trackerGNN

Multi-sensor, multi-object tracker using GNN assignment

Description

The `trackerGNN` System object™ is a tracker capable of processing detections of many targets from multiple sensors. The tracker uses a global nearest-neighbor (GNN) assignment algorithm. The tracker initializes, confirms, predicts, corrects, and deletes tracks. Inputs to the tracker are detection reports generated by `objectDetection`, `fusionRadarSensor`, `irSensor`, or `sonarSensor` objects. The tracker estimates the state vector and state vector covariance matrix for each track. Each detection is assigned to at most one track. If the detection cannot be assigned to any track, the tracker initializes a new track.

Any new track starts in a *tentative* state. If enough detections are assigned to a tentative track, its status changes to *confirmed*. If the detection already has a known classification (the `ObjectClassID` field of the returned track is nonzero), that track is confirmed immediately. When a track is confirmed, the tracker considers the track to represent a physical object. If detections are not assigned to the track within a specifiable number of updates, the track is deleted.

To track objects using this object:

1. Create the `trackerGNN` object and set its properties.

2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
tracker = trackerGNN
tracker = trackerGNN(Name,Value)
```

Description

`tracker = trackerGNN` creates a `trackerGNN` System object with default property values.

example

`tracker = trackerGNN(Name,Value)` sets properties for the tracker using one or more name-value pairs. For example, `trackerGNN('FilterInitializationFcn',@initcvukf,'MaxNumTracks',100)` creates a multi-object tracker that uses a constant-velocity, unscented Kalman filter and allows a maximum of 100 tracks. Enclose each property name in quotes.

Properties

expand all

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`TrackerIndex` — Unique tracker identifier
`0` (default) | nonnegative integer

`FilterInitializationFcn` — Filter initialization function
`@initcvekf` (default) | function handle | character vector

`Assignment` — Assignment algorithm
`'MatchPairs'` (default) | `'Munkres'` | `'Jonker-Volgenant'` | `'Auction'` | `'Custom'`

`CustomAssignmentFcn` — Custom assignment function
character vector

`AssignmentClustering` — Clustering of detections and tracks for assignment
`'off'` (default) | `'on'`

`AssignmentThreshold` — Detection assignment threshold
`30*[1 Inf]` (default) | positive scalar | 1-by-2 vector of positive values

`TrackLogic` — Confirmation and deletion logic type
`'History'` (default) | `'Score'`

`ConfirmationThreshold` — Threshold for track confirmation
scalar | 1-by-2 vector

`DeletionThreshold` — Minimum score required to delete track
`[5 5]` or `-7` (default) | scalar | real-valued 1-by-2 vector of positive values

`DetectionProbability` — Probability of detection used for track score
`0.9` (default) | positive scalar between 0 and 1

`FalseAlarmRate` — Probability of false alarm used for track score
`1e-6` (default) | scalar

`Beta` — Rate of new tracks per unit volume
`1` (default) | positive scalar

`Volume` — Volume of sensor measurement bin
`1` (default) | positive scalar

`MaxNumTracks` — Maximum number of tracks
`100` (default) | positive integer

`MaxNumSensors` — Maximum number of sensors
`20` (default) | positive integer

`MaxNumDetections` — Maximum number of detections
`Inf` (default) | positive integer

`OOSMHandling` — Handling of out-of-sequence measurement (OOSM)
`'Terminate'` (default) | `'Neglect'` | `'Retrodiction'`

`MaxNumOOSMSteps` — Maximum number of out-of-sequence measurement steps
`3` (default) | positive integer

`StateParameters` — Parameters of track state reference frame
`struct([])` (default) | `struct array`

`HasDetectableTrackIDsInput` — Enable input of detectable track IDs
`false` (default) | `true`

`HasCostMatrixInput` — Enable cost matrix input
`false` (default) | `true`

`NumTracks` — Number of tracks maintained by tracker
nonnegative integer

`NumConfirmedTracks` — Number of confirmed tracks
nonnegative integer

`EnableMemoryManagement` — Enable memory management properties
`false` or `0` (default) | `true` or `1`

`MaxNumDetectionsPerSensor` — Maximum number of detections per sensor
`100` (default) | positive integer

`MaxNumDetectionsPerCluster` — Maximum number of detections per cluster
`5` (default) | positive integer

`MaxNumTracksPerCluster` **— Maximum number of tracks per cluster**
`5` (default) | positive integer

`ClusterViolationHandling` **— Handling of run-time violation of cluster bounds**
`'Split and warn'` (default) | `'Terminate'` | `'Split'`

`ClassFusionMethod` **— Class fusion method**
`"None"` (default) | `"Bayes"`

`InitialClassProbabilities` **— Prior class probability distribution for new tracks**
`1` (default) | N-element vector of nonnegative scalars that sum to `1`

`ClassFusionWeight` **— Weight factor of class cost**
`0.7` (default) | scalar in range `[0,1]`

Usage

To process detections and update tracks, call the tracker with arguments, as if it were a function (described here).

Syntax

```
confirmedTracks = tracker(detections,time)
confirmedTracks = tracker(detections,time,costMatrix)
confirmedTracks = tracker( __ ,detectableTrackIDs)
[confirmedTracks,tentativeTracks,allTracks] = tracker( __ )
[confirmedTracks,tentativeTracks,allTracks,analysisInformation] = tracker( __ )
```

Description

`confirmedTracks = tracker(detections,time)` returns a list of confirmed tracks that are updated from a list of detections, `detections`, at the update time, `time`. Confirmed tracks are corrected and predicted to the update time.

`confirmedTracks = tracker(detections,time,costMatrix)` also specifies a cost matrix, `costMatrix`.

To enable this syntax, set the `HasCostMatrixInput` property to `true`.

`confirmedTracks = tracker( __ ,detectableTrackIDs)` also specifies a list of expected detectable tracks, `detectableTrackIDs`.

To enable this syntax, set the `HasDetectableTrackIDsInput` property to `true`.

`[confirmedTracks,tentativeTracks,allTracks] = tracker( __ )` also returns a list of tentative tracks, `tentativeTracks`, and a list of all tracks, `allTracks`.

`[confirmedTracks,tentativeTracks,allTracks,analysisInformation] = tracker( __ )` also returns information, `analysisInformation`, which can be used for track analysis.

Input Arguments

expand all

`detections` — Detection list
cell array of `objectDetection` objects

`time` — Time of update
scalar

`costMatrix` — Cost matrix
real-valued N-by-M matrix

`detectableTrackIDs` — Detectable track IDs
real-valued M-by-1 vector | real-valued M-by-2 matrix

Output Arguments

expand all

`confirmedTracks` — Confirmed tracks
array of `objectTrack` objects | array of structures

`tentativeTracks` — Tentative tracks
array of `objectTrack` objects | array of structures

`allTracks` — All tracks
array of `objectTrack` objects | array of structures

`analysisInformation` — Additional information for analyzing track updates
structure

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

expand all

Specific to `trackerGNN`

Common to All System Objects

Examples

collapse all

Track Two Objects Using trackerGNN

Construct a `trackerGNN` object with the default 2-D constant-velocity Kalman filter initialization function, `initcvkf`.

Open Live Script

```
tracker = trackerGNN('FilterInitializationFcn', @initcvkf, ...
    'ConfirmationThreshold', [4 5], ...
    'DeletionThreshold', 10);
```

Update the tracker with two detections both having nonzero `ObjectClassID`. These detections immediately 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);
```

Find the positions and velocities.

```
positionSelector = [1 0 0 0; 0 0 1 0];
velocitySelector = [0 1 0 0; 0 0 0 1];

positions = getTrackPositions(tracks,positionSelector)
```

```
positions = 2×2

    10     0
     0    10
```

```
velocities = getTrackVelocities(tracks,velocitySelector)
```

```
velocities = 2×2

     0     0
     0     0
```

Detection Class Fusion Using `trackerGNN`

Create two `objectDetection` objects at time t = 0 and t = 1, respectively. The `ObjectClassID` of the two detections is `1`. Specify the confusion matrix for each detection.

Open Live Script

```
detection0 = objectDetection(0,[0 0 0],...
    ObjectClassID=1,...
    ObjectClassParameters=struct("ConfusionMatrix",[0.6 0.2 0.2; 0.2 0.6 0.2; 0.2 0.2 0.6]));
```

```
detection1 = objectDetection(1,[0 0 0],...
    ObjectClassID=1,...
    ObjectClassParameters=struct("ConfusionMatrix",[0.5 0.3 0.2; 0.3 0.5 0.2; 0.2 0.2 0.6]));
```

Create a `trackerGNN` object. Specify the class fusion method as `"Bayes"` and specify the initial probability of each class as `1/3`.

```
tracker = trackerGNN(ClassFusionMethod="Bayes",InitialClassProbabilities=[1/3 1/3 1/3])
```

```
tracker = 
  trackerGNN with properties:

                  TrackerIndex: 0
       FilterInitializationFcn: 'initcvekf'
                  MaxNumTracks: 100
              MaxNumDetections: Inf
                 MaxNumSensors: 20

                    Assignment: 'MatchPairs'
           AssignmentThreshold: [30 Inf]
          AssignmentClustering: 'off'

                  OOSMHandling: 'Terminate'

                    TrackLogic: 'History'
         ConfirmationThreshold: [2 3]
             DeletionThreshold: [5 5]

            HasCostMatrixInput: false
    HasDetectableTrackIDsInput: false
               StateParameters: [1x1 struct]

             ClassFusionMethod: 'Bayes'
     InitialClassProbabilities: [0.3333 0.3333 0.3333]
             ClassFusionWeight: 0.7000

                     NumTracks: 0
            NumConfirmedTracks: 0

        EnableMemoryManagement: false
```

Update the track with the first and second detections sequentially.

```
tracker(detection0,0);
[tracks,~,~,info] = tracker(detection1,1);
```

Show the maintained tracks and analysis information.

```
disp(tracks)
```

```
  objectTrack with properties:

                     TrackID: 1
```

```
                   BranchID: 0
                SourceIndex: 0
                 UpdateTime: 1
                        Age: 2
                      State: [6x1 double]
            StateCovariance: [6x6 double]
            StateParameters: [1x1 struct]
              ObjectClassID: 1
   ObjectClassProbabilities: [0.7500 0.1500 0.1000]
                 TrackLogic: 'History'
            TrackLogicState: [1 1 0 0 0]
                IsConfirmed: 1
                  IsCoasted: 0
             IsSelfReported: 1
           ObjectAttributes: [1x1 struct]
```

```
disp(info)
```

```
       OOSMDetectionIndices: [1x0 uint32]
    TrackIDsAtStepBeginning: 1
                 CostMatrix: 13.8823
                Assignments: [1 1]
           UnassignedTracks: [1x0 uint32]
       UnassignedDetections: [0x1 uint32]
          InitiatedTrackIDs: [1x0 uint32]
            DeletedTrackIDs: [1x0 uint32]
          TrackIDsAtStepEnd: 1
            ClassCostMatrix: -0.1823
```

Algorithms

expand all

› Tracker Logic Flow

› Detection Class Fusion

References

[1] Blackman, S., and R. Popoli. *Design and Analysis of Modern Tracking Systems.* Artech House Radar Library, Boston, 1999.

[2] Bar-Shalom, Y., et al. “Tracking with Classification-Aided Multiframe Data Association.” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 41, no. 3, July 2005, pp. 868–78.

[3] Kuncheva, Ludmila I., et al. “Decision Templates for Multiple Classifier Fusion: An Experimental Comparison.” *Pattern Recognition*, vol. 34, no. 2, Feb. 2001, pp. 299–314.

Extended Capabilities

› C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Version History

Introduced in R2018b

expand all

› **R2022b:** Fuse detection classification information

See Also

Blocks

Global Nearest Neighbor Multi Object Tracker

Functions

`assignauction` | `assignjv` | `assignkbest` | `assignkbestsd` | `assignmunkres` | `assignsd` | `assignTOMHT` | `getTrackPositions` | `getTrackVelocities` | `fusecovint` | `fusecovunion` | `fusexcov` | `clusterTrackBranches` | `compatibleTrackBranches` | `pruneTrackBranches` | `triangulateLOS`

Objects

`objectDetection` | `trackingKF` | `trackingEKF` | `trackingUKF` | `trackingABF` | `trackingCKF` | `trackingGSF` | `trackingIMM` | `trackingMSCEKF` | `trackingPF` | `trackHistoryLogic` | `trackScoreLogic` | `objectTrack` | `trackerJPDA` | `staticDetectionFuser` | `trackerTOMHT`

Topics

Introduction to Class Fusion and Classification-Aided Tracking

Analyze Track and Detection Association Using Analysis Info

Automatically Tune Tracking Filter for Multi-Object Tracker

Introduction to Multiple Target Tracking

Introduction to Assignment Methods in Tracking Systems