.Sensors = NoScanning;

% Create a theater plot to visualize sensor, sensor coverage,

% tracks and detections.

tp = theaterPlot('XLim', [-58 58], 'YLim', [-104 12], 'ZLim', [-109 8]);

set(tp.Parent,'YDir','reverse', 'ZDir','reverse');

view(tp.Parent, -37.5, 30);

platp = platformPlotter(tp,'DisplayName','Targets','MarkerFaceColor','k');

detp = detectionPlotter(tp,'DisplayName','Detections','MarkerSize',6, ...

'MarkerFaceColor',[0.85 0.325 0.098],'MarkerEdgeColor','k','History',10000);

covp = coveragePlotter(tp,'DisplayName','Sensor Coverage');

% Configure sensor configurations for the scenario.

sensorConfig = trackingSensorConfiguration(scenario.Platforms{1}.Sensors{1}, ...

'SensorTransformFcn',@cvmeas,'FilterInitializationFcn',@initcvggiwphd);

% Configure tracker.

tracker = trackerPHD('SensorConfigurations',sensorConfig, ...

'PartitioningFcn', @(x)partitionDetections(x,1,4.7),...

'AssignmentThreshold',20, 'ExtractionThreshold',0.8,...

'ConfirmationThreshold',1.5, 'MergingThreshold',20,...

'DeletionThreshold',2e-1, 'BirthRate',1e-3, 'HasSensorConfigurationsInput',true);

% Add a trackPlotter.

tPlotter = trackPlotter(tp,'DisplayName','Tracks');

% Simulation loop

while advance(scenario) && ishghandle(tp.Parent)

% generate sensor data

[dets, configs, sensorConfigPIDs] = detect(scenario);

% read sensor data

allDets = [dets{:}];

if ~isempty(allDets)

% extract column vector of measurement positions

meas = cat(2,allDets.Measurement)';

% extract measurement noise

measCov = cat(3,allDets.MeasurementNoise);

else

meas = zeros(0,3);

measCov = zeros(3,3,0);

end

truePoses = platformPoses(scenario);

truePosition = vertcat(truePoses(:).Position);

% update tracker

[cTracks,tTracks,allTracks] = tracker(dets,configs,scenario.SimulationTime);

% update plots

plotPlatform(platp,truePosition);

plotDetection(detp,meas,measCov);

plotCoverage(covp,coverageConfig(scenario));

% Update the trackPlotter here:

% In this example, we used the sensor configuration generated from

% the sensor model FilterInitializationFcn, initcvggiwphd, which uses

% a constant velocity model and defines the states as [x;vx;y;vy;z;vy].

% To find the positions, we use

positionSelector = [1 0 0 0 0 0;0 0 1 0 0 0;0 0 0 0 1 0];

positions = getTrackPositions(cTracks,positionSelector);

if ~isempty(cTracks)

labels = cell(numel(cTracks),1);

for i =1:numel(cTracks)

labels{i} = {['T',num2str(cTracks(i).TrackID)]};

end

plotTrack(tPlotter, positions,labels);

end

drawnow

end

See also: trackingSensorConfiguration, partitionDetections.

Documentation for trackerPHD