



Tracking radar targets with multiple reflection points



Texas Instruments, Incorporated
20450 Century Boulevard
Germantown, MD 20874 USA

Version History

Date	Owner	Comment
April 13, 2017	Michael Livshitz	Initial version
July 20, 2017		Version 1.0
Nov 17, 2017		Version 1.1 - Updated group and centroid covariance matrices calculation Updated configuration and added advanced configuration parameters
Feb 6, 2018		Version 1.2 Updated configuration parameters description
March 9, 2018		Version 1.3 Added derivations for tracking in 3D space
September 6, 2018		Version 1.4 Added description of 3D or 2D library options Changes in configuration parameters to support 3D and 2D options
October 18, 2018		Version 1.5 Changed Gating function from constant volume to constant gain with limiters.
September 1, 2020		Version 1.6 Updates based on tracker version 0.110
January 1, 2021		Version 1.7 Updates based on tracker version 0.112

Table of Contents

1.	Introduction	5
1.1.	Tracking Module	5
1.2.	Radar Geometry.....	6
1.3.	Choice of Tracking Coordinate System	7
1.4.	2D Space Geometry	7
1.5.	2D Space, Constant Velocity Model.....	7
1.6.	2D Space, Constant Acceleration Model.....	10
1.7.	3D Space, Geometry	11
1.8.	3D Space, Constant Velocity Model (3DV) and Constant Acceleration (3DA) models	12
2.	Kalman Filter Operations	15
2.1.	Prediction Step.....	15
2.2.	Update Step	15
2.3.	Design of Process Noise Matrix.....	16
2.3.1.	Continuous White Noise Model.....	16
2.3.2.	Piecewise White Noise Model	17
3.	Group Tracking.....	19
3.1.	High Level Algorithm Design	19
3.2.	Group Tracking Block Diagram.....	20
3.3.	Prediction Step.....	20
3.4.	Association Step.....	20
3.4.1.	Gating Function	21
3.4.2.	Scoring Function.....	22
3.5.	Allocation Step	23
3.6.	Updating Step	23
3.7.	Maintenance Step	24
4.	Implementation Details	25
4.1.	GTRACK Library	25
4.2.	Building and using the library	26
4.3.	Configuration Parameters.....	27
4.3.1.	Advanced parameters.....	28

4.4.	Memory Requirements	34
4.4.1.	Data Memory	34
4.4.1.	Program Memory	34
4.5.	Benchmarks.....	35
5.	Performance	35
5.1.	Tracking Reliability	35
5.1.1.	Test Case description	36
5.1.2.	Results.....	37
5.2.	Tracking Precision	39
5.2.1.	Test Case description	40
5.2.2.	Results.....	40
5.3.	Tracking Resolution.....	40
5.3.1.	Initial Separation Tests.....	40
5.3.2.	Dynamic Separation Tests.....	41
6.	References	41
7.	Appendix	42
7.1.	Evaluating Partial Derivatives for 2D space tracking	42
7.1.1.	Evaluating range partial derivatives.....	42
7.1.2.	Evaluating azimuth partial derivatives.....	42
7.1.1.	Evaluating doppler partial derivatives	42
7.2.	Evaluating Partial Derivatives for 3D space tracking	44
7.2.1.	Evaluating range partial derivatives.....	44
7.2.2.	Evaluating azimuth partial derivatives.....	44
7.2.1.	Evaluating elevation partial derivatives.....	45
7.2.2.	Evaluating Doppler partial derivatives.....	45

1. Introduction

1.1. Tracking Module

In Radar Processing stack, the tracking algorithms are the implementations of the localization processing Layer. Tracker is expected to work on the Detection layer inputs, and provide localization information to the classification layers.

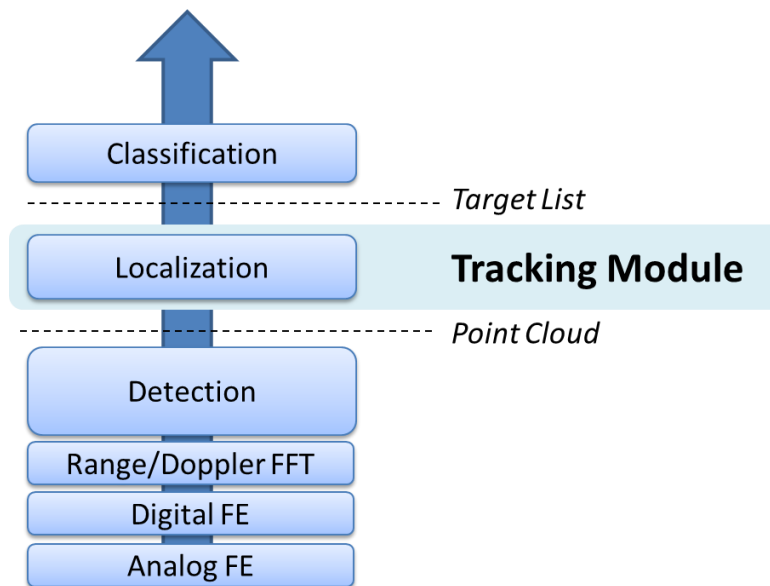


Figure 1. Radar Processing Layers

With high resolution Radar sensors, the detection layer is capable of sensing multiple reflections from real life targets, delivering a rich set of measurement vectors (in some cases, thousands of vectors per frame), known as a *Point Cloud*. Each measurement vector represents a reflection point, with Range, Azimuth, and Radial velocity. Each measurement vector may also contain reliability information.

Tracking Layer is expected to input the point cloud data, perform target localization, and report the results (a *Target List*) to a classification layer. Therefore, the output of the tracker is a set of trackable objects with certain properties (like position, velocity, physical dimensions, point density, and other features) that can be used by a classifier to make an identification decision.

1.2. Radar Geometry

The picture below illustrates single reflection point at time n . Real life radar targets are represented by multiple reflection points. Each point is represented by range, angle, and radial velocity (range rate):

- Range r , $R_{min} < r < R_{max}$
- Azimuth angle φ , $-\varphi_{max} < \varphi < +\varphi_{max}$
- Radial velocity \dot{r}

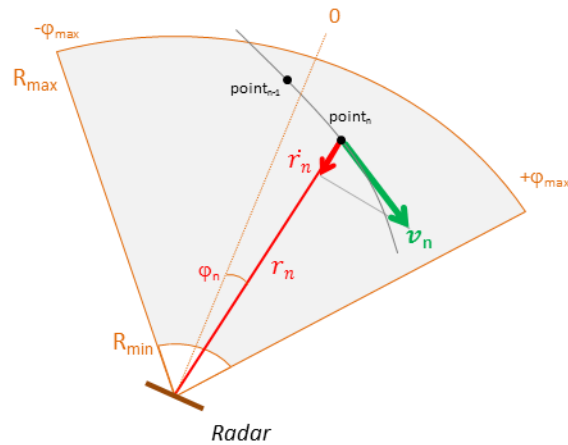


Figure 2. Radar Geometry in 2D

1.3. Choice of Tracking Coordinate System

For convenience of target motion extrapolation, we chose tracking in Cartesian coordinates. This allows for simple Newtonian linear prediction models. We chose to keep measurement inputs in polar coordinates to avoid error coupling. We will use extended Kalman filter to linearize the dependencies between tracking states and measurement vector.

Tracking can operate in either 2D or 3D Cartesian spaces. For each space, we use either constant velocity model or constant acceleration model.

1.4. 2D Space Geometry

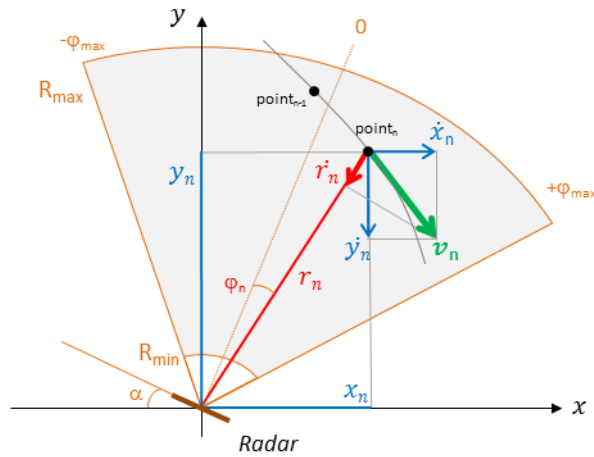


Figure 3. Tracking in 2D

The angular location coordinates are converted to Cartesian coordinates using

$$x = r \cos\left(\frac{\pi}{2} - (\alpha + \varphi)\right) = r \sin(\alpha + \varphi)$$
$$y = r \sin\left(\frac{\pi}{2} - (\alpha + \varphi)\right) = r \cos(\alpha + \varphi)$$

The objective is to track the location of the objects using the noisy measurements of range, angle, and Doppler (radial velocity)

1.5. 2D Space, Constant Velocity Model

We use Kalman filter to “refine” the location estimates. The state of the Kalman filter at time instant n is defined as

$$\mathbf{s}(n) = \mathbf{F}\mathbf{s}(n-1) + \mathbf{w}(n) \quad (1-1)$$

where the state vector $\mathbf{s}(n)$ is defined in Cartesian coordinates,

$$\mathbf{s}(n) \triangleq [x(n) \quad y(n) \quad \dot{x}(n) \quad \dot{y}(n)]^T, \quad (1-2)$$

\mathbf{F} is a transition matrix

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1-3)$$

and $\mathbf{w}(n)$ is the vector of process noise with covariance matrix $\mathbf{Q}(n)$ of size 4×4 .

The input measurement vector $\mathbf{u}(n)$ includes range, angle and radial velocity

$$\mathbf{u}(n) = [r(n) \quad \varphi(n) \quad \dot{r}(n)]^T \quad (1-4)$$

The relationship between the state of the Kalman filter and measurement vector is expressed as:

$$\mathbf{u}(n) = \mathbf{H}(\mathbf{s}(n)) + \mathbf{v}(n), \quad (1-5)$$

Where, \mathbf{H} is a measurement matrix,

$$\mathbf{H}(\mathbf{s}(n)) = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \tan^{-1}(x, y) - \alpha \\ \frac{x\dot{x} + y\dot{y}}{\sqrt{x^2 + y^2}} \end{bmatrix}, \quad (1-6)$$

the function $\tan^{-1}(x, y)$ is defined as

$$\tan^{-1}(x, y) \triangleq \begin{cases} \tan^{-1}\left(\frac{x}{y}\right), & y > 0, \\ \frac{\pi}{2}, & y = 0, \\ \tan^{-1}\left(\frac{x}{y}\right) + \pi, & y < 0. \end{cases} \quad (1-7)$$

and $\mathbf{v}(n)$ is vector of measurement noise with covariance matrix $\mathbf{R}(n)$ of size 3×3 .

In above formulation, the measurement vector $\mathbf{u}(n)$ is related to the state vector $\mathbf{s}(n)$ via a non-linear relation. Because of this, we use Extended Kalman filter (EKF), which simplifies the relation between $\mathbf{u}(n)$ and $\mathbf{s}(n)$ by retaining only the first term in the Taylor series expansion of $\mathbf{H}(\cdot)$.

$$\mathbf{u}(n) = \mathbf{H}(\mathbf{s}_{apr}(n)) + \mathbf{J}_H(\mathbf{s}_{apr}(n))[\mathbf{s}(n) - \mathbf{s}_{apr}(n)] + \mathbf{v}(n), \quad (1-8)$$

where, $\mathbf{s}_{apr}(n)$ is a-priori estimation of state vector at time n based on $n-1$ measurements,

$$\mathbf{J}_H(\mathbf{s}) = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial \dot{y}} \\ \frac{\partial \varphi}{\partial x} & \frac{\partial \varphi}{\partial y} & \frac{\partial \varphi}{\partial \dot{x}} & \frac{\partial \varphi}{\partial \dot{y}} \\ \frac{\partial \dot{r}}{\partial x} & \frac{\partial \dot{r}}{\partial y} & \frac{\partial \dot{r}}{\partial \dot{x}} & \frac{\partial \dot{r}}{\partial \dot{y}} \end{bmatrix}. \quad (1-9)$$

Calculating partial derivatives (see the Appendix below):

$$\mathbf{J}_H(\mathbf{s}) = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 \\ \frac{y}{x^2+y^2} & -\frac{x}{x^2+y^2} & 0 & 0 \\ \frac{y(\dot{x}y-\dot{y}x)}{(x^2+y^2)^{3/2}} & \frac{x(\dot{y}x-\dot{x}y)}{(x^2+y^2)^{3/2}} & \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} \end{bmatrix} \quad (1-10)$$

1.6. 2D Space, Constant Acceleration Model

For constant acceleration model, the state of the Kalman filter at time instant n is defined as

$$\mathbf{s}(n) = \mathbf{F}\mathbf{s}(n-1) + \mathbf{w}(n), \quad (1-11)$$

where the state vector $\mathbf{s}(n)$ is defined in Cartesian coordinates,

$$\mathbf{s}(n) = [x(n) \quad y(n) \quad \dot{x}(n) \quad \dot{y}(n) \quad \ddot{x}(n) \quad \ddot{y}(n)], \quad (1-12)$$

\mathbf{F} is a transition matrix

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & T & 0 & 0.5T^2 & 0 \\ 0 & 1 & 0 & T & 0 & 0.5T^2 \\ 0 & 0 & 1 & 0 & T & 0 \\ 0 & 0 & 0 & 1 & 0 & T \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1-13)$$

and $\mathbf{w}(n)$ is the vector of process noise with covariance matrix $\mathbf{Q}(n)$ of size 6×6 .

The input measurement vector $\mathbf{u}(n)$ is the same as in constant velocity model, includes range, angle and radial velocity

$$\mathbf{u}(n) = [r(n) \quad \varphi(n) \quad \dot{r}(n)]^T. \quad (1-14)$$

The relationship between the state of the Kalman filter and measurement vector is expressed as:

$$\mathbf{u}(n) = \mathbf{H}(\mathbf{s}(n)) + \mathbf{v}(n), \quad (1-15)$$

where

$$\mathbf{H}(\mathbf{s}(n)) = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \tan^{-1}(x, y) \\ \frac{x\dot{x} + y\dot{y}}{\sqrt{x^2 + y^2}} \end{bmatrix} \quad (1-16)$$

To linearize state to measurement relations, we perform partial derivatives

$$\mathbf{J}_H(\mathbf{s}) = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial \dot{y}} & \frac{\partial r}{\partial \ddot{x}} & \frac{\partial r}{\partial \ddot{y}} \\ \frac{\partial \varphi}{\partial x} & \frac{\partial \varphi}{\partial y} & \frac{\partial \varphi}{\partial \dot{x}} & \frac{\partial \varphi}{\partial \dot{y}} & \frac{\partial \varphi}{\partial \ddot{x}} & \frac{\partial \varphi}{\partial \ddot{y}} \\ \frac{\partial \dot{r}}{\partial x} & \frac{\partial \dot{r}}{\partial y} & \frac{\partial \dot{r}}{\partial \dot{x}} & \frac{\partial \dot{r}}{\partial \dot{y}} & \frac{\partial \dot{r}}{\partial \ddot{x}} & \frac{\partial \dot{r}}{\partial \ddot{y}} \end{bmatrix} \quad (1-17)$$

$$J_H(s) = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 & 0 & 0 \\ \frac{y}{x^2+y^2} & -\frac{x}{x^2+y^2} & 0 & 0 & 0 & 0 \\ \frac{y(\dot{x}y-\dot{y}x)}{(x^2+y^2)^{3/2}} & \frac{x(\dot{y}x-\dot{x}y)}{(x^2+y^2)^{3/2}} & \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 \end{bmatrix} \quad (1-18)$$

1.7. 3D Space, Geometry

In the picture below, sensor is positioned at the origin. Target trajectory is shown at times $n-2$, $n-1$, and n . Target is moving with velocity vector \mathbf{v} . Measurement vector \mathbf{u} at time n includes range, azimuth, elevation and radial velocity:

$$\mathbf{u}(n) = [r(n) \quad \varphi(n) \quad \theta(n) \quad \dot{r}(n)]^T \quad (1-19)$$

State vector in Cartesian coordinates will be \mathbf{s}_{3DV} for constant velocity model, and \mathbf{s}_{3DA} for constant acceleration model:

$$\mathbf{s}_{3DV}(n) \triangleq [x(n) \quad y(n) \quad z(n) \quad \dot{x}(n) \quad \dot{y}(n) \quad \dot{z}(n)]^T \quad (1-20)$$

$$\mathbf{s}_{3DA}(n) \triangleq [x(n) \quad y(n) \quad z(n) \quad \dot{x}(n) \quad \dot{y}(n) \quad \dot{z}(n) \quad \ddot{x}(n) \quad \ddot{y}(n) \quad \ddot{z}(n)]^T \quad (1-21)$$

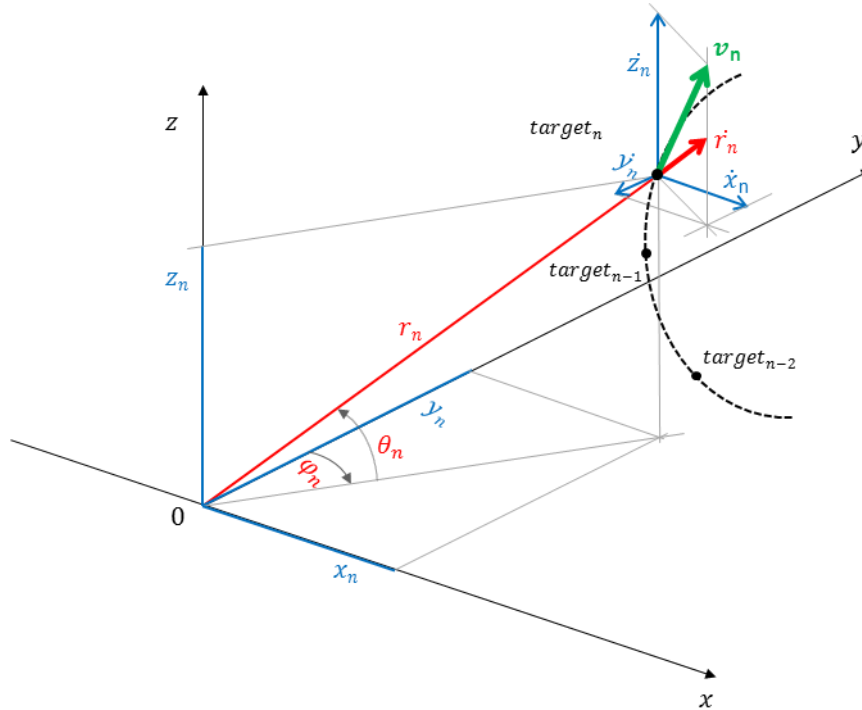


Figure 4. Tracking in 3D

1.8. 3D Space, Constant Velocity Model (3DV) and Constant Acceleration (3DA) models

We use Kalman filter to “refine” the location estimates. The state of the Kalman filter at time instant n is defined as

$$\mathbf{s}(n) = \mathbf{F}\mathbf{s}(n-1) + \mathbf{w}(n) \quad (1-22)$$

where the state vector $\mathbf{s}(n)$ is defined in Cartesian coordinates. For 3D space, we use \mathbf{s}_{3DV} or \mathbf{s}_{3DA} as defined in previous paragraph.

Transition matrix \mathbf{F} is

$$\mathbf{F}_{3DV} = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (1-23)$$

or

$$\mathbf{F}_{3DA} = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 & 0.5T^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 & 0 & 0.5T^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & T & 0 & 0 & 0.5T^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (1-24)$$

and $\mathbf{w}(n)$ is the vector of process noise with covariance matrix $\mathbf{Q}(n)$ of size 6×6 or 9×9.

The input measurement vector $\mathbf{u}(n)$ includes range, azimuth, elevation, and radial velocity

$$\mathbf{u}(n) = [r(n) \quad \varphi(n) \quad \theta(n) \quad \dot{r}(n)]^T \quad (1-25)$$

The relationship between the state of the Kalman filter and measurement vector is expressed as:

$$\mathbf{u}(n) = \mathbf{H}(\mathbf{s}(n)) + \mathbf{v}(n), \quad (1-26)$$

Where, \mathbf{H} is a measurement matrix,

$$\mathbf{H}(\mathbf{s}(n)) = \begin{bmatrix} \sqrt{x^2 + y^2 + z^2} \\ \tan^{-1}(x, y) \\ \tan^{-1}\left(z, \sqrt{x^2 + y^2}\right) \\ \frac{x\dot{x} + y\dot{y} + z\dot{z}}{\sqrt{x^2 + y^2 + z^2}} \end{bmatrix}, \quad (1-27)$$

the function $\tan^{-1}(a, b)$ is defined as

$$\tan^{-1}(a, b) \triangleq \begin{cases} \tan^{-1}\left(\frac{a}{b}\right), & b > 0, \\ \frac{\pi}{2}, & b = 0, \\ \tan^{-1}\left(\frac{a}{b}\right) + \pi, & b < 0. \end{cases} \quad (1-28)$$

and $\mathbf{v}(n)$ is vector of measurement noise with covariance matrix $\mathbf{R}(n)$ of size 4×4 .

In above formulation, the measurement vector $\mathbf{u}(n)$ is related to the state vector $\mathbf{s}(n)$ via a non-linear relation. Because of this, we use Extended Kalman filter (EKF), which simplifies the relation between $\mathbf{u}(n)$ and $\mathbf{s}(n)$ by retaining only the first term in the Taylor series expansion of $\mathbf{H}(\cdot)$.

$$\mathbf{u}(n) = \mathbf{H}(\mathbf{s}_{apr}(n)) + \mathbf{J}_H(\mathbf{s}_{apr}(n))[\mathbf{s}(n) - \mathbf{s}_{apr}(n)] + \mathbf{v}(n), \quad (1-29)$$

where, $\mathbf{s}_{apr}(n)$ is a-priori estimation of state vector at time n based on $n-1$ measurements,

$$\mathbf{J}_H(\mathbf{s}) = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial z} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial \dot{y}} & \frac{\partial r}{\partial \dot{z}} \\ \frac{\partial \varphi}{\partial x} & \frac{\partial \varphi}{\partial y} & \frac{\partial \varphi}{\partial z} & \frac{\partial \varphi}{\partial \dot{x}} & \frac{\partial \varphi}{\partial \dot{y}} & \frac{\partial \varphi}{\partial \dot{z}} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial y} & \frac{\partial \theta}{\partial z} & \frac{\partial \theta}{\partial \dot{x}} & \frac{\partial \theta}{\partial \dot{y}} & \frac{\partial \theta}{\partial \dot{z}} \\ \frac{\partial \dot{r}}{\partial x} & \frac{\partial \dot{r}}{\partial y} & \frac{\partial \dot{r}}{\partial z} & \frac{\partial \dot{r}}{\partial \dot{x}} & \frac{\partial \dot{r}}{\partial \dot{y}} & \frac{\partial \dot{r}}{\partial \dot{z}} \end{bmatrix}. \quad (1-30)$$

Calculating partial derivatives (see the Appendix below):

$$\mathbf{J}_H(\mathbf{s}_{3DV}) = \begin{bmatrix} \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 \\ \frac{y}{x^2+y^2} & -\frac{x}{x^2+y^2} & 0 & 0 & 0 & 0 \\ -\frac{x}{r^2} \frac{z}{\sqrt{x^2+y^2}} & -\frac{y}{r^2} \frac{z}{\sqrt{x^2+y^2}} & \frac{\sqrt{x^2+y^2}}{r^2} & 0 & 0 & 0 \\ \frac{y(\dot{x}y - \dot{y}x) + z(\dot{x}z - \dot{z}x)}{r^3} & \frac{x(\dot{y}x - \dot{x}y) + z(\dot{y}z - \dot{z}y)}{r^3} & \frac{x(\dot{z}x - \dot{x}z) + y(\dot{z}y - \dot{y}z)}{r^3} & \frac{x}{r} & \frac{y}{r} & \frac{z}{r} \end{bmatrix} \quad (1-31)$$

$$J_H(S_{3DA}) = \begin{bmatrix} \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{y}{x^2+y^2} & -\frac{x}{x^2+y^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{x}{r^2} \frac{z}{\sqrt{x^2+y^2}} & -\frac{y}{r^2} \frac{z}{\sqrt{x^2+y^2}} & \frac{\sqrt{x^2+y^2}}{r^2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{y(\dot{x}y-\dot{y}x)+z(\dot{x}z-\dot{z}x)}{r^3} & \frac{x(\dot{y}x-\dot{x}y)+z(\dot{y}z-\dot{z}y)}{r^3} & \frac{x(\dot{z}x-\dot{x}z)+y(\dot{z}y-\dot{y}z)}{r^3} & \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 \end{bmatrix} \quad (1-32)$$

2. Kalman Filter Operations

The notation we use:

- $s_i(n)$ – State vector of tracking object i at time n . Each tracking object has its own state vector, which is predicted and updated independently. For simplicity reasons we omit the tracking index.
- $s_{apr}(n)$ – A-priori (predicted) estimates of tracking state at time n .
- $P(n)$ – State vector estimation error covariance matrix at time n , defined as $P(n) = \text{Cov}[s(n) - s_{apr}(n)]$.
- $P_{apr}(n)$ – A-priori (predicted) estimates of state vector covariance matrix at time n .

2.1. Prediction Step

With available measurement until time instant $n-1$, the *a priori* state and error covariance estimates are obtained using:

$$s_{apr}(n) = \mathbf{F}s(n-1) \quad (2-1)$$

$$\mathbf{P}_{apr}(n) = \mathbf{F}\mathbf{P}(n-1)\mathbf{F}^T + \mathbf{Q}(n-1). \quad (2-2)$$

The above equations constitute the *prediction* step of the Kalman filter. $\mathbf{Q}(n)$ is the process noise covariance matrix. See section 2.3 for the details.

In addition, we also compute $\mathbf{H}(s_{apr}(n))$ using formula above.

2.2. Update Step

As measurements at time instant n become available, state and error covariance estimates are updated in the following measurement update procedure:

- a) Compute innovation (or measurement residual)

$$\mathbf{y}(n) = \mathbf{u}(n) - \mathbf{H}(s_{apr}(n)) \quad (2-3)$$

- b) Compute innovation covariance

$$\mathbf{C}(n) = \mathbf{J}_H(s_{apr}(n))\mathbf{P}_{apr}(n)\mathbf{J}_H^T(s_{apr}(n)) + \mathbf{R}(n) \quad (2-4)$$

- c) Compute Kalman gain

$$\mathbf{K}(n) = \mathbf{P}_{apr}(n)\mathbf{J}_H^T(s_{apr}(n)) \text{inv}[\mathbf{C}(n)] \quad (2-5)$$

d) Compute a-posteriori state vector

$$\mathbf{s}(n) = \mathbf{s}_{apr}(n) + \mathbf{K}(n)\mathbf{y}(n) \quad (2-6)$$

e) Compute a-posteriori error covariance

$$\mathbf{P}(n) = \mathbf{P}_{apr}(n) - \mathbf{K}(n)\mathbf{J}_H \left(\mathbf{s}_{apr}(n) \right) \mathbf{P}_{apr}(n) \quad (2-7)$$

2.3. Design of Process Noise Matrix

This section is adopted from the works in [1].

The choice of $\mathbf{Q}(n)$ is important for the behavior of the Kalman filter. If \mathbf{Q} is too small then the filter will be overconfident in its prediction model and will diverge from the actual solution. If \mathbf{Q} is too large than the filter will be too much influenced by the noise in the measurements and perform sub-optimally.

The kinematic system (a system that can be modeled using Newton's equations of motion) is continuous, i.e. their inputs and outputs can vary at any arbitrary point in time. However, Kalman filters used here are discrete. We sample the system at regular intervals. Therefore we must find the discrete representation for the noise term in the equation above. This depends on what assumptions we make about the behavior of the noise. We will consider two different models for the noise.

2.3.1. Continuous White Noise Model

Let's say that we need to model the position, velocity, and acceleration. We can then assume that acceleration is constant for each discrete time step. Of course, there is process noise in the system and so the acceleration is not actually constant. The tracked object will alter the acceleration over time due to external, un-modeled forces. In this section we will assume that the acceleration changes by a continuous time zero-mean white noise.

Since the noise is changing continuously we will need to integrate to get the discrete noise for the discretization interval that we have chosen. We will not prove it here, but the equation for the discretization of the noise is

$$\mathbf{Q} = \int_0^{\Delta t} \mathbf{F}(t)\mathbf{Q}_c\mathbf{F}^T(t)dt, \quad (2-8)$$

where \mathbf{Q}_c is the continuous noise. The general reasoning should be clear. $\mathbf{F}(t)\mathbf{Q}_c\mathbf{F}^T(t)$ is a projection of the continuous noise based on our process model $\mathbf{F}(t)$ at the instant t . We want to know how much noise is added to the system over a discrete interval Δt , so we integrate this expression over the interval $[0, \Delta t]$.

For the second order Newtonian system, the fundamental matrix is

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta t & \Delta t^2/2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix}. \quad (2-9)$$

We now define the continuous noise as

$$\mathbf{Q}_c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Phi_s \quad (2-10)$$

where Φ_s is the spectral density of the white noise. This can be derived, but is beyond the scope for now. In practice we often do not know the spectral density of the noise, and so this turns into an "engineering" factor - a number we experimentally tune until our filter performs as we expect. We can see that the matrix that Φ_s is multiplied by effectively assigns the power spectral density to the acceleration term. This makes sense; we assume that the system has constant acceleration except for the variations caused by noise. The noise alters the acceleration.

Computing the integral, we obtain

$$\mathbf{Q} = \begin{bmatrix} \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} \\ \frac{\Delta t^4}{8} & \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t \end{bmatrix} \Phi_s \quad (2-11)$$

Extrapolating back to 6 states,

$$\mathbf{Q} = \begin{bmatrix} \frac{\Delta t^5}{20} & 0 & \frac{\Delta t^4}{8} & 0 & \frac{\Delta t^3}{6} & 0 \\ 0 & \frac{\Delta t^5}{20} & 0 & \frac{\Delta t^4}{8} & 0 & \frac{\Delta t^3}{6} \\ \frac{\Delta t^4}{8} & 0 & \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^4}{8} & 0 & \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{6} & 0 & \frac{\Delta t^2}{2} & 0 & \Delta t & 0 \\ 0 & \frac{\Delta t^3}{6} & 0 & \frac{\Delta t^2}{2} & 0 & \Delta t \end{bmatrix} \Phi_s \quad (2-12)$$

2.3.2. Piecewise White Noise Model

Another model for the noise assumes that the that highest order term (say, acceleration) is constant for the duration of each time period, but differs for each time period, and each of these is uncorrelated between time periods. In other words there is a discontinuous jump in acceleration at each time step. This is subtly different than the model above, where we assumed that the last term had a continuously varying noisy signal applied to it.

We will model this as

$$f(x) = Fx + \Gamma w \quad (2-13)$$

where Γ is the noise gain of the system, and w is the constant piecewise acceleration (or velocity, or jerk, etc).

For the second order system

$$F = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \quad (2-14)$$

In one time period, the change in acceleration will be $w(t)$, change in velocity will be $w(t)\Delta t$, and change in position will be $w(t)\Delta t^2/2$. This gives us

$$\Gamma = \begin{bmatrix} \Delta t^2/2 \\ \Delta t \\ 1 \end{bmatrix} \quad (2-15)$$

The covariance of the process noise is then

$$Q = E[\Gamma w(t)w(t)\Gamma^T] = \Gamma \sigma_w^2 \Gamma^T \quad (2-16)$$

$$Q = \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t \\ \frac{\Delta t^2}{2} & \Delta t & 1 \end{bmatrix} \sigma_w^2 \quad (2-17)$$

It is not clear whether this model is more or less correct than the continuous model - both are approximations to what is happening to the actual object. Only experience and experiments can guide to the appropriate model. It is expected that either model provides reasonable results, but typically one will perform better than the other.

The advantage of the second model is that we can model the noise in terms of σ^2 which we can describe in terms of the motion and the amount of error we expect. The first model requires us to specify the spectral density, which is not very intuitive, but it handles varying time samples much more easily since the noise is integrated across the time period. However, these are not fixed rules - use whichever model (or a model of your own devising) based on testing how the filter performs and/or your knowledge of the behavior of the physical model.

A good rule of thumb is to set σ somewhere from $\frac{1}{2}\Delta a$ to Δa , where Δa is the maximum amount that the acceleration will change between sample periods. In practice we pick a number, run simulations on data, and choose a value that works well.

3. Group Tracking

3.1. High Level Algorithm Design

With advances in detection accuracy, the real world radar targets (cars, pedestrians, walls, landing ground, etc.) are presented to a tracking processing layer as a set of multiple reflection points. Those detection points form a group of correlated measurements with range, angle, and angular velocity. Of course, at any time there could be multiple real world targets. Therefore, we seek a tracker capable of working with multiple target groups.

The group tracking approach is illustrated in figure below.

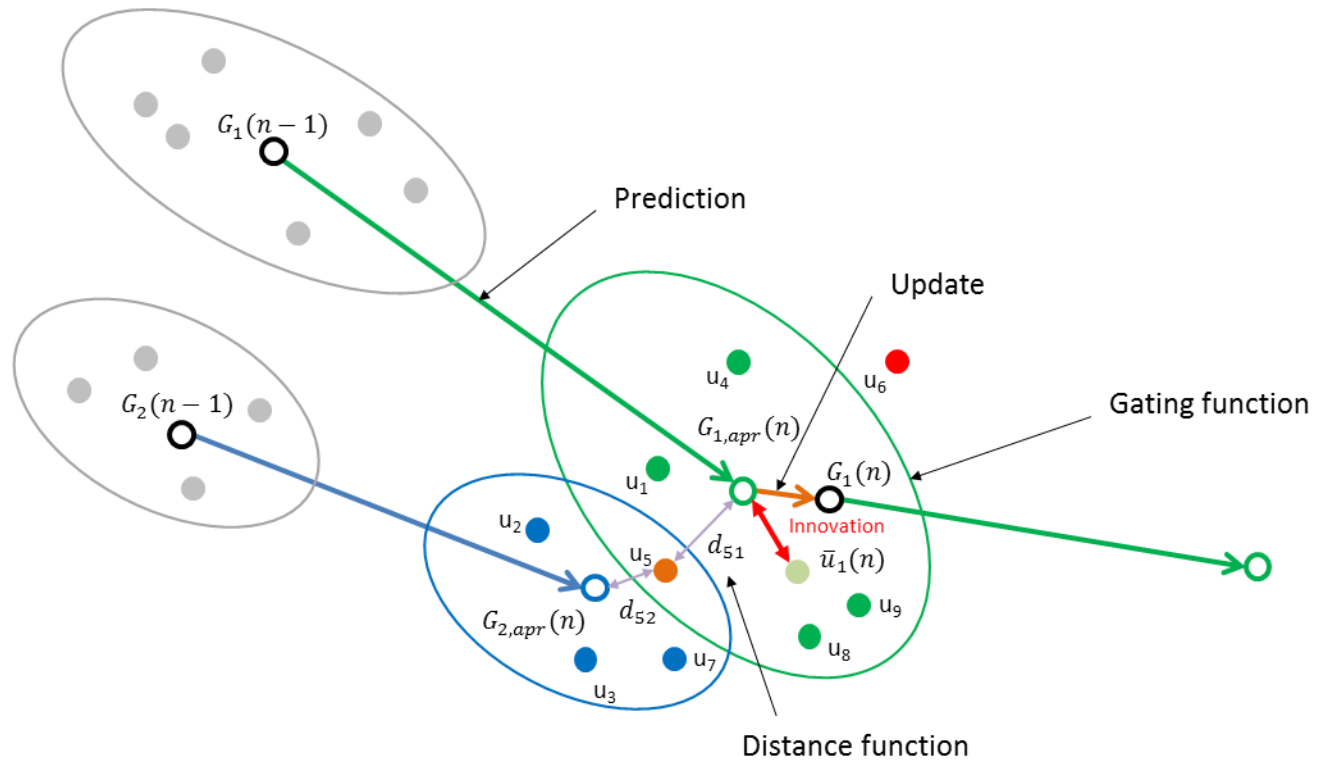


Figure 5. Group Tracking

3.2. Group Tracking Block Diagram

The algorithm flows through the major steps shown in a block diagram and described below:

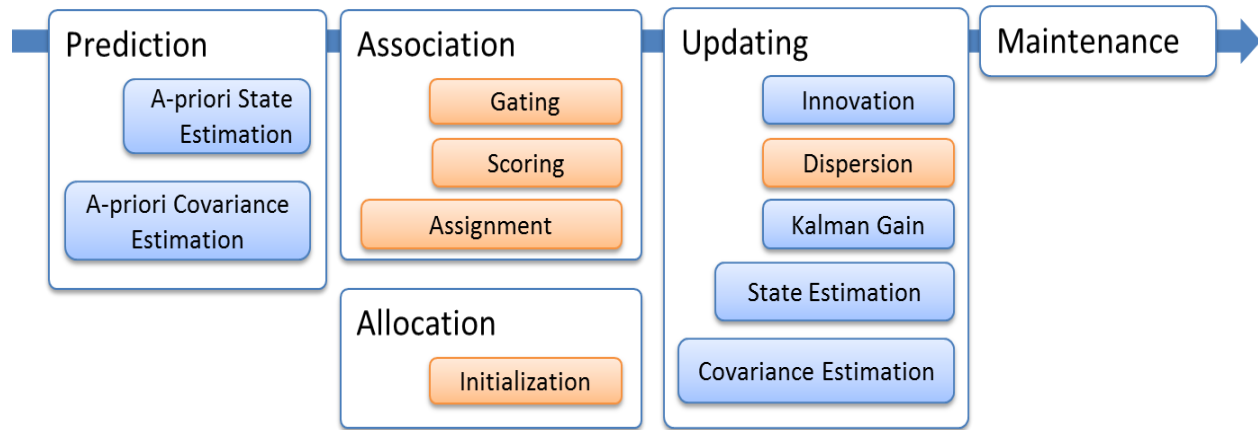


Figure 6. Tracking Block Diagram

The functions shown in blue are classical extended Kalman filter operations. The functions are shown in brown are additions to support multipoint grouping.

3.3. Prediction Step

We use the Kalman filter prediction process to estimate tracking group centroid for time n based on state and process covariance matrices estimated at time $n-1$. We compute a-priori state and covariance estimations for each trackable object. At this step we also compute measurement vector estimations. See section 2.1 for details.

3.4. Association Step

Assume existence of one or more tracks and the associated predicted state vector. For each given track we form a gate about the predicted centroid. The gate should account for a) target maneuver, for the dispersion of the group and for the measurement noise.

We use group residual covariance matrix to build an ellipsoid in 3D measurement space about the tracking group centroid. The ellipsoid will represent a *gating function* to qualify individual measurements we observe at time n . The gating function design is explained in the section below.

For the measurements within the gate, we compute *normalized distance function* as a cost function to associate a measurement to each track

The assignment process minimizes the cost function, assigning one measurement at a time to the closest track. This creates a set of measurements associated with each track

3.4.1. Gating Function

The gating function represents the amount of innovations we are willing to accept given current state of uncertainty that exists in a current track.

To measure the amount of uncertainty, we define the group residual covariance matrix as

$$\mathbf{C}_G = \mathbf{J}_H \left(\mathbf{s}_{apr}(n) \right) \mathbf{P}_{apr}(n) \mathbf{J}_H^T \left(\mathbf{s}_{apr}(n) \right) + \mathbf{R}_m + \hat{\mathbf{D}} \quad (3-1)$$

Note that this group covariance matrix \mathbf{C}_G is between a member of the measurement group and the group centroid. In practice, we compute the \mathbf{C}_G during the update step. We discuss it here for clarity reasons and to contrast this group covariance with centroid covariance \mathbf{C}_C used for Kalman process.

The term $\mathbf{J}_H \left(\mathbf{s}_{apr}(n) \right) \mathbf{P}_{apr}(n) \mathbf{J}_H^T \left(\mathbf{s}_{apr}(n) \right)$ represents the uncertainty in the centroid due to target maneuvering, and is similar to the term used for individual target tracking.

The term \mathbf{R}_m is the diagonal matrix of the measurement variances. See 3.6 for more details.

Finally, the term $\hat{\mathbf{D}}$ is the estimation of group track dispersion matrix. See 3.6 for more details.

For each existing track i , for all measurement vectors j we obtained at time n , we define a distance function d_{ij}^2 , which represents the amount of innovation the new measurement adds to an existing track.

$$d_{ij}^2 \triangleq \mathbf{y}_{ij}^T \text{inv}(\mathbf{C}_i) \mathbf{y}_{ij} \quad (3-2)$$

$$d_{ij}^2 = [\mathbf{u}_j(n) - \mathbf{H}_i(\hat{\mathbf{s}}^{-1}(n))]^T \text{inv}(\mathbf{C}_{Gi}(n)) [\mathbf{u}_j(n) - \mathbf{H}_i(\hat{\mathbf{s}}^{-1}(n))] \quad (3-3)$$

We define the chi-squared test (because the sum of squares of M Gaussian random variables with zero mean is chi-square distribution with degree of freedom M) limits the amount of innovation we are willing to accept as

$$d_{ij}^2 < G \quad (3-4)$$

The boundary condition represents arbitrarily oriented ellipsoid centered at a-priory expectation of the measurement vector, while G represents the largest normalized distance from to the measurement that we are going to accept into the group. To avoid stability issues with constant gain approach (the bubble captures new points, grows, which increases the chances to capture more points), we add a limiting factor. Under no circumstances the targets can capture points further than pre-configured limits.

3.4.2. Scoring Function

The picture below illustrates the post gating situation, where measurement vectors $\{u_1, u_2, u_3, u_7\}$ pass the gating test for the green track, while vectors $\{u_3, u_4, u_5, u_8, u_9\}$ pass the gating test for the blue track.

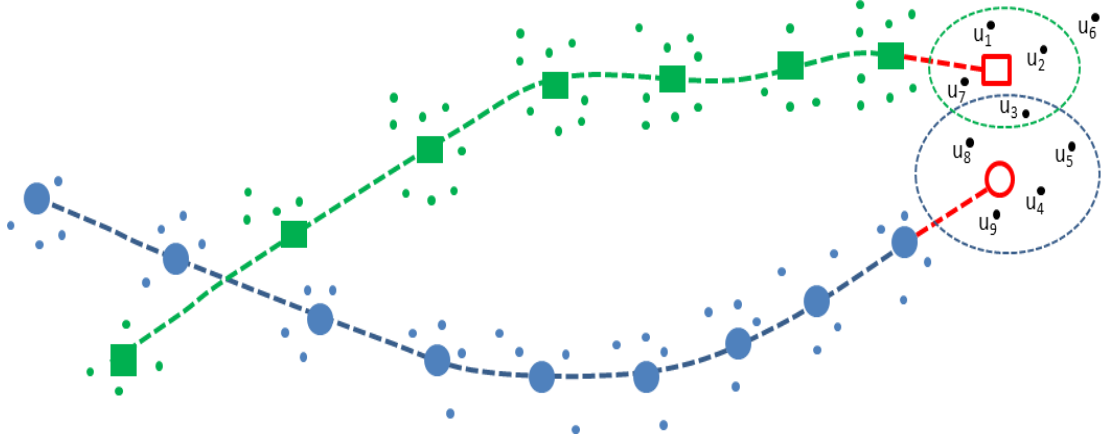


Figure 7. Scoring function Illustration

As shown in [2], the likelihood function (assuming Gaussian distribution for the residual) associated with assignment of observation j to track i is

$$g_{ij} = \frac{e^{-\frac{d_{ij}^2}{2}}}{(2\pi)^{M/2} \sqrt{|C_i|}} \quad (3-5)$$

Where, $|C_i|$ is determinant of the residual covariance matrix for track i , $d_{ij}^2 = y_{ij}^T \text{inv}(C_i) y_{ij}$, and y_{ij} is residual vector from observation j to track i .

To maximize the g_{ij} , by taking the logarithm, we derive the scoring criteria we want to minimize:

$$D_{ij}^2 = \ln|C_i| + d_{ij}^2 \quad (3-6)$$

3.5. Allocation Step

For measurements not associated with any track (that are outside of any existing gate), new group tracker is allocated and initialized. This is an iterative process, similar to DBSCAN clustering algorithm. It is significantly simpler, since it is only done for the leftover measurements.

We first select a leading measurement and set a centroid (range/angle) equal to it. The leading point's radial velocity is used to unroll other candidate's radial velocities. One candidate at a time we first check whether the point is within velocity bounds (velocity check, followed by a distance check. If passed, the centroid is recalculated, and point is added to the cluster. Once finished, we perform the few qualifying tests for cluster. We may need to see minimal number of measurements, strong enough combined SNR, and/or minimal amount of dynamicity of the centroid. If passed, we create (allocate) a new tracking object and use the associated points to initialize dispersion matrices. Clusters with fewer points are ignored.

3.6. Updating Step

This process is similar to the steps described in 2.2. Below we only outline the differences.

Tracks are updated based on the set of associated measurements computed at association step. For each track, we first compute $\bar{\mathbf{u}}(n)$ which is mean of all associated measurements.

Then, we compute the amount of innovation using the mean of all measurements associated with it:

$$\mathbf{y}(n) = \bar{\mathbf{u}}(n) - \mathbf{H}(\mathbf{s}_{apr}(n)) \quad (3-7)$$

Then, we compute residual centroid covariance matrix:

$$\mathbf{C}_c = \mathbf{J}_H(\mathbf{s}_{apr}(n)) \mathbf{P}_{apr}(n) \mathbf{J}_H^T(\mathbf{s}_{apr}(n)) + \mathbf{R}_c \quad (3-8)$$

where \mathbf{R}_c is a measurement noise covariance matrix, computed as below:

$$\mathbf{R}_c = \frac{R_m}{N_A} + f(N_A, \hat{N}) \hat{\mathbf{D}} \quad (3-9)$$

where,

$\hat{\mathbf{D}}$ is the estimation of group track dispersion matrix,

N_A is the number of measurements associated with given track,

\hat{N} is the estimated number of elements in target tracked by the given track,

$f(N_A, \hat{N})$ is a weighting factor that is a function of the number of observations and estimated number of elements in the target.

The term \mathbf{R}_m is the diagonal matrix of the measurement variances. We estimate those variances by iteratively computing the estimated spread of the associated point values across each measurement dimension. We make an assumption of uniform distribution of measurement values within the target dimension. We also make an assumption of “2 sigma” spread coverage to compute the variances.

From the equation above, measurement noise covariance matrix is the sum of two factors. The first term $\frac{\mathbf{R}_m}{N_A}$ represents the error in measuring the centroid due to radar measurement error and is decreased by the number of the measurements associated with tracking centroid. The second term represents the uncertainty due to the fact that not all the elements may have been observed. We define the weighting factor for the case with no false detections ($N_A \leq \hat{N}$) as:

$$f(N_A, \hat{N}) = \frac{\hat{N} - N_A}{(\hat{N} - 1)N_A} \quad (3-10)$$

Note the limits on this function: $f(N_A, \hat{N}) = 0$ when all elements are detected and $f(N_A, \hat{N}) = 1$ when we received single associated measurement.

We estimate \hat{N} as

$$\hat{N}_n = (1 - \alpha_N)\hat{N}_{n-1} + \alpha_N \hat{N}_n \quad (3-11)$$

We also estimate $\hat{\mathbf{D}}$ as described below.

For the measurement vector $\mathbf{u}(n) = [r(n) \quad \varphi(n) \quad \dot{r}(n)]^T$, the dispersion matrix is

$$\mathbf{D} = \begin{bmatrix} d_{rr}^2 & d_{r\varphi}^2 & d_{r\dot{r}}^2 \\ d_{r\varphi}^2 & d_{\varphi\varphi}^2 & d_{\varphi\dot{r}}^2 \\ d_{r\dot{r}}^2 & d_{\varphi\dot{r}}^2 & d_{\dot{r}\dot{r}}^2 \end{bmatrix} \quad (3-12)$$

Where $d_{ab}^2 \triangleq \frac{1}{N} \sum_{i=1}^N (a_i - \bar{a})(b_i - \bar{b})$, and $a, b \in \{r, \varphi, \dot{r}\}$, and \bar{a}, \bar{b} are means computed for a given group of measurements in a standard manner.

$$\hat{\mathbf{D}}_n = (1 - \alpha_D)\hat{\mathbf{D}}_{n-1} + \alpha_D \mathbf{D} \quad (3-13)$$

Once \mathbf{R}_C is computed, the rest steps (computing the Kalman gain, state and covariance matrices) are identical to in 2.2.

3.7. Maintenance Step

Each track goes through a life cycle of events. At maintenance step we may decide to change the state or to delete the track that is not used any more.

4. Implementation Details

4.1. GTRACK Library

Tracking algorithm is implemented as a library. Application task creates an algorithm instance with configuration parameters that describe sensor, scenery, and behavior of radar targets. Algorithm is called once per frame from Application Task context. It is possible to create multiple instances of group tracker.

The figure below explains the steps algorithm goes during each frame call. Algorithm inputs measurement data in Polar coordinates (range, angle, Doppler), and tracks objects in Cartesian space. Therefore we use Extended Kalman Filter (EKF) process.

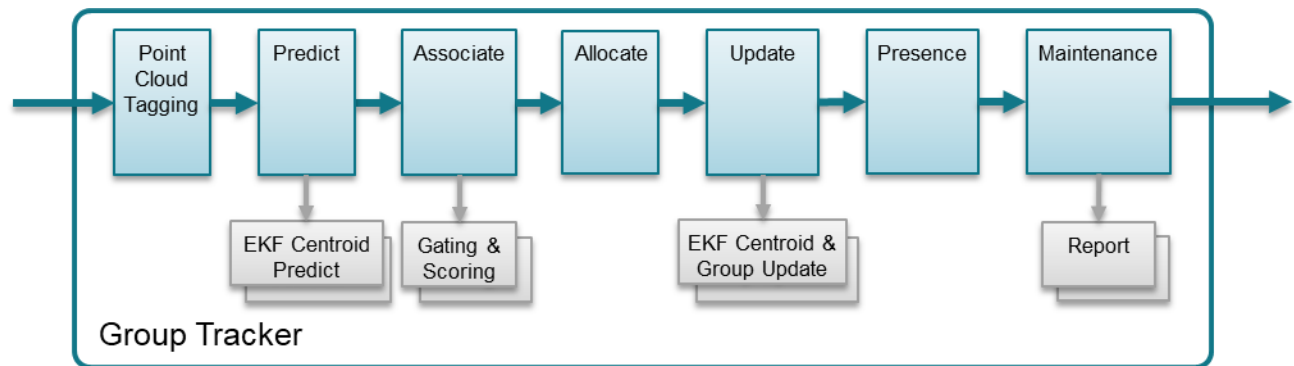


Figure 8. Group Tracking Algorithm

Point cloud input is first tagged based on scene boundaries. Some points may be get tagged as “outside the boundaries”, and will be ignored in association and allocation processes.

Predict function estimates tracking group centroid for time n based on state and process covariance matrices estimated at time $n-1$. We compute a-priori state and error covariance estimations for each trackable object. At this step we also compute measurement vector estimations.

Association function allows each tracking unit to indicate whether each measurement point is “close enough” (gating), and if it is, to provide the bidding value (scoring). Point is assigned to the highest bidder. Association function includes optional ghost tagging feature. In some configurations (ex. 2D Wall mounted), we observed significant amount of multipath reflection points behind the target. Once enabled, the hypothetical ghost also bids for the association. If won, the point is tagged as the “ghost” point and will not be assigned to any target. The resulted ghost points will be ignored.

Points that are not assigned are going through the Allocate function. During the Allocation process, points are first joined into a sets based on their proximity in measurement coordinates. Each set becomes a candidate for allocation decision. It has to pass multiple tests to become a new track. Once passed, the new tracking unit is allocated.

During Update step, tracks are updated based on the set of associated points. We compute the innovation, Kalman gain, and a-posteriori state vector and error covariance. In addition to classic EKF, the error covariance calculation includes group dispersion in measurement noise covariance matrix. If presence detection is enabled, the algorithm determines whether there is any target within the configured occupancy zone.

The Report function queries each tracking unit and produces the algorithm output.

4.2. Building and using the library


The algorithm is delivered as a source code with ready-to-build makefile infrastructure. Library can be built to support either 2D or 3D geometries. Application is expected to include the library (2D or 3D) as it needs. The configuration is passed to the algorithm instance at module create time. The configuration parameters are described in the below chapter. Recommended values for Traffic monitoring (TM) and People counting (PC) use cases are given as an example. Changes from previous document version are highlighted in red.

4.3. Configuration Parameters

The configuration parameters are used to configure Tracking algorithm. They shall be adjusted to match customer use case based on particular scenery and targets characteristics.


For up to date documentation, please check with the documentation auto-generated by doxygen tools. Below is a snapshot of the documentation based Tracker library version 0.112.

The configuration parameters are passed via *GTRACK_moduleConfig* structure and are described below:

 TEXAS INSTRUMENTS		Technology for Innovators™
GTRACK_moduleConfig Struct Reference		
GTRACK Algorithm Library » Externals » External Data Structures		
GTRACK Configuration. More...		
#include <gtrack.h>		
Data Fields		
GTRACK_STATE_VECTOR_TYPE	stateVectorType	State Vector Type, Supported Types are 2DA, S={X,Y, Vx,Vy, Ax,Ay} and 3DA, S={X,Y,Z, Vx,Vy,Vz, Ax,Ay,Az}.
GTRACK_VERBOSE_TYPE	verbose	Verboseness Level. A bit mask representing levels of verbosity: NONE WARNING DEBUG ASSOCIATION DEBUG GATE_DEBUG MATRIX_DEBUG Once event level is lower then requested verbosity level, Library generates a log by calling gtrack_log function.
uint16_t	maxNumPoints	Maximum Number of Measurement Points per frame. Up to GTRACK_NUM_POINTS_MAX supported The library will allocate memories based on this parameter.
uint16_t	maxNumTracks	Maximum Number of Tracking Objects. Up to GTRACK_NUM_TRACKS_MAX supported The library will allocate memories based on this parameter.
float	initialRadialVelocity	Expected target radial velocity at the moment of detection, m/s.
float	maxRadialVelocity	Maximum radial velocity reported by sensor +/- m/s.
float	radialVelocityResolution	Radial Velocity resolution, m/s.
float	maxAcceleration [3]	Maximum expected target acceleration in lateral (X), longitudinal (Y), and vertical (Z) directions, m/s ² Used to compute processing noise matrix. For 2D options, the vertical component is ignored.
float	delta T	Frame rate, ms.
uint16_t	boresightFilteringEnable	Flag to enable/disable boresight filtering. This is used only in ceiling mount mode and ignored in wall mount mode.
GTRACK_advancedParameters *	advParams	Advanced parameters, set to NULL for defaults.

4.3.1. Advanced parameters

Advanced parameters are divided into few sets. Each set can be omitted, and defaults will be used by the algorithm. Customers are expected to modify needed parameters to achieve better performance.

Technology for Innovators™

Main Page

Modules

Data Structures

Files

Data Structures

Data Fields

Data Fields

GTRACK_advancedParameters Struct Reference

External Data Structures

GTRACK Advanced Parameters. [More...](#)

#include <gtrack.h>

Data Fields

GTRACK_gatingParams *	gatingParams Pointer to gating parameters.
GTRACK_allocationParams *	allocationParams Pointer to allocation parameters.
GTRACK_stateParams *	stateParams Pointer to tracking state parameters.
GTRACK_sceneryParams *	sceneryParams Pointer to scenery parameters.
GTRACK_presenceParams *	presenceParams Pointer to presence detection parameters.

GTRACK_gatingParams *	gatingParams Pointer to gating parameters.
GTRACK_allocationParams *	allocationParams Pointer to allocation parameters.
GTRACK_stateParams *	stateParams Pointer to tracking state parameters.
GTRACK_sceneryParams *	sceneryParams Pointer to scenery parameters.
GTRACK_presenceParams *	presenceParams Pointer to presence detection parameters.

Scenery Parameters

This set of parameters describes the scenery. It allows user to configure the tracker with expected boundaries, and areas of static behavior. User can define up to 2 boundary boxes, and up to 2 static boxes. Sensor position and orientation are important for 3D configurations. See below for details

Technology for Innovators™

Main Page

Modules

Data Structures

Files

Data Structures

Data Fields

GTRACK_sceneryParams Struct Reference

External Data Structures

GTRACK Scenery Parameters. [More...](#)

```
#include <gtrack.h>
```

Data Fields

GTRACK_sensorPosition	sensorPosition
Sensor position, set to (0.f, 0.f) for 2D, set to (0.f, 0.f, H) for 3D. Where H is sensor height, in m.	
GTRACK_sensorOrientation	sensorOrientation
Sensor orientation, set to (0.f, 0.f) for 2D, (AzimTilt, ElevTilt) for 3D. Where AzimTilt and ElevTilt are rotations along Z and X axes correspondly.	
uint8_t	numBoundaryBoxes
Number of scene boundary boxes. If defined (numBoundaryBoxes > 0), only points within the boundary box(s) can be associated with tracks.	
GTRACK_boundaryBox	boundaryBox [GTRACK_MAX_BOUNDARY_BOXES]
Scene boundary boxes.	
uint8_t	numStaticBoxes
Number of scene static boxes. If defined (numStaticBoxes > 0), only targets within the static box(s) can persist as static.	
GTRACK_boundaryBox	staticBox [GTRACK_MAX_STATIC_BOXES]
Scene static boxes.	

Detailed Description

GTRACK Scenery Parameters.

Scenery uses 3-dimensional Cartesian coordinate system, defined as W in the picture below

It is expected that the $Z=0$ plane corresponds to the scene floor

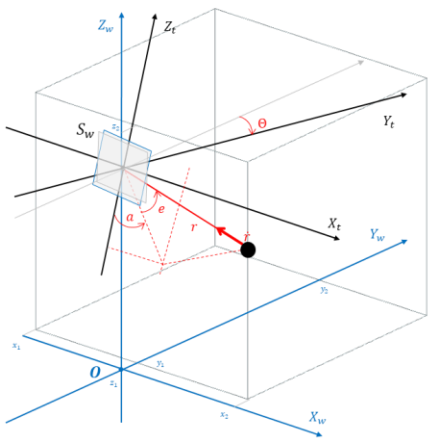
The X coordinates are left (negative)-right; the Y coordinates are near-far.

Origin (O) is typically colocated with sensor projection to $Z=0$ plane

- Sensor Position is 3 dimensional coordinate of the sensor
 - For example, (0,0,2) will indicate that sensor is directly above the origin at the height of 2m
- Sensor Orientation is sensor's boresight rotation: down tilt (theta) and azimuthal tilt (not supported)

User can define up to `GTRACK_MAX_BOUNDARY_BOXES` boundary boxes, and up to `GTRACK_MAX_STATIC_BOXES` static boxes.


- Boundary Boxes are used to define area of interest. All reflection points outside the boundary area are ignored.
 - For example, boundary box can be used to ignore the targets in the hallway outside the room, or to ignore potential ghost-behind-the-wall reflections
- Static Boxes defines the zone where targets are expected to become static for a long time. Typically, that area is a smaller (0.5-1.5m) then boundary
 - When not directly configured by customer, the application makes box 0.5m smaller from each side
 - When targets are within the static zone, and miss detection event occurs, the [larger] static threshold will apply for de-allocation. When outside the area, the [smaller] exit threshold will be applied
 - Static reflection points outside the static area are ignored.



- World is Cartesian space, $W \triangleq \{X_w, Y_w, Z_w\}$ with origin O at floor level.
- All configuration Boxes are configured by user using world coordinates
- Sensor location in the world is $S_w = \{0, 0, H\}$
- Point Cloud is Spherical, $P \triangleq \{r, \alpha, \theta\}$ with origin at sensor
- Tracker operates in Cartesian Space $T \triangleq \{X_t, Y_t, Z_t\}$, also sensor origin

Allocation Parameters

The reflection points reported in point cloud are associated with existing tracking instances. Points that don't get associated are subjects for the allocation decision. Each candidate point is clustered into an allocation set. To join the set, the point needs to be within `maxDistanceThre` and `maxVelThre` from the set's centroid. Once the set is formed, it has to have more than `pointsThre` members, and pass the minimal velocity and SNR thresholds.



Technology for Innovators™

Main Page

Modules

Data Structures

Files

Data Structures

Data Fields

GTRACK_allocationParams Struct Reference

External Data Structures

Data Fields

GTRACK Allocation Function Parameters. More...

```
#include <gtrack.h>
```

Data Fields

float

snrThre

Minimum total SNR of the allocation set-candidate. The threshold shall be roughly equal to the expected linear SNR value of the detection point at 6m range multiplied by pointsThre.

float

snrThreObscured

Minimum total SNR when behind another target. Set it larger then snrThre to require stronger SNR for the obscured allocations.

float

velocityThre

Minimum initial velocity, m/s.

uint16_t

pointsThre

Minimum number of points in a set.

float

maxDistanceThre

Maximum squared distance between points in a set.

float

maxVelThre

Maximum velocity delta between points in a set.

State Transition Parameters

Each tracking instance can be in either FREE, DETECT, or ACTIVE states. Once per frame the instance can get a HIT event (have non-zero points associated to a target instance) or MISS (no points associated) event.

Once in FREE state, the transition to DETECT state is made by the allocation decision. See previous section for the allocation decision configuration parameters.


Once in DETECT state, we use `det2active` threshold for the number of consecutive hits to transition to ACTIVE state, or `det2free` threshold of number of consecutive misses to transition back to FREE state.

Once in ACTIVE state, the target can be in either DYNAMIC (moving) or STATIC condition. When the MISS event occurs, the miss count is incremented. Then, handling of the MISS (no points associated) is as follow:

- If the target is in the “static zone” AND the target is in STATIC condition then the assumption is made that the reason we don’t have detection is because we removed them as “static clutter”. In this case we compare the miss count with [larger] `static2free` threshold to “extend the life expectation” of the static targets.
- If the target is outside the static zone, then the assumption is made that the reason we didn’t get the points is that target is exiting. In this case, we use `exit2free` threshold to quickly free the exiting targets.
- Otherwise, (meaning target is in the “static zone”, but has non-zero motion in radial projection) we assume that the reason of not having detections is that target got obscured by other targets. In this case, we continue target motion according to the model, and use `active2free` threshold.

On HIT event, the miss count is cleared.

In addition to the miss count, we maintain separate counter for STATIC condition. This counter is checked against the `sleep2free` threshold.

 **TEXAS INSTRUMENTS**

Technology for Innovators™

Main Page

Modules

Data Structures

Files

Data Structures

Data Fields

GTRACK_stateParams Struct Reference

External Data Structures

Data Fields

GTRACK Tracking Management Function Parameters. [More...](#)

#include <gtrack.h>

Data Fields

uint16_t	det2actThre	DETECTION => ACTIVE threshold. This is a threshold for the number of continuous HITS to transition from DETECT to ACTIVE state.
uint16_t	det2freeThre	DETECTION => FREE threshold. This is a threshold for the number of continuous misses to transition from DETECT to FREE state.
uint16_t	active2freeThre	ACTIVE => FREE threshold. This is a generic threshold for continuous misses in ACTIVE state when no special conditions (<code>static2free</code> / <code>exit2free</code>) apply. The corresponding counter is reset with either static or dynamic points associated.
uint16_t	static2freeThre	ACTIVE & STATIC & STATIC_ZONE => FREE threshold. The threshold is for continuous misses for static target in static zone.
uint16_t	exit2freeThre	ACTIVE & !STATIC_ZONE => FREE threshold. This is a threshold for continuous misses for target outside of static zone.
uint16_t	sleep2freeThre	ACTIVE & STATIC & STATIC_ZONE => FREE threshold. This is a maximum time target can be STATIC. There is separate counter that is reset only with dynamic point associated.

Gating Parameters

Gating parameters set is used in association process to provide a boundary for the points that can be associated with a given track. These parameters are target-specific. For example, for people counting, it is expected that the limits are set based on human body dimensions and dynamicity limits: (ex, 1.5x1.5x2 m in depth x width x height, and 4m/s of Doppler spread). The gain is expected to be around 3 (ex. based on “three sigma rule”).

 TEXAS INSTRUMENTS		Technology for Innovators™	
Main Page	Modules	Data Structures	Files
Data Structures	Data Fields		

GTRACK_gatingParams Struct Reference

External Data Structures

Data Fields

GTRACK Gating Function Parameters. More...

```
#include <gtrack.h>
```

Data Fields

float	gain	Gain of the gating function. It is set based on expected tracking errors and uncertainties of detection layer.
GTRACK_gateLimits	limits	Gating function limits. It is based on physical dimensions and agility of the targets. Setting it too small will result in allocating multiple tracks for the single object, setting it too big will cause allocating single track for multiple objects.

 TEXAS INSTRUMENTS		Technology for Innovators™	
Main Page	Modules	Data Structures	Files
Data Structures	Data Fields		

GTRACK_gateLimits Struct Reference

External Data Structures

Data Fields

GTRACK Gate Limits. More...

```
#include <gtrack.h>
```

Data Fields

float	depth	Depth limit, m.
float	width	Width limit, m.
float	height	Height limit, m.
float	vel	Radial velocity limit, m/s.

Presence Detection Parameters

The algorithm combines "raw detection" and "known target in the area" indications. For "raw detection" indication it re-uses the candidate set created by allocation process. It checks the set against the occupancy thresholds: number of points in the set against the pointsThre and set's velocity against the velocityThre. For "known target in the area" the algorithm checks whether known target measurement centroid is within the occupancy boxes.

TEXAS INSTRUMENTS

Technology for Innovators™

Main Page

Modules

Data Structures

Files

Data Structures

Data Fields

GTRACK_presenceParams Struct Reference

External Data Structures

Data Fields

GTRACK Presence Detection Parameters. More...

#include <gtrack.h>

Data Fields

uint16_t

pointsThre

occupancy threshold, number of points. Setting pointsThre to 0 disables presence detection

float

velocityThre

occupancy threshold, approaching velocity

uint16_t

on2offThre

occupancy on to off threshold

uint8_t

numOccupancyBoxes

Number of occupancy boxes. Presence detection algorithm will determine whether the combined shape is occupied. Setting numOccupancyBoxes to 0 disables presence detection.

GTRACK_boundaryBox

occupancyBox [GTRACK_MAX_OCCUPANCY_BOXES]

Scene occupancy boxes.

Detailed Description

GTRACK Presence Detection Parameters.

This set of parameters describes the presence detection function

Presence is computed over the combined shape of occupancy boxes. Each box is described using 3-dimensional Cartesian coordinate system. It is expected that the Z=0 plane corresponds to the scene floor. The X coordinates are left (negative)-right; the Y coordinates are near-far. Origin is typically colocated with sensor projection to Z=0 plane. User can define up to **GTRACK_MAX_OCCUPANCY_BOXES** occupancy boxes.

If any target exists with the occupancy area, the algorithm returns 1. Otherwise, it returns 0.

The algorithm combines "raw detection" and "known target in the area" indications.

For "raw detection" indication re-uses the candidate set created by allocation process. It checks the occupancy thresholds: number of points in the set against the pointsThre and set's velocity against the velocityThre.

For "known target in the area" the algorithm checks whether known target measurement centroid is within the occupancy boxes.

4.4. Memory Requirements

Initial memory footprint for group tracker implementation with 250 measurement points and 20 tracks is 80KB of data and 20KB of program space.

4.4.1. Data Memory

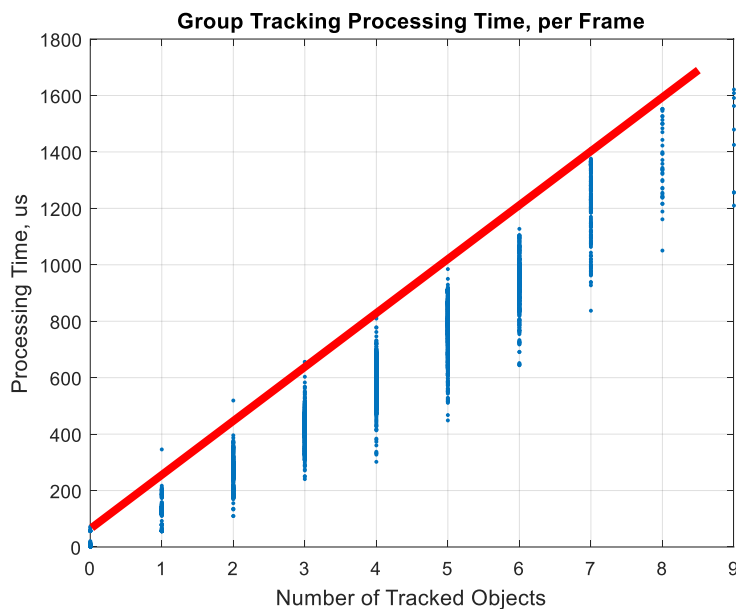
NumTracks	20				
NumPoints	250				
	Size, B	Element Name	Element Type	Size	Num
TrackModule					
	544	inst	GtrackModuleInstance	544	1
	80	inst->hTracks	void *	4	20
	1000	inst->bestScore	float	4	250
	500	inst->bestIndex	uint16_t	2	250
	6	inst->allocIndex	uint16_t	2	3
	240	inst->tidElem	GTrack_ListElem	12	20
	720	inst->targetDesc	GTRACK_targetDesc	36	20
	40	inst->targetInd	uint16_t	2	20
SubTotal	3130				
TrackUnit					
	11520	inst	GtrackUnitInstance	576	20
SubTotal	11520				
Total, Bytes	14650				

4.4.1. Program Memory

	text	data	bss	dec	hex	filename
	212	0	0	212	d4	gtrackListlib.oer4f
	252	0	0	252	fc	gtrackModuleConstants.oer4f
	1980	0	0	1980	7bc	gtrackModuleCreate.oer4f
	1882	0	0	1882	75a	gtrackModuleStep.oer4f
	1182	0	0	1182	49e	gtrackUnitCreate.oer4f
	320	0	0	320	140	gtrackUnitEvent.oer4f
	442	0	0	442	1ba	gtrackUnitPredict.oer4f
	30	0	0	30	1e	gtrackUnitReport.oer4f
	1771	0	0	1771	6eb	gtrackUnitScore.oer4f
	328	0	0	328	148	gtrackUnitStart.oer4f

	196	0	0	196	c4	gtrackUnitStop.oer4f
	2186	0	0	2186	88a	gtrackUnitUpdate.oer4f
	1586	0	0	1586	632	gtrackUtilities.oer4f
	2514	0	0	2514	9d2	matrixMath.oer4f
Total	14881					

4.5. Benchmarks



Tracking processing cost $\approx 200\mu\text{s}$ per tracking object

5. Performance

Tracking performance can be intuitively expressed in three classes of metrics:

1. Tracking Reliability, which shows how many mistakes the tracker made in terms of missing tracks, false positives, mismatches, failures to recover tracks, etc.
2. Tracking Precision, which expresses how well exact features of the correctly tracked objects are estimated,
3. Tracking Resolution, which expresses how much feature separation needed between two objects to be tracked separately

5.1. Tracking Reliability

The test case to measure tracking reliability is simulated randomized longevity test. We modeled tracker performance in traffic monitoring usage case.

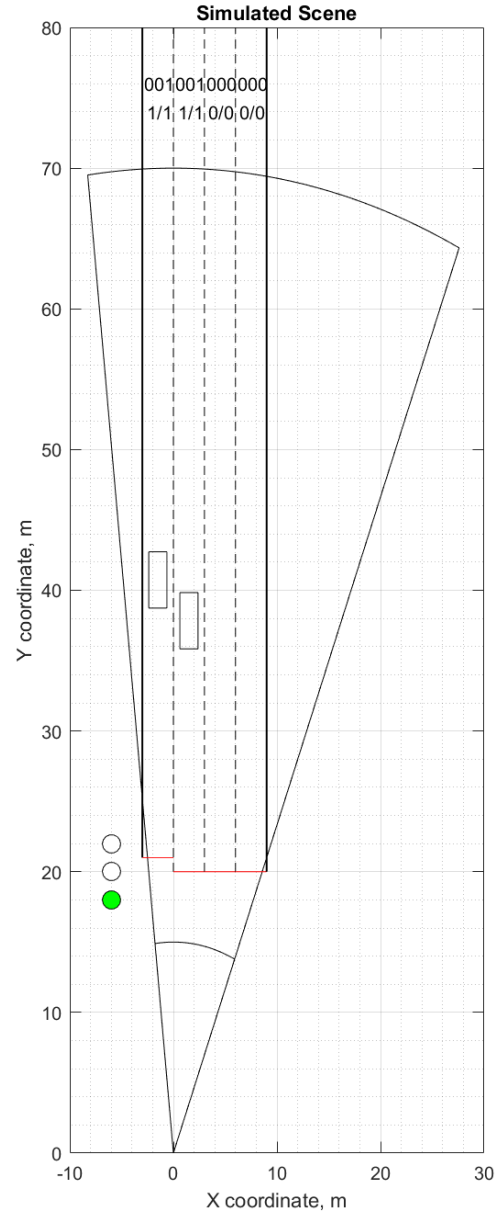
5.1.1. Test Case description

The sensor geometry is illustrated in the picture below. Traffic is simulated by random Poisson process, where vehicles arrive into 4-Lane intersection at random velocities. The intersection is controlled by the traffic light with predetermined latencies for green, yellow, and red signals. The stop line is at 20m from the sensor. Vehicles movement is modeled with following rules:

1. Each vehicle maintains constant velocity unless there is an obstacle in front. The obstacle can be another vehicle in front or either red or yellow traffic light.
2. With obstacle in front, the vehicle will decelerate to match the obstacle speed. The amount of deceleration is bounded to a maximum allowed for the given vehicle type.
3. The vehicles preserve a safety 2m distance.
4. Once obstacle cleared, the vehicles accelerate. The acceleration step and maximum amount of acceleration is configured per vehicle type.

The reliability test simulates at least 10 minutes of traffic.

Sensor output (the point cloud) is randomly generated and is modelled after the test data captured from the moving vehicle. The model includes statistical modeling of number of reflection points, points distribution in range/angle space, and distribution of doppler information in a cloud. Each vehicle at any time instance is represented by a point cloud (the set of reflection points with range/angle and radial velocity information).



Note: Point cloud from multiple vehicles is assumed additive, so we didn't model the point cloud degradation due to multiple objects.

For the point cloud synthesis, we collected reflection points from the field test in two detection layer configurations:

- Configuration A, with “maximum possible” reflections (single range CFAR-CASO, no windowing in doppler dimension, 0.2deg DoA step)
- Configuration B, with about 1/3 of reflections (dual pass range/doppler CFAR-CASO, 0.2deg DoA)

The tracker inputs the points cloud, associates reflection points to the tracked objects, predicts, and models the objects behavior, and outputs estimated object properties to an upper layer.

Each configuration is run in two subcases: with

The upper layer processing estimates the tracking error by comparing the ground truth with tracker output. As a ground truth we consider a centroid of the rectangle of simulated vehicle. For each track, based on global time stamp, we synchronize the ground truth point and tracker output. We compute mean square error individually for all time instances the tracker existed. We declare correct tracking of the object if at any point the mean square error did not exceed the desired threshold, and tracking error event otherwise.

In addition, upper layer processing counts number of objects at 25m lane. This count is compared to a ground truth number as well.

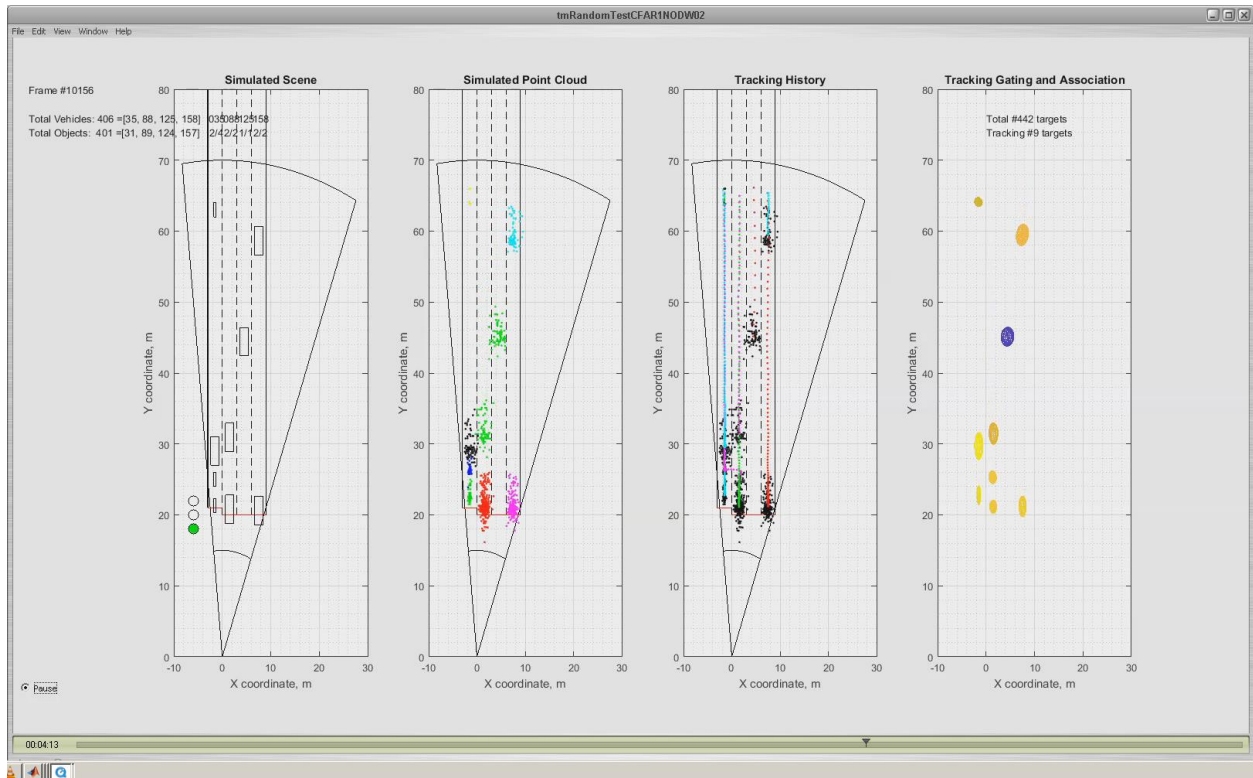
5.1.2. Results

The results are summarized below

Configuration A results

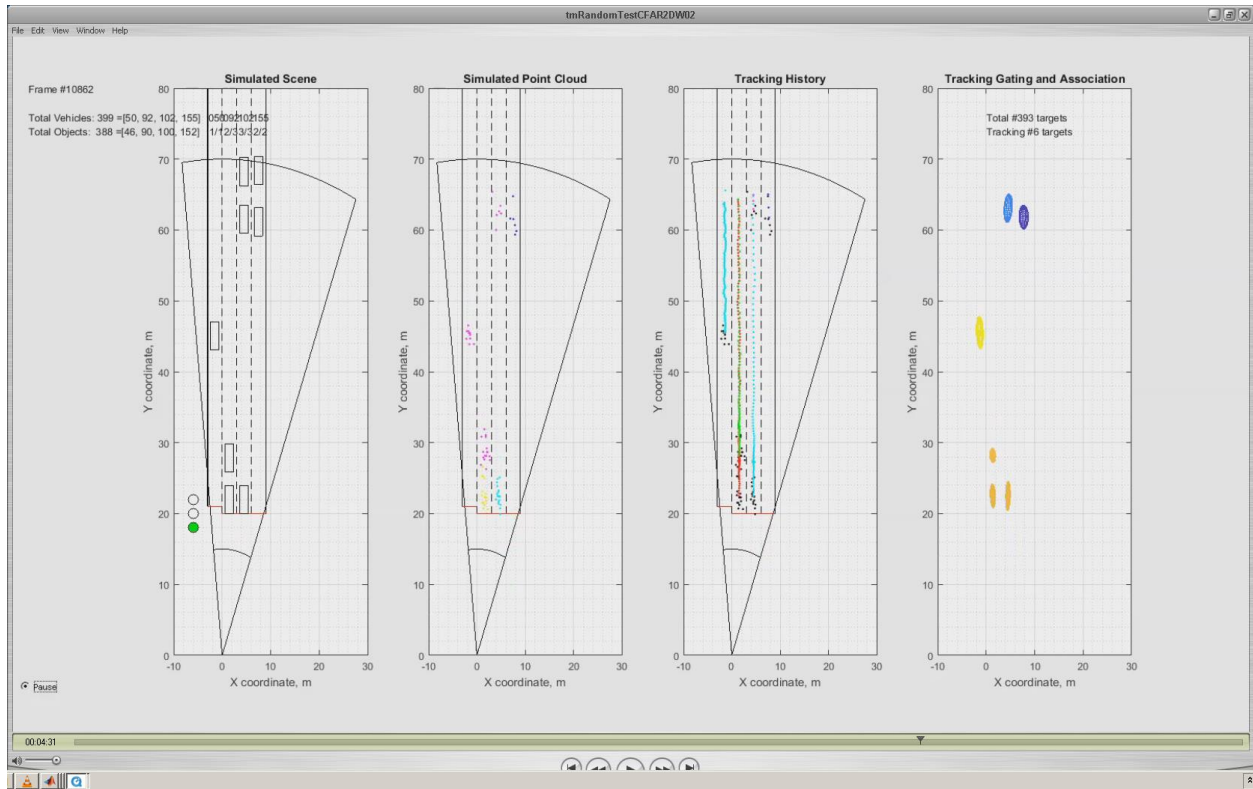
Lane	Number of Vehicles	Number of Counted objects (at 25m)	Counting Reliability, %	Number of correctly tracked objects	Tracking Reliability, %
1	58	58		56	
2	117	122		110	
3	177	184		173	
4	228	233		216	
Total	580	597	99.5	555	95.7

TODO: Due to changes in unrolling functions, the need to Re-run the configuration A tests



Configuration B results

Lane	Number of Vehicles	Number of Counted objects (at 25m)	Counting Reliability	Number of correctly tracked objects	Tracking Reliability, %
1	61	60		57	
2	124	122		110	
3	150	147		126	
4	211	208		194	
Total	546	537	98.4	487	89.4



Test is reproducible with this random seed:

Test outputs (the 10-minute movie) is saved here:

The error analysis shows that most of the errors fall into two cases:

- a) errors due to erroneous split, i.e. at some time instance the tracker observed too much dispersion within a group, and decided to split a group into two distinct objects
- b) errors due to original acquisition, i.e. the track was started with significant deviation from the ground truth, and wasn't able to converge fast enough

5.2. Tracking Precision

Tracking Precision expresses how well exact features of the correctly tracked objects are estimated. This score is independent of tracker ability to identify and follow the objects. For the correctly tracked object, we compute:

1. Position Tracking Precision. This is the measure of expected positional error (mean and deviation) for matched objects
2. Velocity Tracking Precision. This is the measure of expected velocity error (mean and deviation) for matched objects. Shows the ability of the tracker to estimate object velocity
3. Size Tracking Precision. This is the measure of expected dimensional error (mean and deviation) for matched objects. Shows the ability of the tracker to estimate object size

5.2.1. Test Case description

For tracking precision, we use the same randomized longevity test case as described above. We take only the “successfully tracked” objects, and derive the error statistics for those cases, comparing the tracked object with the ground truth.

5.2.2. Results

For successfully tracked objects (95% cases, see the Tracking Reliability Test results), the following was observed:

Configuration A results

Tracking Precision (% of cases)	Error, Standard Deviation at 40m of Range
Latitude (X position), m	
Longitude (Y position), m	
Velocity, Latitudal, m/s	
Velocity, Longitudal, m/s	

Configuration B results

Tracking precision (89.4 % of cases)	Error, Standard Deviation at 40m of Range
Latitude (X position), m	0.11
Longitude (Y position), m	0.36
Velocity, Latitudal, m/s	0.99
Velocity, Longitudal, m/s	0.4

5.3. Tracking Resolution

Tracking Resolution expresses how much of feature separation is needed between two real life objects to be tracked separately. The feature separation can be measured in Range/Angle/Velocity. We designed separate test cases for initial separation, and dynamic (split).

5.3.1. Initial Separation Tests

Test Case description

We designed three separate test cases for Range, Angular, and radial velocity separation. We simulate the vehicle entry with

- Range difference (same lane, two vehicles, same velocity, one after each other at given distance),

- b) Angle difference (same range, different lanes, same velocity)
- c) Velocity difference (adjacent lanes, minimal angular difference, different velocities)

For all tests we are seeking a minimal separation needed for a tracker to allocate and successfully track two objects as separate ones in more than 95% of cases.

Results

	Test Case Range	Test Case Angle	Test Case Velocity
Range Difference, m	4	0	0
Angular Separation, deg	0	4	0
Radial Velocity, m/s	0	0	4
Test Success Rate	>95%	>95%	>95%

5.3.2. Dynamic Separation Tests

Test Case description

Separate test is designed, where at entrance, two objects are not distinguishable. The tracker shall allocate a single tracking object. However, due to velocity difference, after few tens of frames, tracker shall be able to trigger a split event and track two targets. We measure the amount of separation required (in Range AND Angle AND Velocity) when tracker performs split in >95% cases.

Results

	Test Case Split
Range Difference, m	4
Angular Separation, deg	4
Radial Velocity, m/s	4
Test Success Rate	>95%

6. References

- [1] <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>
- [2] Samuel S. Blackman. Multi-Target Tracking with Radar Applications, Artech House, 1986
- [3] Muhammad Ikram, Nano radar DSP Algorithm Module, Tracking module description

7. Appendix

7.1. Evaluating Partial Derivatives for 2D space tracking

We need to calculate set of partial derivatives to compute Jacobian

$$J_H(s) = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial \dot{y}} & \frac{\partial r}{\partial \ddot{x}} & \frac{\partial r}{\partial \ddot{y}} \\ \frac{\partial \varphi}{\partial x} & \frac{\partial \varphi}{\partial y} & \frac{\partial \varphi}{\partial \dot{x}} & \frac{\partial \varphi}{\partial \dot{y}} & \frac{\partial \varphi}{\partial \ddot{x}} & \frac{\partial \varphi}{\partial \ddot{y}} \\ \frac{\partial \dot{r}}{\partial x} & \frac{\partial \dot{r}}{\partial y} & \frac{\partial \dot{r}}{\partial \dot{x}} & \frac{\partial \dot{r}}{\partial \dot{y}} & \frac{\partial \dot{r}}{\partial \ddot{x}} & \frac{\partial \dot{r}}{\partial \ddot{y}} \end{bmatrix}$$

7.1.1. Evaluating range partial derivatives

$$r = \sqrt{x^2 + y^2}$$

$$\frac{\partial r}{\partial x} = \frac{\partial}{\partial x} (\sqrt{x^2 + y^2}) = \frac{x}{\sqrt{x^2 + y^2}}$$

$$\frac{\partial r}{\partial y} = \frac{\partial}{\partial y} (\sqrt{x^2 + y^2}) = \frac{y}{\sqrt{x^2 + y^2}}$$

7.1.2. Evaluating azimuth partial derivatives

$$\frac{\partial \varphi}{\partial x} = \frac{\partial}{\partial x} \left(\tan^{-1} \left(\frac{y}{x} \right) \right)$$

$$\text{Let } w = \tan^{-1} u \text{ with } u = \frac{y}{x}$$

$$\text{Using the chain rule: } \frac{\partial \varphi}{\partial x} = \frac{dw}{du} \frac{\partial u}{\partial x} = \left(\frac{1}{1+u^2} \right) \left(\frac{1}{y} \right) = \frac{1}{1+\left(\frac{y}{x}\right)^2} \frac{1}{y} = \frac{y}{x^2+y^2}$$

$$\text{Using the chain rule: } \frac{\partial \varphi}{\partial y} = \frac{dw}{du} \frac{\partial u}{\partial y} = \left(\frac{1}{1+u^2} \right) \left(\frac{-x}{y^2} \right) = \frac{1}{1+\left(\frac{y}{x}\right)^2} \frac{-x}{y^2} = -\frac{x}{x^2+y^2}$$

7.1.1. Evaluating doppler partial derivatives

$$\dot{r} = \frac{x\dot{x} + y\dot{y}}{\sqrt{x^2 + y^2}}$$

$$\frac{\partial \dot{r}}{\partial x} = \frac{\partial}{\partial x} \left(\frac{x\dot{x} + y\dot{y}}{\sqrt{x^2 + y^2}} \right) = \dot{x} \frac{\partial}{\partial x} \left(\frac{x}{\sqrt{x^2 + y^2}} \right) + y\dot{y} \frac{\partial}{\partial x} \left(\frac{1}{\sqrt{x^2 + y^2}} \right)$$

Using the quotient rule $\frac{\partial}{\partial x} \left(\frac{f}{g} \right) = \frac{\frac{\partial f}{\partial x} g - f \frac{\partial g}{\partial x}}{g^2}$, solving the first part $\frac{\partial}{\partial x} \left(\frac{x}{\sqrt{x^2+y^2}} \right) = \frac{\sqrt{x^2+y^2} - \frac{x}{\sqrt{x^2+y^2}} x}{(\sqrt{x^2+y^2})^2} = \frac{y^2}{(x^2+y^2)^{3/2}}.$

For the second part, $\frac{\partial}{\partial x} \left(\frac{1}{\sqrt{x^2+y^2}} \right) = -\frac{1}{2} \frac{2x}{(x^2+y^2)^{3/2}} = -\frac{x}{(x^2+y^2)^{3/2}}.$

$$\frac{\partial}{\partial x} \left(\frac{x\dot{x} + y\dot{y}}{\sqrt{x^2+y^2}} \right) = \dot{x} \frac{y^2}{(x^2+y^2)^{3/2}} - y\dot{y} \frac{x}{(x^2+y^2)^{3/2}} = \frac{y(\dot{x}y - \dot{y}x)}{(x^2+y^2)^{3/2}}$$

Similarly,

$$\begin{aligned} \frac{\partial \dot{r}}{\partial y} &= \frac{\partial}{\partial y} \left(\frac{x\dot{x} + y\dot{y}}{\sqrt{x^2+y^2}} \right) = \dot{y} \frac{\partial}{\partial y} \left(\frac{y}{\sqrt{x^2+y^2}} \right) + x\dot{x} \frac{\partial}{\partial y} \left(\frac{1}{\sqrt{x^2+y^2}} \right) = \dot{y} \frac{x^2}{(x^2+y^2)^{3/2}} - x\dot{x} \frac{y}{(x^2+y^2)^{3/2}} \\ &= \frac{x(\dot{y}x - \dot{x}y)}{(x^2+y^2)^{3/2}} \end{aligned}$$

$$\frac{\partial \dot{r}}{\partial \dot{x}} = \frac{\partial}{\partial \dot{x}} \left(\frac{x\dot{x} + y\dot{y}}{\sqrt{x^2+y^2}} \right) = \frac{x}{\sqrt{x^2+y^2}}$$

$$\frac{\partial \dot{r}}{\partial \dot{y}} = \frac{\partial}{\partial \dot{y}} \left(\frac{x\dot{x} + y\dot{y}}{\sqrt{x^2+y^2}} \right) = \frac{y}{\sqrt{x^2+y^2}}$$

Putting all together:

$$J_H(s) = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 & 0 & 0 \\ \frac{y}{x^2+y^2} & -\frac{x}{x^2+y^2} & 0 & 0 & 0 & 0 \\ \frac{y(\dot{x}y - \dot{y}x)}{(x^2+y^2)^{3/2}} & \frac{x(\dot{y}x - \dot{x}y)}{(x^2+y^2)^{3/2}} & \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 \end{bmatrix}$$

7.2. Evaluating Partial Derivatives for 3D space tracking

We need to calculate set of partial derivatives to compute Jacobians for 3DV and 3DA models:

$$J_H(s_{3DV}) = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial z} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial \dot{y}} & \frac{\partial r}{\partial \dot{z}} \\ \frac{\partial \varphi}{\partial x} & \frac{\partial \varphi}{\partial y} & \frac{\partial \varphi}{\partial z} & \frac{\partial \varphi}{\partial \dot{x}} & \frac{\partial \varphi}{\partial \dot{y}} & \frac{\partial \varphi}{\partial \dot{z}} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial y} & \frac{\partial \theta}{\partial z} & \frac{\partial \theta}{\partial \dot{x}} & \frac{\partial \theta}{\partial \dot{y}} & \frac{\partial \theta}{\partial \dot{z}} \\ \frac{\partial \dot{r}}{\partial x} & \frac{\partial \dot{r}}{\partial y} & \frac{\partial \dot{r}}{\partial z} & \frac{\partial \dot{r}}{\partial \dot{x}} & \frac{\partial \dot{r}}{\partial \dot{y}} & \frac{\partial \dot{r}}{\partial \dot{z}} \end{bmatrix}$$

and

$$J_H(s_{3DA}) = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial z} & \frac{\partial r}{\partial \ddot{x}} & \frac{\partial r}{\partial \ddot{y}} & \frac{\partial r}{\partial \ddot{z}} & \frac{\partial r}{\partial \ddot{\ddot{x}}} & \frac{\partial r}{\partial \ddot{\ddot{y}}} & \frac{\partial r}{\partial \ddot{\ddot{z}}} \\ \frac{\partial \varphi}{\partial x} & \frac{\partial \varphi}{\partial y} & \frac{\partial \varphi}{\partial z} & \frac{\partial \varphi}{\partial \ddot{x}} & \frac{\partial \varphi}{\partial \ddot{y}} & \frac{\partial \varphi}{\partial \ddot{z}} & \frac{\partial \varphi}{\partial \ddot{\ddot{x}}} & \frac{\partial \varphi}{\partial \ddot{\ddot{y}}} & \frac{\partial \varphi}{\partial \ddot{\ddot{z}}} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial y} & \frac{\partial \theta}{\partial z} & \frac{\partial \theta}{\partial \ddot{x}} & \frac{\partial \theta}{\partial \ddot{y}} & \frac{\partial \theta}{\partial \ddot{z}} & \frac{\partial \theta}{\partial \ddot{\ddot{x}}} & \frac{\partial \theta}{\partial \ddot{\ddot{y}}} & \frac{\partial \theta}{\partial \ddot{\ddot{z}}} \\ \frac{\partial \dot{r}}{\partial x} & \frac{\partial \dot{r}}{\partial y} & \frac{\partial \dot{r}}{\partial z} & \frac{\partial \dot{r}}{\partial \ddot{x}} & \frac{\partial \dot{r}}{\partial \ddot{y}} & \frac{\partial \dot{r}}{\partial \ddot{z}} & \frac{\partial \dot{r}}{\partial \ddot{\ddot{x}}} & \frac{\partial \dot{r}}{\partial \ddot{\ddot{y}}} & \frac{\partial \dot{r}}{\partial \ddot{\ddot{z}}} \end{bmatrix}$$

7.2.1. Evaluating range partial derivatives

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\frac{\partial r}{\partial x} = \frac{\partial}{\partial x} \left(\sqrt{x^2 + y^2 + z^2} \right) = \frac{x}{\sqrt{x^2 + y^2 + z^2}}$$

$$\frac{\partial r}{\partial y} = \frac{\partial}{\partial y} \left(\sqrt{x^2 + y^2 + z^2} \right) = \frac{y}{\sqrt{x^2 + y^2 + z^2}}$$

$$\frac{\partial r}{\partial z} = \frac{\partial}{\partial z} \left(\sqrt{x^2 + y^2 + z^2} \right) = \frac{z}{\sqrt{x^2 + y^2 + z^2}}$$

7.2.2. Evaluating azimuth partial derivatives

$$\frac{\partial \varphi}{\partial x} = \frac{\partial}{\partial x} \left(\tan^{-1} \left(\frac{y}{x} \right) \right)$$

$$\text{Let } w = \tan^{-1} u \text{ with } u = \frac{y}{x}$$

Using the chain rule: $\frac{\partial \phi}{\partial x} = \frac{dw}{du} \frac{\partial u}{\partial x} = \left(\frac{1}{1+u^2} \right) \left(\frac{1}{y} \right) = \frac{1}{1+\left(\frac{x}{y}\right)^2} \frac{1}{y} = \frac{y}{x^2+y^2}$

Using the chain rule: $\frac{\partial \phi}{\partial y} = \frac{dw}{du} \frac{\partial u}{\partial y} = \left(\frac{1}{1+u^2} \right) \left(\frac{-x}{y^2} \right) = \frac{1}{1+\left(\frac{x}{y}\right)^2} \frac{-x}{y^2} = -\frac{x}{x^2+y^2}$

$$\frac{\partial \phi}{\partial z} = 0$$

7.2.1. Evaluating elevation partial derivatives

$$\theta = \tan^{-1} \left(\frac{z}{\sqrt{x^2 + y^2}} \right)$$

Let $w = \tan^{-1} u$ with $u = \frac{z}{\sqrt{x^2+y^2}}$

Using the chain rule: $\frac{\partial \theta}{\partial x} = \frac{dw}{du} \frac{\partial u}{\partial x} = \left(\frac{1}{1+u^2} \right) \frac{\partial u}{\partial x} = \frac{1}{1+\frac{z^2}{x^2+y^2}} \left(-\frac{1}{2} \right) 2x \frac{z}{(x^2+y^2)^{3/2}} = -\frac{x^2+y^2}{x^2+y^2+z^2} \frac{xz}{(x^2+y^2)^{3/2}} =$
 $-\frac{x}{(x^2+y^2+z^2)} \frac{z}{\sqrt{x^2+y^2}} = -\frac{x}{r^2} \tan \theta$

Using the chain rule: $\frac{\partial \theta}{\partial y} = \frac{dw}{du} \frac{\partial u}{\partial y} = \left(\frac{1}{1+u^2} \right) \frac{\partial u}{\partial y} = \frac{1}{1+\frac{z^2}{x^2+y^2}} \left(-\frac{1}{2} \right) 2y \frac{z}{(x^2+y^2)^{3/2}} = -\frac{x^2+y^2}{x^2+y^2+z^2} \frac{yz}{(x^2+y^2)^{3/2}} =$
 $-\frac{y}{(x^2+y^2+z^2)} \frac{z}{\sqrt{x^2+y^2}} = -\frac{y}{r^2} \tan \theta$

Using the chain rule: $\frac{\partial \theta}{\partial z} = \frac{dw}{du} \frac{\partial u}{\partial z} = \left(\frac{1}{1+u^2} \right) \frac{\partial u}{\partial z} = \frac{1}{1+\frac{z^2}{x^2+y^2}} \frac{1}{\sqrt{x^2+y^2}} = \frac{x^2+y^2}{x^2+y^2+z^2} \frac{1}{\sqrt{x^2+y^2}} = \frac{\sqrt{x^2+y^2}}{(x^2+y^2+z^2)} =$
 $\frac{\sqrt{x^2+y^2}}{r^2}$

7.2.2. Evaluating Doppler partial derivatives

$$\dot{r} = \frac{x\dot{x} + y\dot{y} + z\dot{z}}{\sqrt{x^2 + y^2 + z^2}}$$

$$\frac{\partial \dot{r}}{\partial x} = \frac{\partial}{\partial x} \left(\frac{x\dot{x} + y\dot{y} + z\dot{z}}{\sqrt{x^2 + y^2 + z^2}} \right) = \dot{x} \frac{\partial}{\partial x} \left(\frac{x}{\sqrt{x^2 + y^2 + z^2}} \right) + (y\dot{y} + z\dot{z}) \frac{\partial}{\partial x} \left(\frac{1}{\sqrt{x^2 + y^2 + z^2}} \right);$$

Using the quotient rule $\frac{\partial}{\partial x} \left(\frac{f}{g} \right) = \frac{\frac{\partial f}{\partial x} g - f \frac{\partial g}{\partial x}}{g^2}$, solving $\frac{\partial}{\partial x} \left(\frac{x}{\sqrt{x^2+y^2+z^2}} \right) = \frac{\sqrt{x^2+y^2+z^2} - \frac{x}{\sqrt{x^2+y^2+z^2}} x}{x^2+y^2+z^2} =$
 $\frac{y^2+z^2}{(x^2+y^2+z^2)^{3/2}};$

Using chain rule, $\frac{\partial}{\partial x} \left(\frac{1}{\sqrt{x^2+y^2+z^2}} \right) = -\frac{1}{2} \frac{2x}{(x^2+y^2+z^2)^{3/2}} = -\frac{x}{(x^2+y^2+z^2)^{3/2}};$

$$\begin{aligned}\frac{\partial}{\partial x} \left(\frac{x\dot{x} + y\dot{y} + z\dot{z}}{\sqrt{x^2 + y^2 + z^2}} \right) &= \dot{x} \frac{y^2 + z^2}{(x^2 + y^2 + z^2)^{3/2}} - (y\dot{y} + z\dot{z}) \frac{x}{(x^2 + y^2 + z^2)^{3/2}} = \\ &= \frac{y(\dot{x}y - \dot{y}x) + z(\dot{x}z - \dot{z}x)}{(x^2 + y^2 + z^2)^{3/2}};\end{aligned}$$

Due to symmetry:

$$\frac{\partial \dot{r}}{\partial y} = \frac{x(\dot{y}x - \dot{x}y) + z(\dot{y}z - \dot{z}y)}{(x^2 + y^2 + z^2)^{3/2}};$$

$$\frac{\partial \dot{r}}{\partial z} = \frac{x(\dot{z}x - \dot{x}z) + y(\dot{z}y - \dot{y}z)}{(x^2 + y^2 + z^2)^{3/2}};$$

$$\frac{\partial \dot{r}}{\partial \dot{x}} = \frac{\partial}{\partial \dot{x}} \left(\frac{x\dot{x} + y\dot{y} + z\dot{z}}{\sqrt{x^2 + y^2 + z^2}} \right) = \frac{x}{\sqrt{x^2 + y^2 + z^2}};$$

$$\frac{\partial \dot{r}}{\partial \dot{y}} = \frac{\partial}{\partial \dot{y}} \left(\frac{x\dot{x} + y\dot{y} + z\dot{z}}{\sqrt{x^2 + y^2 + z^2}} \right) = \frac{y}{\sqrt{x^2 + y^2 + z^2}};$$

$$\frac{\partial \dot{r}}{\partial \dot{z}} = \frac{\partial}{\partial \dot{z}} \left(\frac{x\dot{x} + y\dot{y} + z\dot{z}}{\sqrt{x^2 + y^2 + z^2}} \right) = \frac{z}{\sqrt{x^2 + y^2 + z^2}};$$

Putting all together:

$$\begin{aligned}J_H(s_{3DV}) &= \\ &= \begin{bmatrix} \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 \\ \frac{y}{x^2 + y^2} & -\frac{x}{x^2 + y^2} & 0 & 0 & 0 & 0 \\ -\frac{x}{r^2} \frac{z}{\sqrt{x^2 + y^2}} & -\frac{y}{r^2} \frac{z}{\sqrt{x^2 + y^2}} & \frac{\sqrt{x^2 + y^2}}{r^2} & 0 & 0 & 0 \\ \frac{y(\dot{x}y - \dot{y}x) + z(\dot{x}z - \dot{z}x)}{r^3} & \frac{x(\dot{y}x - \dot{x}y) + z(\dot{y}z - \dot{z}y)}{r^3} & \frac{x(\dot{z}x - \dot{x}z) + y(\dot{z}y - \dot{y}z)}{r^3} & \frac{x}{r} & \frac{y}{r} & \frac{z}{r} \end{bmatrix};\end{aligned}$$

$$\begin{aligned}J_H(s_{3DA}) &= \\ &= \begin{bmatrix} \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{y}{x^2 + y^2} & -\frac{x}{x^2 + y^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{x}{r^2} \frac{z}{\sqrt{x^2 + y^2}} & -\frac{y}{r^2} \frac{z}{\sqrt{x^2 + y^2}} & \frac{\sqrt{x^2 + y^2}}{r^2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{y(\dot{x}y - \dot{y}x) + z(\dot{x}z - \dot{z}x)}{r^3} & \frac{x(\dot{y}x - \dot{x}y) + z(\dot{y}z - \dot{z}y)}{r^3} & \frac{x(\dot{z}x - \dot{x}z) + y(\dot{z}y - \dot{y}z)}{r^3} & \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 \end{bmatrix};\end{aligned}$$

where $r = \sqrt{x^2 + y^2 + z^2}$.

7.3. Orthogonal projection of an Ellipsoid E onto a given line

We consider a given line \mathcal{L} , parameterized by a scalar s , defined by

$$\mathcal{L} \stackrel{\text{def}}{=} \{x | x = x_0 + s\mathbf{v}\},$$

where x_0 is a given point and \mathbf{v} is a given non-zero vector, see Figure 9.

Given any point x in the space, based on orthogonality of $(x - x_0) - s\mathbf{v}$ and \mathbf{v} , its orthogonal projection s onto \mathcal{L} corresponds to

$$s = \frac{\mathbf{v}^T(x - x_0)}{\mathbf{v}^T\mathbf{v}}$$

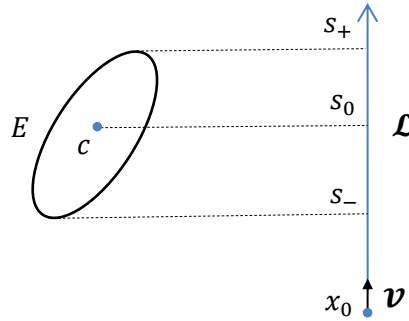


Figure 9. The orthogonal projection of the ellipsoid E onto the line \mathcal{L} is the interval $[S_-, S_+]$,

Now, the ellipsoid E is given by

$$E = \{x | x = c + L^{-T}y, \quad \|y\| < 1\}$$

Thus the projection of points in E correspond to values of s

$$s = \frac{\mathbf{v}^T(c + L^{-T}y - x_0)}{\mathbf{v}^T\mathbf{v}} = s_0 + w^T y, \quad \|y\| < 1$$

where

$$s_0 = \frac{\mathbf{v}^T(c - x_0)}{\mathbf{v}^T\mathbf{v}}$$

$$w = \frac{L^{-1}v}{v^T v}$$

Given the condition $\|y\| < 1$, it is evident that the orthogonal projection of E onto \mathcal{L} corresponds to the interval $[s_-, s_+]$ with

$$s_{\pm} = s_0 \pm |w|$$

And the projection length is

$$s_{\pm} = s_+ - s_- = 2|w|$$

The projections onto orthogonal axes

$$S_{x,y,z} = 2|L^{-1}v_{x,y,z}|$$