trackerJPDA

Joint probabilistic data association tracker
Since R2019a

Description

The `trackerJPDA` System object™ is a tracker capable of processing detections of multiple targets from multiple sensors. The tracker uses joint probabilistic data association to assign detections to each track. The tracker applies a soft assignment where multiple detections can contribute to each track. The tracker initializes, confirms, corrects, predicts (performs coasting), 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 estimate error covariance matrix for each track. Each detection is assigned to at least one track. If the detection cannot be assigned to any existing track, the tracker creates 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* (see the `ConfirmationThreshold` property). If the detection already has a known classification (i.e., the `ObjectClassID` field of the returned track is nonzero), that corresponding 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.

You can enable different JPDA tracking modes by specifying the TrackLogic and MaxNumEvents properties.

- Setting the TrackLogic property to `'Integrated'` to enable the joint integrated data association (JIPDA) tracker, in which track confirmation and deletion is based on the probability of track existence.
- Setting the MaxNumEvents property to a finite integer to enable the k-best joint integrated data association (k-best JPDA) tracker, which generates a maximum of k events per cluster.
- Setting the ClassFusionMethod property to `"Bayes"` to enable detection class fusion.

To track targets using this object:

1. Create the `trackerJPDA` 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 = trackerJPDA
tracker = trackerJPDA(Name,Value)
```

Description

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

example

`tracker = trackerJPDA(Name,Value)` sets properties for the tracker using one or more name-value pairs. For example, `trackerJPDA('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

> **`MaxNumEvents` — Value of k for k-best JPDA**
> `Inf` (default) | positive integer

> **`EventGenerationFcn` — Feasible joint events generation function**
> `@jpdaEvents` (default) | function handle | character vector

> **`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` — Handle 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`

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

`DetectionProbability` — Probability of detection
`0.9` (default) | scalar in the range [0,1]

`InitializationThreshold` — Threshold to initialize a track
`0` (default) | scalar in the range [0,1)

`TrackLogic` — Track confirmation and deletion logic type
`'History'` (default) | `'Integrated'`

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

`DeletionThreshold` — Threshold for track deletion
scalar | real-valued 1-by-2 vector

`HitMissThreshold` — Threshold for registering hit or miss
0.2 (default) | scalar in the range [0,1]

`ClutterDensity` — Spatial density of clutter measurements
`1e-6` (default) | positive scalar

`NewTargetDensity` — Spatial density of new targets
`1e-5` (default) | positive scalar

`DeathRate` — Time rate of target deaths
`0.01` (default) | scalar in the range [0,1]

`InitialExistenceProbability` — Initial probability of track existence
`0.9` (default) | scalar in the range [0,1]

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

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

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

`NumConfirmedTracks` — Number of confirmed tracks
nonnegative integer

`TimeTolerance` — Absolute time tolerance between detections
`1e-5` (default) | positive scalar

`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 fusion
`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 at the update time. Confirmed tracks are corrected and predicted to the update time, `time`.

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

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

`confirmedTracks = tracker( __ ,detectableTrackIDs)` also specifies a list of expected detectable tracks given by `detectableTrackIDs`. This argument can be used with any of the previous input syntaxes.

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

`[confirmedTracks,tentativeTracks,allTracks] = tracker( __ )` also returns a list of tentative tracks and a list of all tracks. You can use any of the input arguments in the previous syntaxes.

`[confirmedTracks,tentativeTracks,allTracks,analysisInformation] = tracker( __ )` also returns analysis information that can be used for track analysis. You can use any of the input arguments in the previous syntaxes.

Input Arguments

expand all

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

`time` — Time of update
scalar

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

Common to All System Objects

Examples

collapse all

Track Two Objects Using trackerJPDA

Construct a *trackerJPDA* object with a default constant velocity Extended Kalman Filter and 'History' track logic. Set *AssignmentThreshold* to 100 to allow tracks to be jointly associated.

Open Live Script

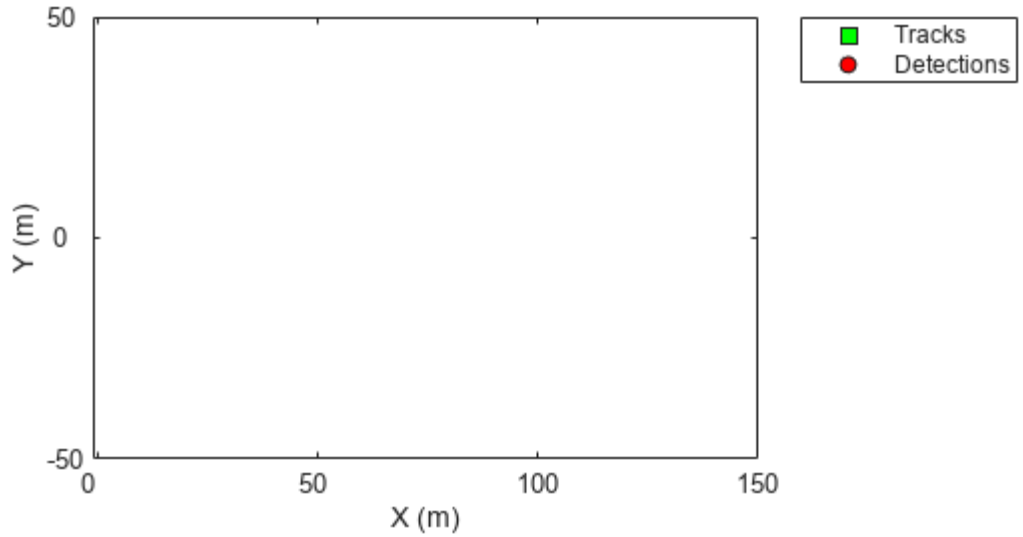
```
tracker = trackerJPDA('TrackLogic','History', 'AssignmentThreshold',100,...
    'ConfirmationThreshold', [4 5], ...
    'DeletionThreshold', [10 10]);
```

Specify the true initial positions and velocities of the two objects.

```
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 obtain the position and velocity, create position and velocity selectors.

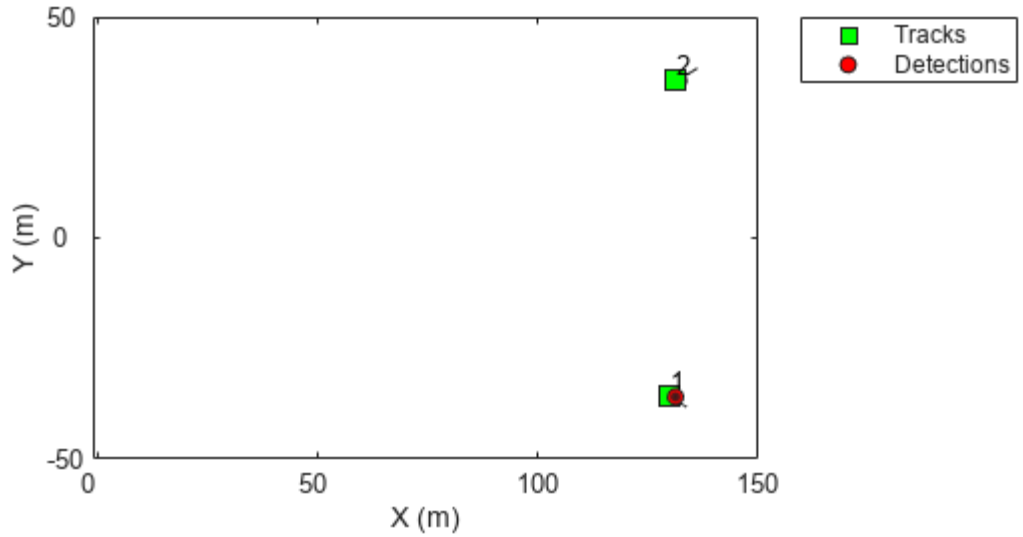
```
positionSelector = [1 0 0 0 0 0; 0 0 1 0 0 0; 0 0 0 0 0 0]; % [x, y, 0]
velocitySelector = [0 1 0 0 0 0; 0 0 0 1 0 0; 0 0 0 0 0 0 ]; % [vx, vy, 0]
```

Update the tracker with detections, display cost and marginal probability of association information, and visualize tracks with detections.

```
dt = 0.2;
for time = 0:dt:30
    % Update the true positions of objects.
    pos_true = pos_true + V_true*dt;

    % Create detections of the two objects with noise.
    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);

    % Extract position, velocity and label info.
    [pos,cov] = getTrackPositions(confirmed,positionSelector);
    vel = getTrackVelocities(confirmed,velocitySelector);
    meas = cat(2,detection.Measurement);
    measCov = cat(3,detection.MeasurementNoise);

    % Update the plot 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;

    % Display the cost and marginal probability of distribution every eight
    % 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('The two tracks were in the same cluster.')
            disp('Marginal probabilities of association:')
            disp(info.Clusters{1}.MarginalProbabilities)
        end
        disp('-----------------------------')
    end
end
```

```
At time t = 8 seconds,
The cost of assignment was:
   1.0e+03 *

    0.0020    1.1523
    1.2277    0.0053
Number of clusters: 2
-----------------------------
At time t = 16 seconds,
The cost of assignment was:
    1.3968    4.5123
    2.0747    1.9558
Number of clusters: 1
The two tracks were in the same cluster.
Marginal probabilities of association:
    0.8344    0.1656
    0.1656    0.8344
    0.0000    0.0000
-----------------------------
At time t = 24 seconds,
The cost of assignment was:
   1.0e+03 *

    0.0018    1.2962
    1.2664    0.0013
Number of clusters: 2
-----------------------------
```



Detection Class Fusion Using `trackerJPDA`

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 `trackerJPDA` object. Set the class fusion method to `"Bayes"` and specify the initial probability of each class as `1/3`.

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

```
tracker =
  trackerJPDA with properties:

                TrackerIndex: 0
      FilterInitializationFcn: 'initcvekf'
                MaxNumEvents: Inf
          EventGenerationFcn: 'jpdaEvents'
                MaxNumTracks: 100
```

```
               MaxNumDetections: Inf
                  MaxNumSensors: 20
                  TimeTolerance: 1.0000e-05

            AssignmentThreshold: [30 Inf]
       InitializationThreshold: 0
          DetectionProbability: 0.9000
                ClutterDensity: 1.0000e-06

                  OOSMHandling: 'Terminate'

                    TrackLogic: 'History'
         ConfirmationThreshold: [2 3]
             DeletionThreshold: [5 5]
              HitMissThreshold: 0.2000

           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: [0.7409 0.1530 0.1060]
                TrackLogic: 'History'
           TrackLogicState: [1 1 0 0 0]
               IsConfirmed: 1
```

```
                  IsCoasted: 0
              IsSelfReported: 1
            ObjectAttributes: [1x1 struct]
```

```
disp(info)
```

```
       OOSMDetectionIndices: [1x0 uint32]
    TrackIDsAtStepBeginning: 1
           UnassignedTracks: [1x0 uint32]
      UnassignedDetections: [1x0 uint32]
                CostMatrix: 13.8823
                  Clusters: {[1x1 struct]}
        InitializedTrackIDs: [1x0 uint32]
            DeletedTrackIDs: [1x0 uint32]
          TrackIDsAtStepEnd: 1
            ClassCostMatrix: -0.1823
```

Display the cluster information.

```
disp(info.Clusters{:})
```

```
         DetectionIndices: 1
                  TrackIDs: 1
          ValidationMatrix: [1 1]
               SensorIndex: 1
                 TimeStamp: 1
    MarginalProbabilities: [2x1 double]
               Likelihood: [2x2 double]
         ClassLikelihood: [2x2 double]
```

Algorithms

expand all

Tracker Logic Flow

Detection Class Fusion for `trackerJPDA`

Feasible Joint Events

References

[1] Fortmann, T., Y. Bar-Shalom, and M. Scheffe. "Sonar Tracking of Multiple Targets Using Joint Probabilistic Data Association." *IEEE Journal of Ocean Engineering.* Vol. 8, Number 3, 1983, pp. 173-184.

[2] Musicki, D., and R. Evans. "Joint Integrated Probabilistic Data Association: JIPDA." *IEEE transactions on Aerospace and Electronic Systems.* Vol. 40, Number 3, 2004, pp 1093-1099.

[3] 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.

Extended Capabilities

C/C++ Code Generation

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

Version History

expand all

Introduced in R2019a

> **R2023a:** Fuse detection classification information

See Also

Functions

`correctjpda` | `jpdaEvents` | `getTrackPositions` | `getTrackVelocities` | `predictTracksToTime`

Objects

`objectDetection` | `trackingKF` | `trackingEKF` | `trackingUKF` | `trackingCKF` | `trackingIMM` | `trackingABF` | `trackHistoryLogic` | `objectTrack` | `staticDetectionFuser` | `trackerTOMHT` | `trackerGNN`

Blocks

Joint Probabilistic Data Association Multi Object Tracker