Integrated Topside (InTop)

FlexDAR Monostatic Tracker

Functional Design Description (FDD)

Revision Number

0.1

Revision Date

---



Prepared by:

AJ Rivera – NRL 5329

|  |
| --- |
| DISTRIBUTION D. Distribution authorized to Department of Defense and U.S. DoD contractors only. Other request for this document shall be referred to Office of Naval Research (Code 31), 875 N. Randolph Street, Arlington, VA 22203 or higher authority. |

"WARNING - This document contains technical data whose export is restricted by the Arms Export Control Act (Title 22, U.S.C. Sec. 2751 et seq.) or the Export Administration Act of 1979, as amended (Title 50, U.S.C., App 2401, et seq.). Violations of export laws are subject to severe criminal penalties. Disseminate in accordance with provisions of DoD Directive 5230.25."

DESTRUCTION NOTICE: Destroy by any method that will prevent disclosure of contents or reconstruction of the document."

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Description | Author |
| 0.1 | xxx | Baseline – Describes Tracker up to git commit 2e87473 | AJ Rivera |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1 SCOPE 5

2 LEXICON 6

3 Meta Data 7

3.1 Node Position Data Report 7

4 Data Representation 8

4.1 Measurement data format 8

4.2 Track Data Format 9

4.3 Tracker Data 10

5 Processing Description 12

5.1 Tracker Subsystem Overview 12

5.2 Tracker API and Controller 13

5.3 FlexDAR System, Tracker API, and Controller Interactions 14

5.4 Track Initiation 16

5.5 Track Prediction 19

5.6 Track Association & Update 21

5.7 Track Maintenance 29

5.8 Track Score Penalization 31

Messages 32

5.9 FlexDAR System to Tracker API 32

5.10 Tracker API to FlexDAR System 33

5.11 Tracker API to Tracker Controller 34

5.12 Tracker Controller to Tracker API 35

6 Functions Pseudocode 36

6.1 Summary 36

6.2 TwoPointDiffStateInit() 37

6.3 DeletePreTracks() 38

6.4 DeleteTracks() 39

6.5 CreatePreTracks() 40

6.6 PromotePretrack() 41

6.7 GetKalmanPredictFilterInputs() 42

6.8 SqrtCubatureKalmanPredict() 43

6.9 isTrackExceedingLimits() 44

6.10 gateObsToTrack () 45

6.11 computeLogLikelihoodRatio() 47

6.12 formClusters() 48

6.13 pairObs2Tracks() 49

6.14 sqrtCubatureKalmanUpdate () 50

6.15 Tria() 52

6.16 PromoteTrack() 53

6.17 CholSemiDef() 54

6.18 Cart2RuvCubature() 55

6.19 findRfTransParam() 56

6.20 ellips2CartEcef() 58

6.21 GetRangeRate() 59

6.22 CalcMixtureMoments() 60

7 Supporting Information (internal – not for release) 61

7.1 Variable Pd 61

7.2 Coordinate System (for LAT/LON) 61

7.3 Linear Algebra C++ Libraries 62

[Figure 1‑1 FlexDAR system block diagram 5](#_Toc461196997)

[Figure 5‑1 Tracker API/Controller Interactions 12](#_Toc461196998)

[Figure 5‑3 FlexDAR system, Tracker API and Controller Interactions 14](#_Toc461196999)

[Figure 5‑5 Pretrack promotion to tentative track 17](#_Toc461197000)

[Figure 5‑6 Track prediction process 20](#_Toc461197001)

[Figure 5‑7 An observation that gates with a track 23](#_Toc461197002)

[Figure 5‑8 Observation-to-track pairing process 27](#_Toc461197003)

[Figure 5‑9 GNN filter track update process 27](#_Toc461197004)

[Table 1 Tracker API and Controller interactions. 14](#_Toc461197005)

[Table 2 Track Initiation summary 17](#_Toc461197006)

[Table 6 Top-level functions 36](#_Toc461197007)

[Table 7 Low-level functions. 36](#_Toc461197008)

# SCOPE

This document is a functional design description for the FlexDAR Monostatic Tracker subsystem. Figure 1‑1 shows the Tracker Subsystem (“Radar DP/Tracking”) enclosed by a red box.

This version only covers functions required by the FlexDAR Build 3.3.



Figure ‑ FlexDAR system block diagram

# LEXICON

|  |  |  |
| --- | --- | --- |
| **Domain** | **Term** | **Description** |
| Processing | Detection | The output of a detector. Measurements associated with a detection may not be fully resolved in range and Doppler. |
| Contact | The final output of multi-CPI dwell with a fully resolved set of measurements in range and range-rate. |
| Monostatic Detection/Contact | A detection/contact that is the result of a monostatic configuration, where the receiver and transmitter are co-located. |
| Tracking | FlexDAR Tracker/Tracking System | A collection of algorithms that, when used together, produce a location and velocity estimate of a target. |
| Target state vector | A numerical description of the estimated location and velocity of a target. |
| Target state covariance matrix | A numerical description of the accuracy associated with the estimated target state vector. |
| Observation | A collective term that refers to both detections and contacts. |

# Meta Data

## Node Position Data Report

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Dimensions | Data Type | Description |
| myNode | Scalar | Node Position Data | My Node. |
| otherNodes[] | Depends | Node Position Data | Other Nodes. |

### Node Position Data Fields

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Dimensions | Data Type | Description |
| nodeId | Scalar | Unsigned long | Node ID. |
| latRads | Scalar | Double | Geodetic latitude (WGS84) of node, in radians. |
| lonRads | Scalar | Double | Geodetic longitude (WGS84) of node, in radians. |
| boresightAzRads | Scalar | Double | Boresight azimuth angle of node, in ENU Spherical coordinates, in radians. |
| boresightElRads | Scalar | Double | Boresight elevation angle of node, in ENU Spherical coordinates, in radians. |
| heightMeters | Scalar | Double | Height of node above reference ellipsoid (WGS84), in meters. |
| velMps [ ] | 3 by 1 | Double | Velocity components of node, in meters per second, in ENU Cartesian coordinates. |

# Data Representation

## Measurement data format

Measurements (i.e. contacts and detections) shall be stored in an array of Measurement objects. Section 4.1.1 shows the properties associated with a Measurement data structure. Properties can be accessed by indexing the array as:

measurement[ithMeasurement].<property name>

For simplicity, array indices in this document start at 1.

### Measurement data structure

|  |  |  |  |
| --- | --- | --- | --- |
| Property Name | Dimensions | Data Type | Description |
| timeMilliSecs | Scalar | Unsigned long | Time the measurement was made, in milliseconds since Epoch. |
| zRuvRr[ ] | 4 by 1 | Double | Location in measurement space (range, u, v, and range-rate) at the receiver. |
| zCart[ ] | 3 by 1 | Double | Location in ECEF Cartesian coordinates. |
| SRRuvRr[ ][ ] | 3 by 3 | Double | Lower-triangular square root of the measurement noise covariance matrix associated with the measurement, in RUVRR coordinates at the receiver. |
| SRCart[ ][ ] | 4 by 4 | Double | Lower-triangular square root of the measurement noise covariance matrix associated with the measurement, in ECEF Cartesian coordinates. |
| rangeRateValid | Scalar | Boolean | Indicates if range-rate is valid. |
| isUsed | Scalar | Boolean | Indicates if measurement has been used by a track. |

## Track Data Format

Track data shall be stored in an array of Track objects. Each track object shall have the properties shown in Section 4.2.1 and can be accessed by indexing the array as:

track[nthTrack].<property name>

The following table shows the name, dimensions, data type, and description for the properties in a track data structure:

### Track Data Structure

|  |  |  |  |
| --- | --- | --- | --- |
| Property Name | Dimensions | Data Type | Description |
| id | Scalar | Unsigned long | Track identifier. |
| score | Scalar | Double | Track score. |
| timeMilliSecs | Scalar | Unsigned long | The time the target state and state covariance matrix was updated, in milliseconds since Epoch. |
| x[ ] | 6 by 1 | Double | Target state (i.e. position and speed), in ECEF Cartesian coordinates. |
| SPCart[ ][ ] | 6 by 6 | Double | Lower-triangular square root of the state covariance matrix at time timeMilliSecs. |
| predTimeMilliSecs | Scalar | Unsigned long | The time of the state prediction, in milliseconds since Epoch. |
| xPred[ ] | 6 by 1 | Double | Predicted target state at time predictedTimeMsec. |
| SPPredCart [ ][ ] | 6 by 6 | Double | Predicted lower-triangular square root of the track covariance matrix at time predTimeMilliSecs. |
| wasUpdatedFlag | Scalar | bool | Indicates if track was updated. |
| firmTrkFlag | Scalar | bool | Indicates if the track is firm. |

## Tracker Data

Tracker data shall comprise of the firm, tentative, and preliminary tracks lists, and tracker-specific metadata (i.e. design parameters). The tracker data shall be stored in a Tracker data structure. The properties can be accessed as:

tracker.<property name>

If, for example, the state vector of the nth firm track is wanted, it can be accessed as:

tracker.firmTracksList[nth].state

Properties marked with an asterisk (\*) shall be stored in a XML document and loaded during tracker initialization. The following table shows the name, dimensions, data type, and description for the properties in a Tracker data structure.

### Tracker Data Structure

|  |  |  |  |
| --- | --- | --- | --- |
| Property Name | Dimensions | Data Type | Description |
| maxNumFirmTracks\* | Scalar | Unsigned long | Maximum number of firm tracks allowed. |
| maxNumTentativeTracks\* | Scalar | Unsigned long | Maximum number of firm tracks allowed. |
| maxNumPreTracks\* | Scalar | Unsigned long | Maximum number of preliminary tracks allowed. |
| firmTrackList[ ] | maxNumFirmTracks | Track | List of firm tracks. |
| tentativeTrackList[ ] | maxNumTentativeTracks | Track | List of tentative tracks. |
| preTrackList[ ] | maxNumPreTracks | Measurement | List of preliminary tracks. |
| maxAccelerationMpsSqrd\* | Scalar | Double | Maximum allowable acceleration for a target, in meters per squared seconds. |
| gateBoundMeters\* | Scalar | Double | Maximum allowable track gate size in Cartesian coordinates. |
| maxSpeedMps\* | Scalar | Double | Maximum allowable track speed. |
| termTracksFlag\* | Scalar | Boolean | Indicates whether tracks that exceed the maximum allowable track gate in any dimension and/or the maximum allowable track speed should be terminated. |
| trackIdCounter | Scalar | Unsigned long | Track identifier number generator. |
| gamma4D | Scalar | Double | Threshold corresponding to a 99.97% probability that the gate contains the true observation (used when range-rate is valid) |
| gamma3D | Scalar | Double | Threshold corresponding to a 99.97% gate probability that it contains the true observation (used when range-rate is invalid) |
| gamma1D | Scalar | Double | Threshold for the 99.97% confidence region of a 1 degree of freedom chi-square distribution |
| fifthOrderCubaturePoints6D [ ] | 76 by 1 | Double | 6 dimensional fifth-order cubature points |
| fifthOrderCubatureWeights6D [ ] | 76 by 1 | Double | 6 dimensional fifth-order cubature weights |
| fifthOrderCubaturePoints3D [ ] | 14 by 1 | Double | 6 dimensional fifth-order cubature points |
| fifthOrderCubatureWeights3D [ ] | 14 by 1 | Double | 6 dimensional fifth-order cubature weights |
| lambda\* | Scalar | Double | Expected clutter density for a scan (false alarms per meters^2\* msin^2/seconds) |
| Pd | Scalar | Double | Probability of detection. |
| PterminatingTrueTrack\* | Scalar | Double | Probability of terminating a true track. Used when computing firm and tentative track promotion/deletion thresholds. |
| PdeclaringFalseTrack\* | Scalar | Double | Probability of declaring a false track. Used when computing firm and tentative track promotion/deletion thresholds. |
| firmTrackLowerScoreThreshold | Scalar | Double | Score threshold for terminating firm tracks. |
| tentativeTrackLowerScoreThreshold | Scalar | Double | Score threshold for terminating tentative tracks. |
| tentativeTrackUpperScoreThreshold | Scalar | Double | Score threshold for promoting tentative tracks. |

# Processing Description

## Tracker Subsystem Overview

The FlexDAR Tracker subsystem shall process observations into firm tracks, and report these tracks to other systems. It shall consist of the following components:

* **Tracker API**: responsible for all interactions with the FlexDAR system.
* **Tracker Controller**: responsible for all interactions with the Tracker API, and for scheduling the data processing functions (Prediction, Association & Update, Maintenance, and Initiation).
* **Track Prediction**: responsible for predicting tracks to the appropriate time.
* **Track Association**: responsible for performing the observation-to-track association.
* **Track Update**: responsible for performing track updates.
* **Track Maintenance**: responsible for deleting and promoting tracks based on the track score.
* **Track Initiation**: responsible for initiating tracks.

The following diagram shows the components of the FlexDAR Tracker:

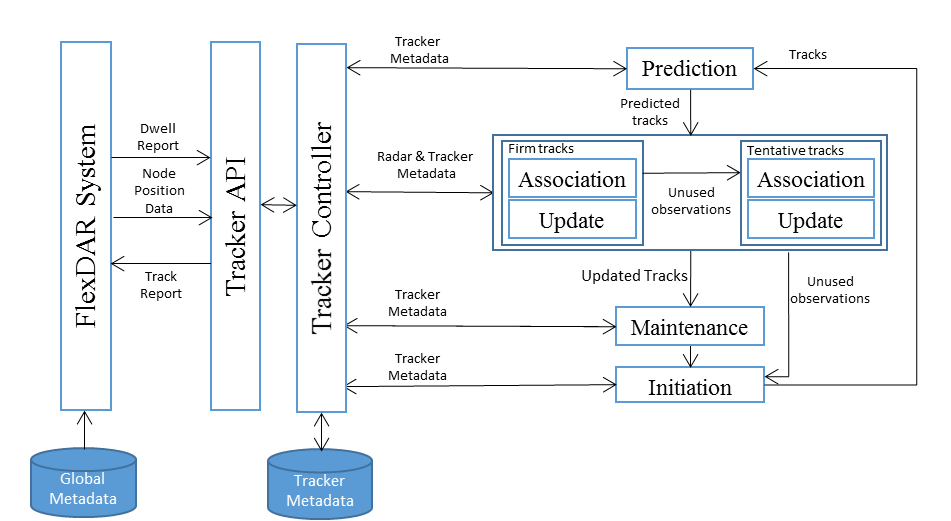


Figure 5‑1 Tracker API/Controller Interactions

The tracker shall use a three-level track initiation process by which preliminary tracks can be promoted to tentative tracks, and tentative tracks can be promoted to firm tracks. The criteria for promoting or deleting firm and tentative tracks is described in the Maintenance section. The criteria for promoting preliminary tracks is described in the Initiation section.

The tracker shall use a square root Kalman filter for track prediction, and a square root Cubature Kalman filter for track update. Both Kalman filters shall employ a First-Order Nearly-Constant Derivative target motion model that uses Discretized Continuous White-Noise Acceleration. The tracker shall update tracks using one of the following data association filters: Global Nearest Neighbor (GNN), or Probabilistic Data Association (PDA) filter. The criteria for deciding which filter to use is described in the Update section.

## Tracker API and Controller

The FlexDAR Tracker API shall be responsible for all interactions with the FlexDAR system. It shall be able to receive the following messages

* Search Dwell Report message: contains the list of contact for a search dwell,
* Track Dwell Report message: contains the list of contact for a track dwell.

Also, it shall send the following message:

* Track Report message: contains a list of new, deleted, and updated firm tracks,

The content of the messages is described in detail in the Messages section.

The Tracker API shall be initialized when the FlexDAR system is initialized. The Tracker API can then, upon request, initialize the Tracker Controller, which loads the tracker and global metadata. When the Tracker API receives a message from the FlexDAR system, it shall convert it to a format suitable for consumption by the Tracker Controller.

The Tracker Controller shall execute the Track Prediction process when a Dwell Report Message is received. Once the Tracker finishes processing the current dwell, the Tracker Controller shall produce a Track Report. This process shall then be repeated for every Dwell Report received.

## FlexDAR System, Tracker API, and Controller Interactions

The following diagram shows the FlexDAR system, Tracker API, and Tracker Controller interactions:

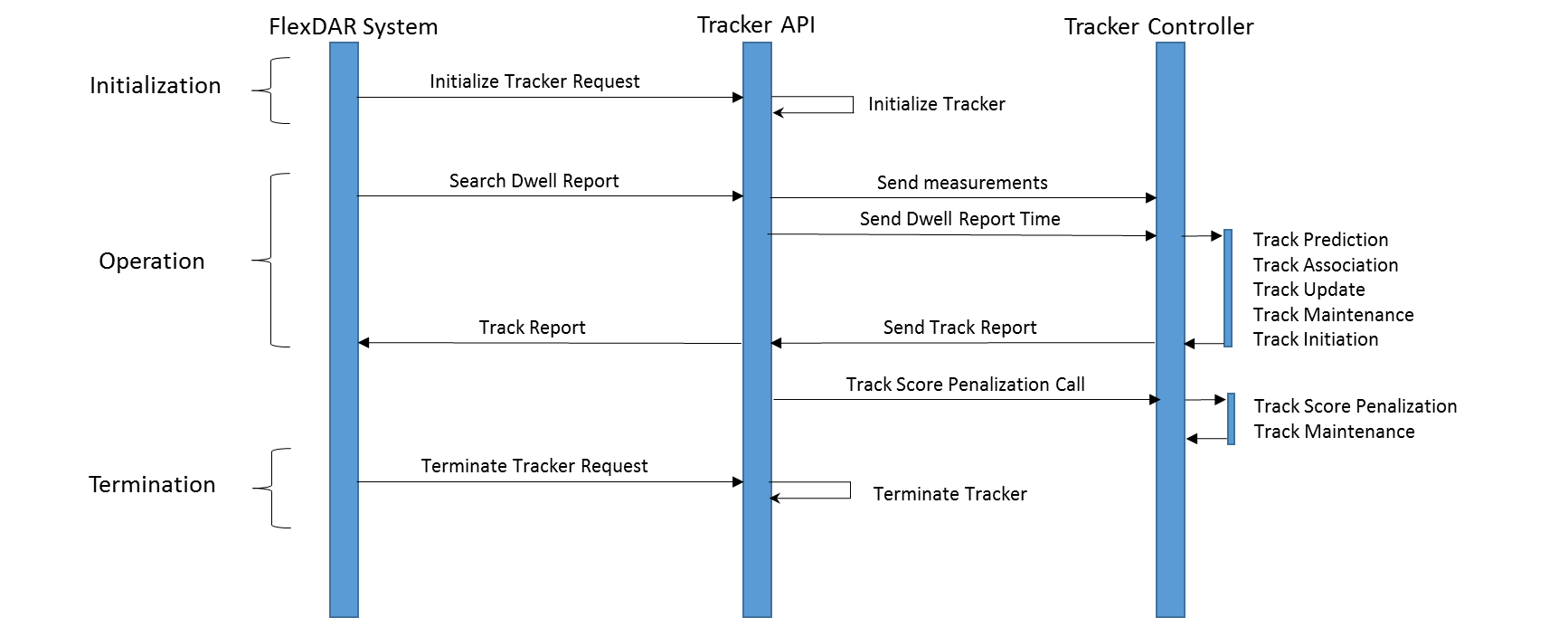


Figure ‑ FlexDAR system, Tracker API and Controller Interactions

API interactions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Area | From | To | Event | Description |
| Initialization | FlexDAR System | Tracker API | Initialize Tracker Request | This message triggers the Tracker API to create a tracker instance. |
| Termination | FlexDAR System | Tracker API | Terminate Tracker Request | This message triggers the Tracker API to delete the tracker instance. | |
| Operation | FlexDAR System | Tracker API | Search Dwell Report | This report contains a list of detections and associated data. | |
| Operation | Tracker API | FlexDAR System | Track Report | This report contains a list of deleted, added, and updated firm tracks. | |

Table Tracker API and Controller interactions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Area | From | To | Event | Description |
| Initialization | Tracker API | Tracker Controller | Initialize Tracker | This message initializes a Tracker Controller. |
| Termination | Tracker API | Tracker Controller | Terminate Tracker | This message triggers the termination of the Tracker Controller. |
| Operation | Tracker API | Tracker Controller | Send measurements | Sends observations in a format suitable for the tracker. |
|  |  |  |  |  |
| Operation | Tracker Controller | Tracker API | Send Track Report | This message sends new, updated, and deleted firm track data to the Tracker API. |
| Operation | Tracker API | Tracker Controller | Track Score Penalization Call | This message calls the track score penalization routines. |

## Track Initiation

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| observationList [ ] | variable | Measurement | List of observations from the dwell report. |
| preTrackList [ ] | maxNumPretracks | Measurement | List of preliminary tracks |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks, including new ones. |
| preTrackList [ ] | maxNumPretracks | Measurement | Updated list of preliminary tracks |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| dwellReportTimeMilliSecs | Scalar | Unsigned long | Dwell report time, in milliseconds since Epoch. |
| maxSpeedMps | Scalar | Double | Maximum allowable track speed. |
| maxAccelerationMpsSqrd | Scalar | Double | Maximum expected acceleration for a target, in meters per squared seconds. |
| maxNumTentativeTracks | Scalar | Unsigned long | Maximum number of firm tracks allowed. |
| trackIdCounter | Scalar | Unsigned long | Track identifier number generator. |
| gamma1D | Scalar | Double | Threshold for the 99.97% confidence region of a 1 degree of freedom chi-square distribution. |

Processing Description

The Initiation block shall be the first step in the three-level track initiation process. After being commanded by the Tracker Controller, the Initiation block shall retrieve the list of observations from the current dwell and all pretracks.

If there are no pretracks in the volume, the Initiation block shall form a pretrack for each one of the unused (i.e. unassociated) observations.

If one or more pretracks exist, for each one a gate shall be set up based on the assumed maximum target speed, such that if there is a target that gave rise to the pretrack in the previous scan, the observation from it in the current scan will fall in the gate with nearly unity probability. A pretrack shall be promoted to tentative track if an observation that falls in its gate passes a series of Doppler velocity tests. Each observation that falls in a pretrack’s gate and passes the required tests shall give rise to a tentative track. Observations with an invalid range-rate will not be subject to the tests. When an observation associates with a pretracks, it shall be marked as used. An observation that does not associate with any of the existing pretracks shall form a new pretrack.

Pretracks that have been given sufficient opportunities to associate with an observation and do not do so shall be deleted by a pretrack removal routine.

The table below summarizes all possible scenarios:

Table Track Initiation summary

|  |  |
| --- | --- |
| Scenario | Result |
| 0 pretracks  and  0 observations | No action |
| >=1 pretrack  and  0 observations | No action |
|
|
|
| 0 pretracks  and  >=1 observation | Each observation becomes a pretrack |
| >=1 pretrack  and  >= 1 observation | Check for associations |
| Each observation that associates with a pretrack forms a tentative track |
| Each unused observation becomes a new pretrack |

The goal of the Doppler velocity tests are to reduce the number of unlikely associations. The tests that an observation must pass are:

1. The observation’s range-rate must be less or equal than the maximum allowable speed for a target. This test shall be computed as:
2. The acceleration of the target must be less or equal than the maximum allowable acceleration for a target. The tests shall be computed as:

For each new tentative track, a Track object shall be created and assigned an identification number, time, score, state, and covariance matrix. A tentative track shall always be assigned an initial score of 0. The predicted state and predicted covariance matrix shall also be set to 0. The state and square-root covariance matrix shall be initialized by the Two-Point Differencing Algorithm using the associated observation and pretrack data as follows:

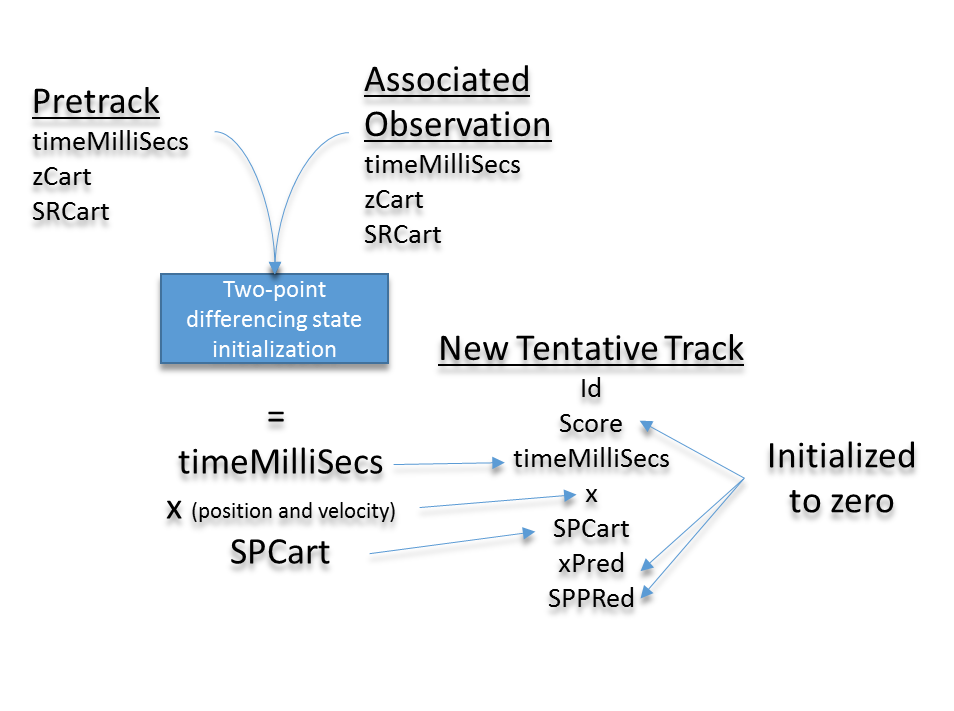


Figure ‑ Pretrack promotion to tentative track

Pseudocode

**GET unused observations list**

**GET pretrack list**

**DO DOPPLER TEST #1 for all unused observations. MARK observations that PASS the test.**

**IF** **number of pretracks == 0 AND number of unused observations > 0**

**CALL createPretracks(unused observations, preTrackList)**

**ELSEIF** **number of pretracks > 0 AND number of ungated observations > 0**

**FOR all pretracks**

**COMPUTE deltaTimeMsec= observation’s time - pretrack’s time**

**COMPUTE maximum possible distance possible given deltaTimeMsec and the maximum speed.**

**//check if observation gates with current pretrack**

**IF** **distance < maximum possible distance**

**DO DOPPLER TEST #2 using Pretrack and Observation data**

**IF Pass TEST #2 and both range-rates are valid**

**MARK observation as used**

**promotePretrack(pretrack, observation)**

**ELSEIF range-rates are NOT valid**

**MARK observation as used**

**promotePretrack(pretrack, observation)**

**ENDIF**

**ENDIF**

**ENDFOR**

**//unused observations become pretrack**

**CALL createPretracks(unused observations)**

**ENDIF**

## Track Prediction

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| firmTrackList [ ] | maxNumFirmTracks | Track | List of firm tracks |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks |
| dwellReportTimeMilliSecs | Scalar | Unsigned long | Dwell report time, in milliseconds since Epoch. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| firmTrackList [ ] | maxNumFirmTracks | Track | List of firm tracks with predicted state vector and covariance matrix |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks with predicted state vector and covariance matrix |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| termTracksFlag | Scalar | Bool | Flag that indicates whether tracks that exceed the maximum allowable track gate in any dimension and/or the maximum allowable track speed should be terminated. |
| gamma1D | Scalar | Double | Threshold for the 99.97% confidence region of a 1 degree of freedom chi-square distribution. |
| gateBoundMeters | Scalar | Double | Maximum allowable track gate size in Cartesian coordinates. |
| maxSpeedMps | Scalar | Double | Maximum allowable track speed. |

Processing Description

The Prediction block shall predict all firm and tentative tracks to the time of the Dwell Report so that the state estimates are at the correct time and can be associated with the observations. A square root Kalman filter shall be used to compute the track predicted state and covariance matrix. The filter shall use the track state and state covariance matrix, and a First-Order Nearly-Constant Derivative target motion model (i.e. state transition matrix) that uses Discretized Continuous White-Noise Acceleration (i.e. process noise matrix).

Once tracks are predicted forward, the Prediction block shall delete tracks whose inaccuracy and/or speed has exceed the acceptable limit (i.e. gate boundary and maximum speed). The following diagram depicts this process:

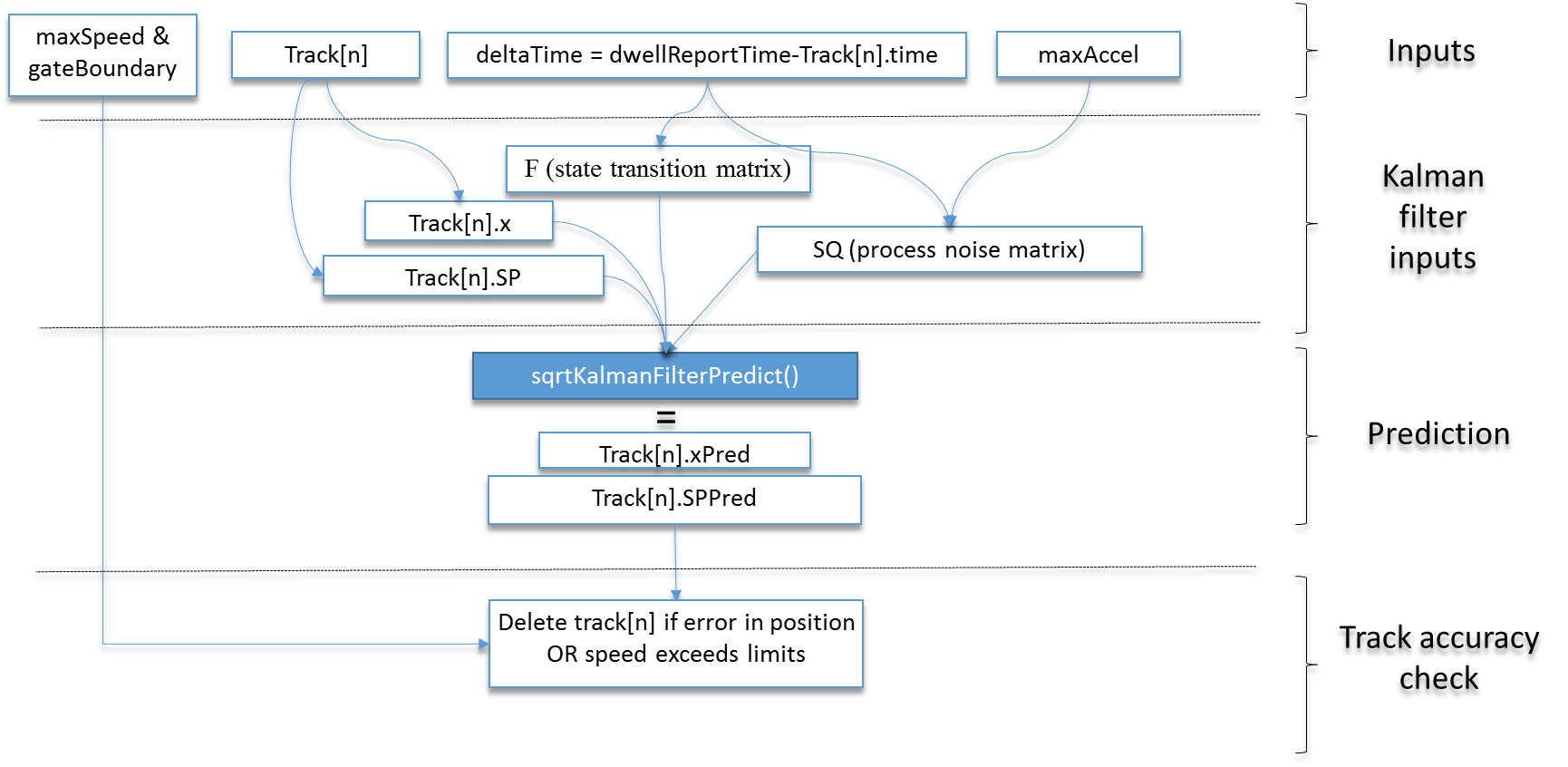


Figure ‑ Track prediction process

Pseudocode

**FOR** **all tentative and firm tracks**

**COMPUTE deltaTimeSec=1E-3\*(dwellReportTimeMilliSecs-track.timeMsec)**

**CALL FPolyKal(deltaTimeSec, 1) to get F**

**CALL processNoiseSuggest(“PolyKal-ROT”, T, maxAccelMpsSqrd) to get q**

**CALL QPolyKal(deltaTimeSec, 1, q) to get Q**

**CALL sqrtDiscKalmanFilterPredict(track.x, track.SP, F, cholSemiDef(Q)) to get track.xPred and track.SPPred**

**COMPUTE track.predictedTimeMsec=dwellReportTimeMilliSecs**

IF termTracksFlag AND isTrackExceedingLimits (track.x, track.SPPred)

**DeleteTracks(track)**

ENDIF

ENDFOR

## Track Association & Update

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| observationList [ ] | maxNumPreTracks | Measurement | List of observations in current scan |
| firmTrackList [ ] | maxNumFirmTracks | Track | List of firm tracks |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| observationList [ ] | maxNumPreTracks | Measurement | List of measurement with updated “used” property. |
| firmTrackList [ ] | maxNumFirmTracks | Track | List of firm tracks. Tracks that gated with the transmit and receive beams have updated the state vector and covariance matrix |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks. Tracks that gated with the transmit and receive beams have updated the state vector and covariance matrix |

Meta Data Needed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Name | Dimensions | Data Type | Description |
| Tracker | gamma4D | Scalar | Double | Threshold corresponding to a 99.97% probability that the gate contains the true observation (used when range-rate is valid) |
| gamma3D | Scalar | Double | Threshold corresponding to a 99.97% gate probability that it contains the true observation (used when range-rate is invalid) |
| fifthOrderCubaturePoints6D [ ] | 76 by 1 | Double | 6 dimensional fifth-order cubature points |
| fifthOrderCubatureWeights6D [ ] | 76 by 1 | Double | 6 dimensional fifth-order cubature weights |
| lambda | Scalar | Double | Expected clutter density for a scan (false alarms per meters^2\* msin^2/seconds) |
| Pd | Scalar | Double | Probability of detection. Used when computing the missed detection cost. |
| maxAccelerationMpsSqrd | Scalar | Double | Maximum expected acceleration for a target, in meters per squared seconds. |
| dwellReportTimeMilliSecs | Scalar | Unsigned long | Dwell report time, in milliseconds since Epoch. |
| Global | myLatRads | Scalar | Double | Geodetic latitude (WGS84) of my node, in radians. |
| myLonRads | Scalar | Double | Geodetic longitude (WGS84) of my node, in radians. |
| myHeightMeters | Scalar | Double | Height of node above reference ellipsoid (WGS84), in meters. |
| myBoresightAzRads | Scalar | Double | Boresight azimuth angle of my node, in ENU Spherical coordinates and radian units. |
| myBoresightElRads | Scalar | Double | Boresight elevation angle of my node, in ENU Spherical coordinates and radian units. |
| myVelMps [ ] | 3 by 1 | Double | Velocity components of my FlexDAR node, in meters per second, in ENU Cartesian coordinates. |

Processing Description

The Association & Update block is a two step process. It shall:

* Perform the observation-to-track associations, and
* Update tracks according to the pairings and data association filter.

### Observation-to-Track Association

The first step in determining if an observation gates with a track shall be to establish a validation region around the track predicted state. Gating is a technique for eliminating unlikely associations. For an observation to gate with a track, it shall pass a chi-square test performed in measurement space if the range-rate is available or in RUV space if the range-rate is not. The threshold for the test shall be given by the 99.97% confidence region of a chi-square distribution of 4 degrees of freedom if the range-rate is not available, or 3 degrees of freedom otherwise.

Since a track’s state vector and state covariance matrix is stored in ECEF Cartesian coordinates, and the observation-to-track association must occur in measurement or RUV space, a Cartesian to RUVRR coordinate conversion is necessary. The Gaussian cubature integration technique shall be used to perform the coordinate conversions.

Any observation that falls in the region shall be marked as used, and its log-likelihood ratio computed as:

where : **z** and **zPred** are 4 by 1 vectors in RUVRR coordinates (or 3x1 in RUV coordinates if range-rate is invalid),

**S** is the innovation covariance matrix in RUV coordinates

The tracks and used observations shall be grouped into clusters for later processing. Clusters shall be defined as a group of 1 or more tracks that have at least 1 gated observation in common, and shall be processed individually. If a track does not associate with an observations, it shall form a cluster with no observations. The table below provides a description of all possible scenarios when forming clusters. Note this description shows a 2D case, but can be extended to 4D:

An observation in the validation region

**X**

Predicted location

Current location

Validation region boundary

Figure ‑ An observation that gates with a track

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Result** | **Description** |
| 0 observations gates with 1 track. | 1 cluster is created and has 0 observations and 1 track. |  |
| 1 observation gates with 1 track. | 1 cluster is created and has 1 observation and 1 track. | **X** |
| 2 or more observations gate with 1 track. | 1 cluster is created and has 2 observations and 1 track. | **X**  **X** |
| 1 observation gates with n tracks (n=2 in this example) | 1 cluster is created and has 2 observation and n tracks. | **X** |
| m observations gate with n tracks.  (m=n=2 in this example) | 1 cluster is created and has m observations and n tracks. | **X**  **X**  **OR**  **X**  **X** |
| 2 observations gate with track 1 and 2; 1 observation gates with track 3. | 2 Clusters created:  Cluster 1 has Track 1 and the top two observations.  Cluster 2 has Track 3 and the bottom observation. |  |

### Track update

Once clusters are created, they shall be processed individually. A routine that assigns to each track -in the cluster- the observation that globally maximizes the log-likelihood ratio shall be used. During the global maximum search process, the missed detection hypothesis shall be treated as an additional observation. In some cases, an observation can initially gate with a track, and thus be marked as used, but when the log likelihood ratio is compared to the cost of a missed detection, the missed detection hypothesis is a better assignment in a likelihood sense. In such cases, the observation **isUsed** property shall be reverted back to unused.

The following figure shows the process for gating observations with the tracks, calculating the log-likelihood ratio for observations that track to a gate, cluster creation, and observation-to-track assignment:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Observation-to-track association:   |  |  |  |  | | --- | --- | --- | --- | |  | o1 | o2 | o3 | | T1 | gates | did not gate | gates | | T2 | gates | gates | did not gate |   1 Cluster formed: it has 2 tracks (T1 and T2) and 3 observations (o1, o2, and o3).  Calculated natural logarithm of missed detection cost = -1.2 = ln(1-Pd)  Calculated log-likelihood ratios:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | o1 | o2 | o3 | missed detection | | T1 | 4 | -Inf | -3 | -1.2 | | T2 | 2 | -2 | -Inf | -1.2 |   Assignments after finding the global maximum of the log-likelihood ratios for each track:  o1 assigned to T1  missed detection assigned to T2   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | o1 | o2 | o3 | missed detection | | T1 | 4 | did not gate | -3 | -1.2 | | T2 | 2 | -2 | did not gate | -1.2 | |  |

Figure ‑ Observation-to-track pairing process

Once the assignments have been calculated, the tracker shall update each track individually according to Global Nearest Neighbor filter. The following diagram shows the process used to update a track when using a GNN filter:

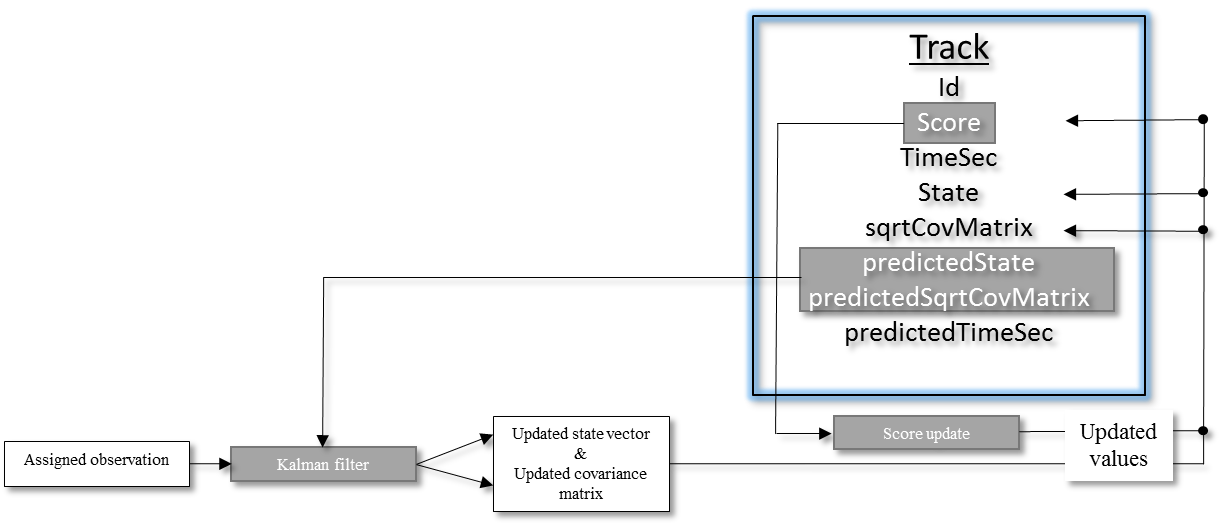


Figure ‑ GNN filter track update process

The GNN filter updates a track with the assigned observation (there can only be one assigned observation per track). The assigned observation shall be the observation that globally maximizes the log-likelihood ratio. If the assigned observation is the missed detection hypothesis, the GNN filter shall set the updated state as the predicted state.

The score shall updated as:

Pseudocode

**GET tracks**

**GET unused observations**

**//Form clusters**

**FOR all unused observations**

**CALL gateObsToTrack(track, observation) to get distance, covMatrixRuv and obsGatesToTrack**

**IF obsGatesToTrack**

**mark observation as used**

**llrMatrix[nthTrack][ithObservation] =**

**computeLogLikelihoodRatio (distance, covMatrixRuv, lambda, Pd)**

**ENDIF**

**ENDFOR**

**CALL formClusters(llrMatrix)**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CLUSTER PROCESSING (UPDATE TRACKS) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**FOR all clusters**

**//Get log-likelihood matrix for this cluster (a subset of llrMatrix)**

**//Perform observation-to-track pairing**

**CALL pairObs2Tracks(clusterLikelihoodRatioMatrix) to get observation2TracksAssignments**

**//Mark unassigned observations as unused**

**FOR all observations**

**IF observation2TracksAssignments[i] == no track assigned**

**Mark observation as unused**

**ENDIF**

**ENDFOR**

**// Update each track in current cluster**

**FOR all tracks in cluster**

**Track.time =dwellReportTimeMilliSecs**

**IF observation is not the missed detection hypothesis**

**CALL sqrtCubatureKalmanUpdate(track, observation) to get track.x, track.SP**

**Track.score=track.score+LogLikelihoodRatio**

**ENDIF**

**ENDFOR //endfor all tracks in a cluster**

**ENDFOR //endfor all clusters**

## Track Maintenance

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| firmTrackList [ ] | maxNumFirmTracks | Track | List of firm tracks |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| firmTrackList [ ] | maxNumFirmTracks | Track | List of firm tracks, including new ones (promoted tentative tracks) |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks that were not deleted after score evaluation |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| firmTrackLowerScoreThreshold | Scalar | Double | Score threshold for terminating firm tracks. |
| tentativeTrackLowerScoreThreshold | Scalar | Double | Score threshold for terminating tentative tracks. |
| tentativeTrackUpperScoreThreshold | Scalar | Double | Score threshold for promoting tentative tracks. |

The track maintenance block shall maintain tracks by testing their score against upper and lower thresholds. The criteria for promoting, keeping, or deleting a tentative track shall be:

*Track.score >: Promote tentative track to firm track*

*Track.score <: Delete tentative track*

The criteria for deleting a firm track shall be:

*Track.score : Delete firm track*

The upper and lower bounds shall be computed as:

*tentativeTrackUpperScoreThreshold =*

*<firm,tentative>TrackLowerScoreThreshold =*

To avoid long termination times, a firm track score shall always be limited to a maximum score of 0.

Pseudocode

**FOR all firm tracks**

**IF track.score > 0**

**COMPUTE Track.score=0**

**ELSEIF track.score < firmTrackLowerScoreThreshold**

**CALL deleteTracks(track)**

**ENDIF**

**ENDFOR**

**FOR all tentative tracks**

**IF track.score > tentativeTrackUpperScoreThreshold**

**COMPUTE promoteTrack(track)**

**ELSEIF track.score < tentativeTrackLowerScoreThreshold**

**CALL deleteTracks(track)**

**ENDIF**

**ENDFOR**

## Track Score Penalization

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| firmTrackList [ ] | maxNumFirmTracks | Track | List of firm tracks |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks |
| preTrackList[ ] | maxNumPreTracks | Measurement | List of preliminary tracks. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| firmTrackList [ ] | maxNumFirmTracks | Track | Updated firm track list. |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | Updated tentative track list. |
| preTrackList[ ] | maxNumPreTracks | Measurement | Updated preliminary track list. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| Pd | Scalar | Double | Probability of detection. Used when computing the missed detection cost. |
| tentativeTrackLowerScoreThreshold | Scalar | Double | Score threshold for terminating tentative tracks. |
| tentativeTrackUpperScoreThreshold | Scalar | Double | Score threshold for promoting tentative tracks. |

The Track Score Penalization block shall penalize tracks that have been given sufficient time to associate with observations. These are tracks that have not been updated in the last ‘cycle’. Equally, pretracks that have been given sufficient time to association with observations are dropped. The score penalization factor is given by:

where Pd is the probability of detection of a track.

Pseudocode

**//Penalize track that have not been updated in the last cycle.**

**missedDetectionCost = log(1-Pd)**

**FOR all tracks**

**IF track.wasUpdatedFlag == false**

**Track.score = track.score+missedDetectionCost**

**ENDIF**

**ENDFOR**

**//Drop all the pretracks that have not been promoted to tentative track in the last cycle.**

**CLEAR preTrkList[]**

# Messages

## FlexDAR System to Tracker API

### Search Dwell Report

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Dimensions | Data Type | Description |
| dwellId | Scalar | Unsigned Integer | ID of dwell. This ID should be unique within any reasonable timeframe. |
| dwellType | Scalar | Enumeration | Always DWELL\_TYPE\_SEARCH. |
| time | Scalar | UTC time | Start time of the dwell. 1nS precision. |
| nCpis | Scalar | Integer | Number of CPIs (always 4). |
| nRecords | Scalar | Integer | Number of contact records |
| txBeamId | Scalar | Unsigned long | Transmit beam Id used. |
| contact[ ] | nRecords | Contact Record | List of contact records. |

#### Contact Record Fields

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Dimensions | Data Type | Description |
| rM | Scalar | Real | Fine range measurement (m). |
| vMps | Scalar | Real | Unambiguous velocity measurement (m/s). |
| u | Scalar | Real | U (of U-V space) measurement. |
| v | Scalar | Real | V (of U-V space) measurement. |
| ampDb | Scalar | Real | Amplitude (db relative to X?) |
| bgDb???? | Scalar | Real | Background level (db relative to X?) |
| iRcell | Scalar | Integer | Integer range bin. (Supporting info.) |
| vAmbMps[ ] | nCpis | Real | Ambiguous velocity measurements from dominant range-Doppler bin from detection beam. |
| dU | Scalar | Real | MLE horizontal offset from beam center in U-V space. (Supporting info.) |
| dV | Scalar | Real | MLE vertical offset from beam center in U-V space. (Supporting info.) |
| uBeam | Scalar | Real | U (of U-V space) of center of detection beam. (Supporting info.) |
| vBeam | Scalar | Real | V (of U-V space) of center of detection beam. (Supporting info.) |

### Initialize Tracker Request

This request signals the Tracker API to create a tracker instance. No fields are needed.

### Terminate Tracker Request

This request signals the Tracker API to delete the tracker instance. No fields are needed.

## Tracker API to FlexDAR System

### Track Report

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| firmTrackList [ ] | maxNumFirmTracks | Track | List of firm tracks. |

## Tracker API to Tracker Controller

### Initialize Tracker

This message instantiates a new Tracker object.

### Send Observations

This message sends the observation list to the tracker controller for processing.

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Dimensions | Data Type | Description |
| observationList[ ] | Depends | Measurement | List of observations for the Tracker Controller |

### Terminate Tracker

This request deletes the tracker object. No fields are needed.

## Tracker Controller to Tracker API

### Send Tracker State

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Dimensions | Data Type | Description |
| curNumFirmTracks | Scalar | Unsigned long | Current number of firm tracks in the tracker. |
| curNumTentativeTracks | Scalar | Unsigned long | Current number of tentative tracks in the tracker. |

### Send Track Report

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| deletedFirmTrackIds [ ] | Depends | Integer | Track IDs of deleted firm tracks. |
| newFirmTracksList [ ] | Depends | Track | List of new firm tracks. |
| udpatedFirmTracksList [ ] | Depends | Track | List of updated firm tracks. |

# Functions Pseudocode

## Summary

The following table shows the function allocation in terms of the data processing blocks:

Table Top-level functions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | **Initiation** | **Prediction** | **Association & Update** | **Maintenance** |
| deletePreTracks | X |  |  |  |
| createPretracks | X |  |  |  |
| promotePretrack | X |  |  |  |
| FPolyKal |  | X |  |  |
| processNoiseSuggest |  | X |  |  |
| QPolyKal |  | X |  |  |
| sqrtCubatureKalmanPredict |  | X |  |  |
| isTrackExceedingLimits |  | X |  |  |
| deleteTracks |  | X |  | X |
| promoteTrack |  |  |  | X |
| gateObsToTrack |  |  | X |  |
| computeLogLikelihoodRatio |  |  | X |  |
| formClusters |  |  | X |  |
| pairObs2Tracks |  |  | X |  |
| sqrtCubatureKalmanUpdate |  |  | X |  |

Table Low-level functions.

|  |  |
| --- | --- |
| **Additional functions** | **Used in** |
| twoPointDiffStateInit | promotePretrack |
| tria | sqrtCubatureKalmanPredict, sqrtCubatureKalmanUpdate,  cholSemiDef |
| kron | QPolyKal, FPolyKal |
| cholSemiDef | promotePretrack |
| cart2RuvCubature | gateObsToTrack |
| findRFTransParam | gateObsToTrack, sqrtCubatureKalmanUpdate |
| ellips2CartEcef | gateObsToTrack, sqrtCubatureKalmanUpdate |
| getRangeRate | sqrtCubatureKalmanUpdate |
| calcMixtureMoments | sqrtCubatureKalmanUpdate |

## TwoPointDiffStateInit()

Used in: PromotePretrack

Purpose: Initialize the state vector and covariance matrix of a track with two observations.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| deltaTSecs | Scalar | Double | Time delta between the measurements |
| oldPosCart [ ] | 3 by 1 | Double | Location in ECEF Cartesian coordinates of first measurement. |
| newPosCart [ ] | 3 by 1 | Double | Location in ECEF Cartesian coordinates of second measurement. |
| oldCovMatCart[ ][ ] | 3 by 3 | Double | Measurement noise covariance matrix associated with the first measurement, in ECEF Cartesian coordinates. |
| newCovMatCart[ ][ ] | 3 by 3 | Double | Measurement noise covariance matrix associated with the second measurement, in ECEF Cartesian coordinates. |
| q [ ][ ] | 6 by 6 | Double | Process noise |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| state [ ] | 6 by 1 | Double | Location in ECEF Cartesian coordinates. |
| P [ ][ ] | 6 by 6 | Double | Covariance matrix associated with the smoothed state, in ECEF Cartesian coordinates. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode:

**//positionIdx=the indices that point to the position elements in the vector or matrix**

**state[positionIdx] = newPosCart**

**state[velocityIdx] = (newPosCart-oldPosCart)/deltaTSecs**

**P[positionIdx][positionIdx] = newCovMatCart**

**P[positionIdx][velocityIdx] = newCovMatCart/deltaTSecs**

**P[velocityIdx][positionIdx] = newCovMatCart/deltaTSecs**

**P[velocityIdx][velocityIdx] = (oldCovMatCart+newCovMatCart)/deltaTSecs^2**

**//the bias due to the process noise**

**P[velocityIdx][velocityIdx] = P[velocityIdx][velocityIdx]+1/3\*q\*deltaTSecs**

## DeletePreTracks()

Used in: Track Initiation

Purpose: Delete a list of pretracks.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| pretrackList [ ] | maxNumPreTracks | Measurement | List of pretracks to be deleted |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No outputs |  |  |  |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode:

**FOR all tracks in pretrackList []**

**SET to empty**

**ENDFOR**

## DeleteTracks()

Used in: Prediction, Maintenance

Purpose: Delete firm or tentative tracks.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| trackList [ ] | if tentative tracks: maxNumTentativeTracks;  if firm tracks: maxNumFirmTracks | Track | List of tracks to be deleted. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No outputs |  |  |  |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode:

**FOR all tracks in trackList []**

**SET to empty**

**ENDFOR**

## CreatePreTracks()

Used in: Track Initiation

Purpose: Form a pretrack given an observation.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| ungatedObservationList [ ] | maxNumPreTracks | Measurement | List of observations that will create pretracks |
| preTrackList [ ] | maxNumPreTracks | Measurement | List of pretracks |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| preTrackList [ ] | maxNumPreTracks | Measurement | List of pretracks, including new ones. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode

**FOR every observation in ungatedObservationList []**

**IF number pretracks in preTrackList < maxNumPretracks**

**Append observation to preTrackList []**

**ENDIF**

**ENDFOR**

## PromotePretrack()

Used In: Track Initiation

Purpose: Promote a pretrack to tentative track.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| observation | Scalar | Measurement | Observation used in pretrack promotion to tentative track |
| pretrack | Scalar | Measurement | Pretrack to be promoted. |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks, including new one. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| maxAccelerationMpsSqrd | Scalar | Double | Maximum expected acceleration for a target, in meters per squared seconds. |

Pseudocode

**IF number tracks in tentativeTrackList < maxNumTentativetracks**

**//”new” represents second observation, and “old” the first one.**

**newTime=observation.timeMsec**

**newPosCart=observation.locationCart**

**newCovMatrixCart=observation.noiseCovMatrixCart**

**oldTime=pretrack.timeMsec**

**oldPosCart=pretrack.locationCart**

**oldCovMatrixCart=pretrack.noiseCovMatrixCart**

**T=newTime-oldTime**

**CALL processNoiseSuggest(“PolyKal-ROT”, T, maxAccelMpsSqrd) to get q**

**CALL twoPointDiffInit (T, oldPosCart,newPosCart, oldCovMatrixCart, newCovMatrixCart,q) to get xInitialized and PInitialized**

**//Append new tentative track to tentativeTrackList[] with following values in the data structure**

**trackId=trackIdCounter**

**timeMsec=observation.timeMsec**

**predictedTimeMsec=0**

**score=0**

**state=xInitialized**

**predictedState=0**

**sqrtCovMatrix=cholSemiDef(PInitialized)**

**predictedSqrtCovMatrix[:][:]=0**

**//increment counter**

**trackIdCounter++**

**ENDIF**

## FPolyKal()

Used In: Track Prediction

Purpose: Get the Transition matrix used in the Kalman filter.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| deltaTimeSec | Scalar | Double | Time difference, in seconds. |
| xDim | Scalar | Integer | The dimensionality of the target state |
| order | Scalar | Integer | The order of the filter |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| F [ ][ ] | xDim by xDim | Double | State transition matrix |

Pseudocode

**numDim = xDim/(order+1)**

**//First, create the matrix for 1D motion**

**numEl = order+1**

**F1 = zeros(numEl, numEl)**

**degList = transpose([0:order])**

**F1(1,:) = T.^degList./factorial(degList)**

**FOR curRow = 2:numEl**

**F1(curRow,curRow:numEl) = F1(1,1:(numEl-curRow+1))**

**ENDFOR**

**F = kron(F1, eye(numDim))**

## processNoiseSuggest()

Used In: Track Prediction

Purpose: Compute the scaling parameter for the process noise matrix.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| algorithm | Scalar | String | Algorithm |
| deltaTimeSec | Scalar | Double | Time difference, in seconds. |
| maxVal | Scalar | Double | Maximum bound used in the algorithm |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| q | Scalar | Double | Scaling parameter |

Pseudocode

**IF algorithm == “PolyKal-ROT”**

**q = (0.75\*maxVal)^2/T**

**ENDIF**

## QPolyKal()

Used In: Track Prediction

Purpose: Get the Process Noise matrix used in the Kalman filter.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| deltaTimeSec | Scalar | Double | Time difference, in seconds. |
| xDim | Scalar | Integer | The dimensionality of the target state |
| order | Scalar | Integer | The order of the filter |
| q | Scalar | Double | Scaling parameter |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| Q [ ][ ] | xDim by xDim | Double | Process Noise matrix |

Pseudocode

**numDim = xDim/(order+1)**

**q = ones(numDim,1)\*q**

**//First, create the matrix for 1D motion**

**numEl = order+1**

**sel = (numEl-1):-1:0**

**[colIdx,rowIdx] = meshgrid(sel,sel)**

**Q1=T.^(colIdx+rowIdx+1)./(factorial(colIdx).\*factorial(rowIdx).\*(colIdx+rowIdx+1))**

**Q = kron(Q1, diagonal(q))**

## kron()

Used In: Various functions

Purpose: Compute the Kronecker product.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| A[ ][ ] | m by n | Double | Matrix |
| B[ ][ ] | p by q | Double | Matrix |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| K[ ][ ] | m\*p by n\*q | Double | State transition matrix |

Pseudocode

**//This is an all-loop implementation of the Kronecker product**

**FOR i = 1 to m**

**FOR j = 1 to n**

**FOR k = 1 to p**

**FOR l = 1 to q**

**K(k+i\*p, l+j\*q) = A(i,j)\*B(k,l)**

**ENDFOR**

**ENDFOR**

**ENDFOR**

**ENDFOR**

## factorial()

Used In: Various functions

Purpose: Calculate the factorial of an integer

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| n | Scalar | Integer | Number to be factorialized |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| f | Scalar | Integer | Factorialized n (n!) |

Pseudocode

**IF (n=0 OR n=1)**

**f=1**

**ELSE**

**//Solve it recursively**

**factorial(n-1)**

**ENDIF**

## SqrtCubatureKalmanPredict()

Used In: Track Prediction

Purpose: Predict tracks forward using the square root cubature Kalman filter.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| stateVector [ ] | 6 by 1 | Double | Track state vector |
| sqrtCovMatrix [ ][ ] | 6 by 6 | Double | Lower-triangular square root of the track covariance matrix. |
| F[ ][ ] | 6 by 6 | Double | State transition matrix |
| SQ[ ][ ] | 6 by 6 | Double | Lower-triangular square root of the process noise covariance matrix. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| predictedState [ ] | 6 by 1 | Double | Predicted state vector |
| predictedSqrtCovMatrix[ ][ ] | 6 by 6 | Double | Predicted Lower-triangular square root of the track covariance matrix. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode

**predictedState = F \* state**

**predictedSqrtCovMatrix = tria( concatenate( F \* sqrtCovMatrix , sqrtProcessNoise ))**

## isTrackExceedingLimits()

Used In: Track Prediction

Purpose: Check if track exceeds the provided limits.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| stateVector [ ] | 6 by 1 | Double | Track state vector |
| sqrtCovMatrix [ ][ ] | 6 by 6 | Double | Lower-triangular square root of the track covariance matrix. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| trackExceedsLimits | Scalar | Bool | Flag that indicates if the track exceeds limits. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| gateBoundMeters[ ] | Scalar | Double | Maximum allowable track gate size in Cartesian coordinates. |
| maxSpeedMps | Scalar | Double | Maximum allowable track speed. |
| gamma1D | Scalar | Double | Threshold for the 99.97% confidence region of a 1 degree of freedom chi-square distribution |

**//Velocity**

**speedMps=sqrt(state[4]^2 + state[5]^2 + state[6]^2)**

**varianceRatioInX = sqrtCovMatrix[1][1]^2 / gateBoundMeters^2**

**varianceRatioInY = sqrtCovMatrix[2][2]^2 / gateBoundMeters^2**

**varianceRatioInZ = sqrtCovMatrix[3][3]^2 / gateBoundMeters^2**

**IF (varianceRatioInX > gamma1D OR**

**varianceRatioInY > gamma1D OR**

**varianceRatioInZ > gamma1D OR**

**speedMps > maxSpeedMps)**

**trackExceedsLimits=True**

**ELSE**

**trackExceedsLimits=False**

**ENDIF**

## gateObsToTrack ()

Used In: Track Association & Update

Purpose: Check if an observations gates with a track.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| track | Scalar | Track | A track. |
| observation | Scalar | Measurement | An observation. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| obsGatesWithTrack | Scalar | Bool | Flag that indicates if observation gates with a track. |
| distance | Scalar | Double | Mahalanobis distance squared (distance to center of the track) |
| covMatrixRuv [ ][ ] | 3 by 3 | Double | Covariance matrix in Measurement space (sum of measurement noise covariance matrix and track’s covariance matrix) |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| myNode | Scalar | Node Position Data | My Node. |
| fifthOrderCubaturePoints6D [ ] | 76 by 1 | Double | 6 dimensional fifth-order cubature points |
| fifthOrderCubatureWeights6D [ ] | 76 by 1 | Double | 6 dimensional fifth-order cubature weights |
| gamma4D | Scalar | Double | Threshold corresponding to a 99.97% probability that the gate contains the true observation (used when range-rate is valid) |
| gamma3D | Scalar | Double | Threshold corresponding to a 99.97% gate probability that it contains the true observation (used when range-rate is invalid) |

Pseudocode

**obsGatesToTrack=False**

**//Get ECEF Cartesian coordinates of my FlexDar node**

**txCartEcef=ellips2CartEcef(myNode)**

**rxCartEcef=ellips2CartEcef(myNode)**

**//Rotation matrix to go from state vector space (cartECEF) to Rx RUV space**

**transformMatrixCartEcefToRxRf=findRFTransParam(myNode)**

**//Get track’s predicted state and covariance matrix in Measurement space (RUV & Range-rate)**

**CALL Cart2RuvCubature(track.xPred, track.SPPred.SPPred, txCartEcef,rxCartEcef, transformMatrixCartEcefToRxRf,**

**fifthOrderCubaturePoints6D, fifthOrderCubatureWeights6D)**

**to get predictedStateRuv and predictedCovMatrixRuv**

**//Predicted measurement+track covariance**

**covMatrixRuv = predictedCovMatrixRuv + observation.sqrtCovMatrixRuv**

**IF isfinite(observation.locationRuv[4])**

**innovation = observation.locationRuv-predictedStateRuv**

**//The Mahalanobis distance squared**

**distance = transpose(innovation) \* inverse(S) \* innovation**

**IF (distance<=gamma4D)**

**obsGatesToTrack=True**

**ENDIF**

**ELSE**

**innovation = observation.locationRuv[1:3]-predictedStateRuv[1:3][1:3]**

**distance= transpose(innovation) \* inverse(S[1:3][1:3]) \* innovation**

**IF (distance<=gamma3D)**

**obsGatesToTrack=True**

**ENDIF**

**ENDIF**

## computeLogLikelihoodRatio()

Used In: Track Association & Update

Purpose: Compute the log-likelihood ratio.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| distance | Scalar | Double | Mahalanobis distance squared (distance to center of the track) |
| covMatrixRuv [ ][ ] | 3 by 3 | Double | Covariance matrix in Measurement space (sum of measurement noise covariance matrix and track’s covariance matrix) |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| llr | Scalar | Double | Log-likelihood ratio. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| lambda | Scalar | Double | Expected clutter density for a scan (false alarms per meters^2\* msin^2/seconds) |
| Pd | Scalar | Double | Probability of detection. |

Pseudocode

**llr = -(0.5\*distance + log( (lambda/Pd) \* sqrt(determinant(2\*pi\*covMatrixRuv)) )**

## formClusters()

Used In: Track Association & Update

Purpose: Create clusters given the track-to-observation pairings.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| llrMatrix [ ][ ] | numTracks by numObs | Double | log-likelihood ratio matrix. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| ClusterSet | Variable | - | A set of clusters. Each one has the track IDs and measurement idx in the cluster |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode:

**//Identify which observations are shared amongst tracks: In a new matrix called binaryMatrix with same //dimensions as llrMat, assign value of 1 to elements that are non-infinite ,and 0 for infinite.**

**SET binaryMatrix**

**//A cluster has at least 1 track and 0 or more observations. If observations gate with more than 1 track //(as indicated by the binaryMatrix), the tracks are part of the same cluster, including previously gated //observations.**

**clusterIdx = 1**

**FOR all tracks**

**IF num measurements that gated with the track == 0**

**//Form a new cluster**

**Cluster.trackId = current track ID**

**Cluster.measIdx = empty**

**//Add new cluster to cluster set**

**ClusterSet(clusterIdx) = Cluster**

**clusterIdx = clusterIdx+1**

**ELSEIF num measurements that gated with the track > 0**

**FOR all measurements that gated with the track**

**//Form a new cluster**

**Cluster.trackId = current track ID**

**Cluster.measIdx = current measurement idx**

**//Check if existing clusters in cluster set already have current track ID or meas //Idx**

**IF track ID and/or measurement idx already exist in cluster set**

**MERGE this cluster with the existing cluster set where track ID and/or measurement idx was found**

**ELSE**

**//Add new cluster to cluster set**

**ClusterSet(clusterIdx) = Cluster**

**clusterIdx = clusterIdx+1**

**END**

**END**

**END**

## pairObs2Tracks()

Used In: Track Association & Update

Purpose: Perform the observation-to-track association (pairings).

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| clusterLlrMatrix [ ][ ] | numTracksInCluster by numObsInCluster | Double | log-likelihood ratio matrix for a cluster. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| obs2TrackAssignments [ ] | numObsInCluster | Unsigned long | Observation to track assignments for a cluster. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| Pd | Scalar | Double | Probability of detection. |

Pseudocode

**//Compute missed detection cost**

**missedDetCost=log(1-Pd)**

**//Add new column to clusterLlrMatrix with the missedDetCost (every row (which represents a track) will have this value at the end)**

**Append missedDetCost to clusterLlrMatrix**

**//Perform global maximization**

**Find the global maximum for each observation, and save the row where it is.**

**//Each element in obs2TrackAssignments[] corresponds to the row where the assigned track is (e.g. observation #2 was assigned to track #1, thus obs2TrackAssignments[2]=1).**

**SET obs2TrackAssignments with saved values.**

## sqrtCubatureKalmanUpdate ()

Used In: Track Association & Update

Purpose: Update a track using the square root cubature Kalman filter.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| track | Scalar | Track | Track to be updated. |
| observation | Scalar | Measurement | Observation used be used to update the track. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| updatedState[ ] | 6 by 1 | Double | Updated target state. |
| updatedSqrtCovMatrix[ ][ ] | 6 by 6 | Double | Updated Lower-triangular square root of the track covariance matrix. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| myNode | Scalar | Node Position Data | My Node. |
| fifthOrderCubaturePoints6D [ ] | 76 by 1 | Double | 6 dimensional fifth-order cubature points |
| fifthOrderCubatureWeights6D [ ] | 76 by 1 | Double | 6 dimensional fifth-order cubature weights |

Pseudocode

**sqrtW=sqrt(fifthOrderCubatureWeights6D)**

**//Calculate the updated estimates**

**// Predicted cubature state points (see Matlab documentation for more info on bsxfun)**

**xPredPoints = bsxfun(@minus,track.SPPred\*fifthOrderCubaturePoints6D,track.xPred)**

**// Predicted, centered cubature state points**

**xPredCenPoints=bsxfun(@times,bsxfun(@minus,xPredPoints,xPred),sqrtW')**

**// Predicted cubature measurement points**

**zPredPoints=zeros(zDim,76)**

**//Get ECEF Cartesian coordinates of my FlexDar node**

**txCartEcef=ellips2CartEcef(myNode)**

**rxCartEcef=ellips2CartEcef(myNode)**

**//Rotation matrix to go from state vector space (cartECEF) to Rx RUV space**

**transformMatrixCartEcefToRxRf=findRFTransParam(myNode)**

**FOR curP=1:numCubPoints**

**z[1:3][:]=Cart2Ruv(xPredPoints[1:3][:], txCartEcef[1:3], rxCartEcef [1:3], transformMatrixCartEcefToRxRf)**

**//Compute the range rate.**

**z(4,:)=getRangeRate(xPredPoints, txCartEcef, rxCartEcef)**

**ENDFOR**

**// Measurement prediction (element-wise multiplication denoted by .\*)**

**zPred[1][:]= calcMixtureMoments(z[1][:], fifthOrderCubatureWeights6D)**

**zPred[2][:]= sum(z[2][:] .\* fifthOrderCubatureWeights6D)**

**zPred[3][:]= sum(z[3][:] .\* fifthOrderCubatureWeights6D)**

**zPred[4][:]= calcMixtureMoments(z[4][:], fifthOrderCubatureWeights6D)**

**// The innovation**

**innov=z-zPred**

**// Centered, predicted cubature measurement points**

**zPredCenPoints=bsxfun(@times,bsxfun(@minus,zPredPoints,zPred),sqrtW')**

**// Root innovation covariance**

**Szz=tria([zPredCenPoints,SR])**

**// The cross covariance.**

**Pxz=xPredCenPoints\*zPredCenPoints'**

**// The filter gain**

**W=(Pxz/Szz')/Szz**

**// Updated state estimate**

**updatedState=xPred+W\*innov**

**// Updated state root covariance**

**updatedsqrtCovMatrix=tria(xPredCenPoints-W\*zPredCenPoints , W\*SR)**

## Tria()

Used In: sqrtCubatureKalmanPredict, sqrtCubatureKalmanUpdate, cholSemiDef

Purpose: Transforms a matrix of any shape into a square, lower-triangular matrix. Transformation needed in the square root Kalman filter.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| A [ ][ ] | Depends | Double | A matrix that is not square. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| S [ ][ ] | Depends | Double | A square lower-triangular matrix. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode

**// QR factorization; just use the R (ignore Q)**

**CALL qr(transpose(A)) to get R**

**S=transpose(R)**

**//Make the diagonal elements all positive.**

**sel=diag(S)<0**

**S[:][sel]=-S[:][sel]**

## PromoteTrack()

Used In: Track Maintenance

Purpose: Promote a tentative track to firm track.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| observation | Scalar | Measurement | Observation used in pretrack promotion to tentative track |
| pretrack | Scalar | Measurement | Pretrack to be promoted. |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| tentativeTrackList [ ] | maxNumTentativeTracks | Track | List of tentative tracks, including new one. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| maxAccelerationMpsSqrd | Scalar | Double | Maximum expected acceleration for a target, in meters per squared seconds. |

Pseudocode

**IF number tracks in firmTrackList < maxNumFirmTracks**

**//Firm and tentative tracks have the same structure**

**Copy tentative track to firm track list**

**//Delete track from tentative track list**

**CALL deleteTracks(track)**

**ENDIF**

## CholSemiDef()

Used In: promotePretrack

Purpose: Perform the Choleskly semi-definite matrix decomposition.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| weightedCovMat [ ][ ] | 6 by 6 | Double | Weighted Covariance Matrix |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| sqrtWeightedCovMat [ ][ ] | 6 by 6 | Double | Square root Lower-triangular covariance matrix (using a Cholesky factorization) |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode

**// LDL matrix decomposition**

**CALL ldl(weightedCovMat) to get matrix L and matrix D**

**CALL tria(L\*sqrt(abs(D))) to get sqrtWeightedCovMat**

## Cart2RuvCubature()

Used In: gateObsToTrack

Purpose: Convert an ECEF Cartesian coordinate, including the covariance matrix, into RUV.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| stateCartEcef [ ] | 3 by 1 | Double | Cartesian point in ECEF Cartesian coordinates. |
| sqrtCovMatCartEcef [ ][ ] | 3 by 3 | Double | A lower-triangular square root covariance matrix |
| txCartEcef [ ] | 3 by 1 | Double | Transmitter location, in ECEF Cartesian coordinates. |
| rxCartEcef [ ] | 3 by 1 | Double | Receiver location, in ECEF Cartesian coordinates. |
| transformMatrixCartEcefToRxRf [ ][ ] | 3 by 3 | Double | Transformation matrix to go from ECEF Cartesian to radar-face at the receiver. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| stateRuv [ ] | 3 by 1 | Double | State in RUV space. |
| covMatRuv [ ][ ] | 3 by 3 | Double | Covariance Matrix in RUV coordinates. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| fifthOrderCubaturePoints3D [ ] | 14 by 1 | Double | 3 dimensional fifth-order cubature points |
| fifthOrderCubatureWeights3D [ ] | 14 by 1 | Double | 3 dimensional fifth-order cubature weights |

Pseudocode

**//Transform cubature points to match the given Gaussian**

**cubPoints= bsxfun(@plus, sqrtCovMatCartEcef \* fifthOrderCubaturePoints3D, stateCartEcef)**

**//Convert each point to RUV.**

**FOR each cubature point**

**//The target location in the receiver's coordinate system.**

**cartesianEcefPoint= transformMatrixCartEcefToRxRf\*( cubPoints[point]- rxCartEcef )**

**//The transmitter location in the receiver's local coordinate system.**

**zTxL= transformMatrixCartEcefToRxRf\*( txCartEcef - rxCartEcef )**

**r1=norm(cartesianEcefPoint)**

**r2=norm(cartesianEcefPoint-zTxL)**

**//Get the RUV coordinate for this point**

**r=(r1+r2)/2**

**u= cartesianEcefPoint[1]/r1**

**v= cartesianEcefPoint[2]/r1**

**ruvPoints [point] [1:3]= { r, u, v}**

**ENDFOR**

**//Extract the first two moments (mean and variance) of the transformed points**

**CALL calcMixtureMoments(ruvPoints, fifthOrderCubatureWeights3D) to get stateRuv and covMatRuv**

## findRfTransParam()

Used In: gateObsToTrack, sqrtCubatureKalmanUpdate

Purpose: Get the transformation matrix to convert an ECEF Cartesian coordinate to a radar-facing coordinate.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| node | Scalar | Node Position Data | Node Position Data object with position information for the node. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| transformationMatrix [ ][ ] | 3 by 3 | Double | Transformation matrix needed to rotate an ECEF Cartesian vector to Cartesian radar-face. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| WGS84SemiMajorAxisMeters | Scalar | Double | WGS84 Semi-major axis length, in meters. |
| WGS84FlatteningFactor | Scalar | Double | WGS84 flattening factor. |

Pseudocode

**a= WGS84SemiMajorAxisInMeters**

**f= WGS84FlatteningFactor**

**height=node.heightMeters**

**phi=node.latRads //latitude in radians**

**lambda= node.lonRads //longitude in radians**

**//Cartesian ECEF to ENU\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**sinP=sin(phi)**

**cosP=cos(phi)**

**sinL=sin(lambda)**

**cosL=cos(lambda)**

**u3={{cosP\*cosL},**

**{ cosP\*sinL},**

**{ sinP}}**

**c3=norm(u3)**

**u3=u3/c3**

**//The square of the first numerical eccentricity**

**e2=2\*f-f^2**

**//The normal radius of curvature.**

**Ne=a/sqrt(1-e2\*sinP^2)**

**//The derivative of the normal radius of curvature with respect to phi.**

**dNedPhi=a\*e2\*cosP\*sinP/(1-e2\*sinP^2)^(3/2)**

**//u1 is dr/dlambda, normalized (East).**

**u1={{-(Ne+h)\*cosP\*sinL},**

**{ (Ne+h)\*cosP\*cosL},**

**{ 0}}**

**c1=norm(u1)**

**//u2 is dr/dphi, normalized (North)**

**u2={{(cosP\*dNedPhi-(Ne+h)\*sinP)\*cosL},**

**{(cosP\*dNedPhi-(Ne+h)\*sinP)\*sinL},**

**{(Ne\*(1-e2)+h)\*cosP+(1-e2)\*dNedPhi\*sinP}}**

**c2=norm(u2)**

**u2=u2/c2**

**//If the point is too close to the poles, then it is possible that c1 is**

**//nearly equal to zero. However, u1 can just be found by orthogonality:**

**//it is orthogonal to u3 and u2. Cross is the cross product**

**u1=cross(u2,u3)**

**u={u1,u2,u3}**

**az=node.boresightAzRads**

**el=node.boresightElRads**

**//Cartesian ENU to Cartesian radar-face \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**A ={{ -cos(az) , sin(az), 0},**

**{ -sin(az)\*sin(el), -cos(az)\*sin(el), cos(el)},**

**{ sin(az)\*cos(el), cos(az)\*cos(el), sin(el)}}**

**//Matrix to go from Cartesian ECEF to Cartesian radar-face**

**transformationMatrix= A \* tranposed(u)**

## ellips2CartEcef()

Used In: gateObsToTrack, sqrtCubatureKalmanUpdate

Purpose: Transform an ellipsoidal coordinate to ECEF Cartesian.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| latRads | Scalar | Double | Geodetic latitude of node, in degrees. |
| lonRads | Scalar | Double | Geodetic longitude of node, in degrees. |
| heightMeters | Scalar | Double | Height of node above reference ellipsoid (WGS84), in meters. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| ecefCartPoint [ ] | 3 by 1 | Double | Converted points, in ECEF Cartesian coordinates. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| WGS84SemiMajorAxisMeters | Scalar | Double | WGS84 Semi-major axis length, in meters. |
| WGS84FlatteningFactor | Scalar | Double | WGS84 flattening factor. |

Pseudocode

**phi=latRads //longitude in radians**

**lambda=lonRads //longitude in radians**

**h=heighMeters**

**sinP=sin(phi)**

**cosP=cos(phi)**

**sinL=sin(lambda)**

**cosL=cos(lambda)**

**//The square of the first numerical eccentricity**

**e2=2\*f-f^2**

**//The normal radii of curvature.**

**Ne=a./sqrt(1-e2\*sinP.^2)**

**x=(Ne+h).\*cosP.\*cosL**

**y=(Ne+h).\*cosP.\*sinL**

**z=(Ne\*(1-e2)+h).\*sinP**

**ecefCartPoint={x},{y},{z}**

## GetRangeRate()

Used In: sqrtCubatureKalmanUpdate

Purpose: Compute the range rate.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| state [ ] | 6 by 1 | Double | State vector in ECEF Cartesian coordinates, where the first three elements are the position, and last three elements are the velocity components. |
| txCartEcef [ ] | 3 by 1 | Double | Transmitter position and velocity components, in ECEF Cartesian coordinates. |
| rxCartEcef [ ] | 3 by 1 | Double | Receiver position and velocity components, in ECEF Cartesian coordinates. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| rangeRate | Scalar | Double | Range-rate. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode

**//Get the position vectors**

**tgtPos=xTar[1:3][:]**

**rxPos=xRx[1:3][:]**

**txPos=xTx[1:3][:]**

**//Get the velocity vectors**

**tgtVel=xTar[4:6][:]**

**rxVel=xRx[4:6][:]**

**txVel=xTx[4:6][:]**

**dTgtRx=tgtPos-rxPos //distance to receiver**

**dTgtTx=tgtPos-txPos //distance to transmitter**

**dtrRat=bsxfun(@rdivide, dTgtRx,sqrt(sum(dTgtRx.\* dTgtRx,1)))**

**dtlRat=bsxfun(@rdivide, dTgtTx,sqrt(sum(dTgtTx.\* dTgtTx,1)))**

**rr=sum((dtrRat+dtlRat).\* tgtVel,1)-sum(dtrRat.\* rxVel,1)-sum(dtlRat.\* txVel,1)**

**rangeRate=rr/2**

## CalcMixtureMoments()

Used In: sqrtCubatureKalmanUpdate, cart2RuvCubature

Purpose: Compute the first and second moment of a Gaussian mixture.

Data Input(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| points [ ] | 76 by 1 | Double | Vector with converted cubature points. |
| weights [ ] | 76 by 1 | Double | Weights associated with vector points. |

Data Output(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| mu | Scalar | Double | The mean of the weighted points. |

Meta Data Needed

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Dimensions | Data Type | Description |
| No metadata needed |  |  |  |

Pseudocode

**//The mean**

**mu=sum(bsxfun(@times,points,weights'))**

# Supporting Information (internal – not for release)

## Variable Pd

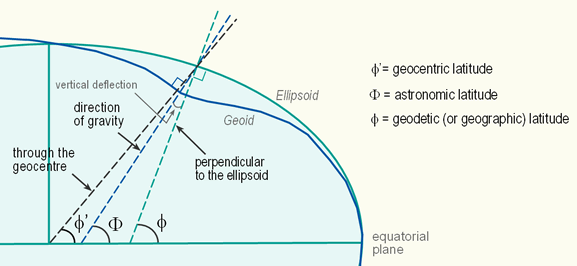
If one changes the track Pd depending on its location relative to beam center, the missed detection cost, which is a function of the Pd, is nearly the same for ranges of Pd=[0, 0.7]. Note that a track score is penalized by this value (ln(1-Pd)) when nothing gates with it. If the threshold to terminate firm tracks is

-7, the track current score is 0, and the missed detection cost is -1, then it would take 7 scans (7\*-1) for the track to be dropped assuming no observation is assigned to it in the course of those 7 scans.



## Coordinate System (for LAT/LON)

Picture below shows differences between different latitudes:

[](http://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwin1Nvg86bJAhVIRCYKHXibDokQjRwIBw&url=http://kartoweb.itc.nl/geometrics/coordinate%20systems/coordsys.html&psig=AFQjCNFCVFNL45pjMWu-Ur0Eo4ti9mZgTQ&ust=1448380444405676)

## Linear Algebra C++ Libraries

Reason: Since Filtering involves a high number of matrix operations and manipulations, a library to perform these operations efficiently may be required. One option is to write the operations ourselves, but it would be a poor decision given the availability of free, robust C++ libraries. A trade-off study has been written for this purposes.