FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Multiple target tracking with PHD filters
Student:	Petr Jechumtál
Supervisor:	doc. Ing. Kamil Dedecius, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

Abstract: While single target tracking is a relatively well established discipline relying mostly on Kalman and particle filters, simultaneous tracking of multiple targets is still very demanding. It involves the aforementioned filters too, but faces the uncertainty about the measurement-target coupling. The targets may more or less randomly appear and disappear, which calls for random initialization and removal of the low-level filters. Moreover, if the measuring principle involves active radars or similar technologies, the measurements are subject to clutter.
One way of solving the task consists in the probability hypothesis density (PHD) filters. The approach involves modelling the respective collections of targets and measurements as random finite sets. The probability hypothesis density (PHD) recursions are employed to propagate the posterior intensity, which is a first-order statistic of the random finite set of targets, in time.

Goal: The bachelor thesis aims at a design and experimental validation of the Gaussian PHD filter. The steps are as follows:
1. Study the basic principles of single target tracking with Kalman filters.
2. Study the theory of the PHD filters. That is, get familiar with random finite sets, intensity functions, Gaussian mixture-based representation of targets, mixtures pruning and merging.
3. Design and implement a multiple target tracking algorithm(s) in python.
4. Experimentally validate the implementation on simulated data. Discuss results and properties of the implementation.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 13 January 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Difficulty: Very high (advanced mathematics - random finite sets)

References:

[1] B.-N. Vo and W.-K. Ma, "The Gaussian Mixture Probability Hypothesis Density Filter," in IEEE Transactions on Signal Processing, vol. 54, no. 11, pp. 4091-4104, Nov. 2006, doi: 10.1109/TSP.2006.881190.

[2] D. Smith and S. Singh, "Approaches to Multisensor Data Fusion in Target Tracking: A Survey," in IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 12, pp. 1696-1710, Dec. 2006, doi: 10.1109/TKDE.2006.183.

[3] E. Brekke, Fundamentals of Sensor Fusion. Target Tracking, Navigation and SLAM. NTNU, 2020.

[4] Á. F. García-Fernández, J. L. Williams, K. Granström and L. Svensson, "Poisson Multi-Bernoulli Mixture Filter: Direct Derivation and Implementation," in IEEE Transactions on Aerospace and Electronic Systems, vol. 54, no. 4, pp. 1883-1901, Aug. 2018, doi: 10.1109/TAES.2018.2805153.

Bachelor's thesis

MULTIPLE TARGET TRACKING WITH PHD FILTERS

Petr Jechumtál

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: doc. Ing. Kamil Dedecius, Ph.D.
May 3, 2023

Contents

List of Figures

List of code listings

List of Tables

I want to thank my supervisor, doc. Ing. Kamil Dedecius, Ph.D., for his patience and willingness to always advise me. I would also like to thank my friends and family, who have supported me throughout my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 3, 2023 .....

Abstract

This thesis focuses on the problem of tracking multiple targets, which is still a very challenging discipline. It deals with the uncertainty about the measurement-target association in a noisy-cluttered-environment. Furthermore, targets can appear and disappear more or less randomly. The solution described in this thesis is based on the Bayesian inference and consists of the probability hypothesis density (PHD) filters. This approach involves modeling the relevant sets of targets and measurements as random finite sets. The probability density recursions (PHDs) are used to propagate the posterior intensity, which is a first-order statistic of a random finite set of targets, over time.

Keywords Kalman filter, Bayesian inference, random finite set, multiple target tracking, state estimation, Gaussian mixture, intensity

Abstrakt

Tato práce se zaměřuje na problém sledování více cílů, což je stále velmi náročná disciplína. Zabývá se nejistotou spojení mezi měřením a cílem v prostředí se zvýšeným šumem. Cíle se navíc mohou objevovat a mizet víceméně náhodně. Řešení popsané v této práci je založeno na bayesovské inferenci a sestává z filtrů hustoty pravděpodobnostních hypotéz. Tento přístup zahrnuje modelování příslušných množin cílů a měření jako náhodné konečné množiny. K šíření aposteriorní intenzity, což je statistika prvního řádu náhodné konečné množiny cílů v čase, se používají rekurze hustoty pravděpodobnosti.

Klíčová slova Kalmánův filter, Bayesovská inference, konečná náhodná množiná, sledování více cílů, odhad stavů, Gaussovská směs, intenzita

Acronyms

CVM	Constant Velocity Model
JPDA	Joint Probabilistic Data Association
MAP	Maximum A Posteriori
MMSE	Minimum Mean Square Error
PDA	Probabilistic Data Association
PHD	Probability Hypothesis Density
PHMT	Probabilistic Multiple Hypothesis Tracking
RFS	Random Finite Set

Introduction

Target tracking is an essential part of systems that perform functions such as tracking, guidance, or obstacle avoidance. Tracking algorithms receive input measurements from sensors that provide radar, sonar, or video signals. Measurements are taken at regular intervals, and the task is to estimate the state of the target at each time instant. The state involves, for instance, the target position, speed, acceleration, and potentially other attributes. Successive estimates provide traces that describe the target trajectory.

An almost universally accepted mathematical framework used to describe this problem is the filtering theory, particularly Bayesian filtering. The probability distribution describing the knowledge of the target state is recursively predicted by propagation through the state evolution model describing the target motion and updated when new observations become available. The mean and covariance of the state are determined at each time step from the posterior distribution. The Kalman filter, derived in 1960, is the most commonly used filtering technique for linear and Gaussian target models.

The situation described above considers one target and its measurements. However, there are additional issues in real scenarios. First, the presence of clutter (false measurements), then we do not know which measurements are from the target. Second, the case of multiple targets, where the association between the target and its measurement is uncertain. The aim is then to estimate the location of an unknown number of targets based on observations of targets corrupted by noise, with the possibility that false detections may occur and that the observations may be false alarms caused by clutter.

Structure of the thesis

This thesis is divided into 5 main chapters. The aim of Chapter 1 is to explain the important concepts and ideas on which we build later in this thesis. In Chapter 2, we come to the problem of target tracking by deriving a Kalman filter. Then, in Chapter 3, we extend the tracking to multiple targets and illustrate everything with the example of probability hypothesis density (PHD) filters. Chapter 4 describes our implementation of the linear Gaussian PHD filter, over which we run simulations and experimentally verify its performance in Chapter 5.

Chapter 1

Bayesian modeling

In the first chapter, we introduce you to the basic concepts of the Bayesian modeling, which are important for understanding the whole work. We first introduce the Bayes formula and then proceed through the various topics needed to derive the Kalman filter. Throughout the paper, we consider that, in line with the Bayesian literature, we do not distinguish between random variables and their realizations. This chapter is mostly inspired by [1] and [2].

1.1 Bayes Theorem

The Bayes theorem is one of the fundamental concepts in statistics. It is named after English statistician Thomas Bayes, who proposed the principle in 1763. The Bayes theorem is used to determine the probability of a hypothesis in the presence of more evidence or information. The exact formulation is as follows:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}, \tag{1.1}$$

where A and B are events and $p(B) \neq 0$. $p(A|B)$ is the conditional probability of event A occurring given that B is true, $p(A)$ and $p(B)$ is probability of event A and B, respectively [1].

However, for the purposes of this paper, the following formulation is more appropriate. Assume that we have a random variable x, its distribution f depends on a parameter θ that can be multi-dimensional. Furthermore, $f(x|\theta)$ and $\pi(\theta)$ are their probability density functions. Then the Bayes formula is following

$$\pi(\theta|x) = \frac{f(x|\theta)\pi(\theta)}{f(x)}. \tag{1.2}$$

The terms in the last formula are:

- $\pi(\theta|x)$ is the posterior density of θ.
- $\pi(\theta)$ is the prior density of θ.
- $f(x|\theta)$ is the likelihood of observations.
- $f(x)$ is marginal density of observations.

Since $f(x)$ is only a normalization factor, we can rewrite Equation 1.2 into the proportional form as follows:

$$\pi(\theta|x) \propto f(x|\theta)\pi(\theta). \tag{1.3}$$

1.2 Prior and Posterior distributions

The probability distribution that expresses the (subjective) beliefs about the parameters is called the prior probability distribution, often simply called the prior. We can then update our prior distribution with the data using the Bayes theorem to obtain a posterior distribution. The posterior distribution is a probability distribution that represents the updated beliefs about the parameters after having seen the data [2]. However, the exact computation of the posterior distribution is difficult and often unnecessary. A key tool for the Bayesian statistics that greatly simplifies the computation is the concept of conjugate prior distributions.

1.2.1 Conjugate prior distribution

A prior distribution $\pi(\theta)$ is said to be a conjugate prior distribution if the posterior distribution $\pi(\theta|x)$ remains in the same distribution family as the prior.

Unfortunately, this behavior is not common. In order to ensure that our prior is conjugate, we need another level of abstraction, namely the exponential family of distributions, that can be defined in the following way [3].

▶ **Definition 1.1** (Exponential family)**.** *A family $\{F_\theta\}$ of distributions of a random variable x parameterized by a scalar or multivariate parameter θ is said to form an exponential family if the probability density function can be written in the form [4]*

$$f(x|\theta) = h(x)g(\theta)\exp\{\eta(\theta)^\intercal T(x)\}, \tag{1.4}$$

where $\eta(\theta)$ is natural parameter, $T(x)$ is the sufficient statistic, $g(\theta)$ is the normalizing function, and $h(x)$ is the base measure.

We can now fully formulate the definition of a conjugate prior [5],

▶ **Definition 1.2** (Conjugate prior)**.** *Suppose we have $f(x|\theta)$, which is the exponential family distribution defined in Definition 1.1. We say that a prior distribution $\pi(\theta)$ with hyper-parameters ξ and ν is conjugate to it if its probability density has the form,*

$$\pi(\theta) = q(\xi,\nu)g(\theta)^\nu \exp\{\eta(\theta)^\intercal \xi\}, \tag{1.5}$$

where ξ has the same dimension as $T(x)$, $\nu \in \mathbb{R}^+$ and $q(\xi,\nu)$ is a known function. The function $g(\theta)$ is the same as the normalization function in the density for $f(x|\theta)$.

From [6], we know that for a Bayesian update then, the density product from an exponential class with conjugate prior reduces to a simple update of hyperparameters of the form,

$$\begin{aligned} \xi_t &= \xi_{t-1} + T(x_t), \\ \nu_t &= \nu_{t-1} + 1. \end{aligned} \tag{1.6}$$

Table 1.1 lists some popular distribution and their conjugate prior distribution.

Distribution of observations	Inferred parameter	Conjugate prior	Posterior
Bernoulli	probability p	Beta	Beta
Binomial	probability p	Beta	Beta
Normal with known variance σ^2	mean μ	Normal	Normal
Poisson	rate λ	Gamma	Gamma
Exponential	rate λ	Gamma	Gamma

Table 1.1 Distributions and their conjugate priors. Standard notation of inferred parameters is assumed.

1.2.2 Example

In this example, we show how to estimate the probability p of the Bernoulli formula using the conjugate Beta distribution. Its purpose is to demonstrate the use of the Bayes theorem and conjugate priors as shown in Sections 1.1 and 1.2, respectively.

Assume we have a binary random variable X_t describing the result of 100 coin tosses, denoted x_t where $t = 1, 2, \ldots$ We can model this experiment using the Bernoulli distribution with a parameter p describing the (unknown) probability of head,

$$X_t \sim Ber(p), \quad p \in (0, 1), \tag{1.7}$$

where p is the probability of success, which we try to estimate. X_t can only take values from the set $\{0, 1\}$ (tail or head).

If we rewrite the probability density function model into exponential according to Definition 1.1, we obtain

$$\begin{aligned} f(x_t|\theta) &= f(x_t|p) \\ &= p^{x_t}(1-p)^{1-x_t} \\ &= \exp\{\ln[p^{x_t} \cdot (1-p)^{1-x_t}]\} \\ &= \exp\{x_t \ln p + (1-x_t)\ln(1-p)\} \\ &= \underbrace{1}_{h(x_t)} \cdot \underbrace{1}_{g(p)} \cdot \exp\left\{ \underbrace{\begin{bmatrix} \ln p \\ \ln(1-p) \end{bmatrix}^{\intercal}}_{\eta(p)} \underbrace{\begin{bmatrix} x_t \\ 1-x_t \end{bmatrix}}_{T(x_t)} \right\}. \end{aligned} \tag{1.8}$$

As mentioned in the Section 1.2 appropriate conjugate prior for p is a Beta distribution. Its probability density has the form

$$\pi(p|a, b) = \frac{1}{B(a, b)} p^{a-1}(1-p)^{b-1}, \qquad a, b > 0, \tag{1.9}$$

where

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \tag{1.10}$$

is the Beta function defined by the proportion of Gamma functions, and a, b are hyperparameters.

Now let us rewrite this model into a more appropriate form that reflects our knowledge gained in the previous time steps $t-1$ so that we can incorporate this knowledge into the current state t,

$$\pi(p|a_{t-1}, b_{t-1}) = \frac{1}{B(a_{t-1}, b_{t-1})} p^{a_{t-1}-1}(1-p)^{b_{t-1}-1}. \tag{1.11}$$

In the next step, we rewrite the model in exponential form, which can be done in the following way,

$$\begin{aligned} \pi(p|\xi_{t-1}, \nu_{t-1}) &= \pi(p|a_{t-1}, b_{t-1}) \\ &= \frac{1}{B(a_{t-1}, b_{t-1})} p^{a_{t-1}-1}(1-p)^{b_{t-1}-1} \\ &= \underbrace{\frac{1}{B(a_{t-1}, b_{t-1})}}_{q(\xi_{t-1}, \nu_{t-1})} \cdot \underbrace{1}_{g(p)^{\nu_{t-1}}} \exp\left\{ \underbrace{\begin{bmatrix} \ln p \\ \ln(1-p) \end{bmatrix}^{\intercal}}_{\eta(p)} \underbrace{\begin{bmatrix} a_{t-1}-1 \\ b_{t-1}-1 \end{bmatrix}}_{\xi_{t-1}} \right\}. \end{aligned} \tag{1.12}$$

We know from Section 1.2.1 that we can simplify the Bayesian update for the conjugate prior to the form,

$$\begin{aligned} \xi_t &= \xi_{t-1} + T(x_t), \\ \nu_t &= \nu_{t-1} + 1. \end{aligned} \tag{1.13}$$

Since $g(\nu) = 1$, we can completely omit the second equation from our calculations. The posterior hyperparameter ξ_t are equivalent to

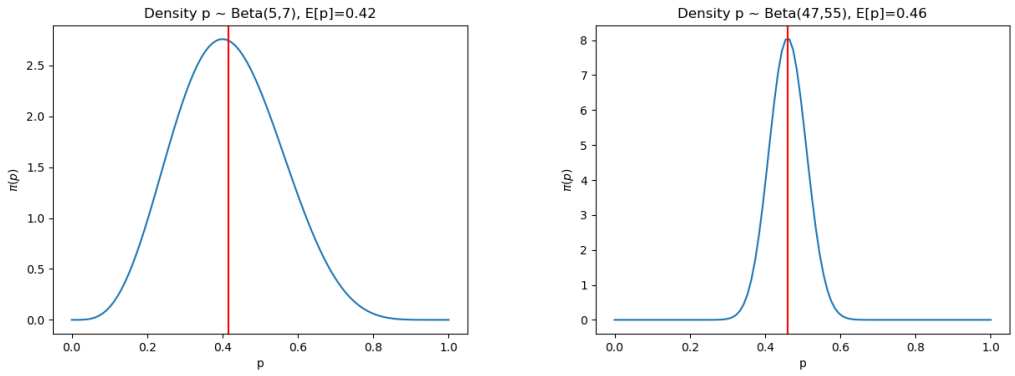
$$\begin{bmatrix} a_t - 1 \\ b_t - 1 \end{bmatrix} = \begin{bmatrix} a_{t-1} - 1 \\ b_{t-1} - 1 \end{bmatrix} + \begin{bmatrix} x_t \\ 1 - x_t \end{bmatrix}. \tag{1.14}$$

It follows that to update our hyperparameters a, b, and we only need to add $a + 1$ if the head fell or $b + 1$ if the tail fell. For the estimation of parameter p, we use the mean of the beta distribution, which is as follows:

$$\mathbb{E}[p] = \mathbb{E}[p|a, b] = \frac{a}{a + b}. \tag{1.15}$$

The last essential thing for our experiment is to choose the correct initial values of the hyperparameters a, b. Since we do not know anything about the coin (whether it is fair or not), $a = b = 1$ seem to be an appropriate value to model exactly this information. In Figure 1.1, we can see the results of our experiment, where the mean of the Beta distribution determines our estimate of the parameter p. For the demonstration, I chose the results after 10 and 100 coin flips.

The results show nicely that our uncertainty around the estimate of p decreases with increasing flips. Moreover, we can hypothesize that our coin is fair because p is close to the value of 0.5.



(a) 10 coins flip

(b) 100 coins flip

Figure 1.1 Beta distribution charts of coin flips

1.3 State-space model

Until now, we have considered models with fixed parameters. However, if our (hidden) parameters change and we know the pattern of their evolution, then we get a state-space model. It is a complete overview of the state of the system at a certain point in time. Knowing the state at some initial time t_0 and knowing the system inputs after time t_0 allows us to determine the state at a later time t_1. Thus, the state at time t_0 represents the complete history of the system

behavior before time t_0, insofar as this history affects future behavior. The state-space model classically becomes two equations, one describing the hidden process and the other for the data model.

At any fixed time, the state of a system can be described by vector x_t. For example, one of the state variables of a tracking system is the position of the tracked object.

Let us assume that we have a state model that is time-invariant and that we move in discrete times $t = 0, 1, 2, \ldots$ The state-space model consists of two models: first, the hidden process model, and second, the measurement model. The target states use the hidden process model for their evolution. At the same time, states generate measurements according to the measurement model. The state evolution is then captured in Figure 1.2. We look at the models in detail in the following sections.

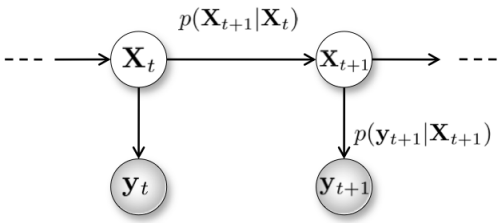


Figure 1.2 Evolution of the target state and its contribution to the origin of the measurement [7].

1.3.1 Hidden process model

According to [8], we can define such a model using the following equations,

$$x_t = Ax_{t-1} + Bu_t + w_t, \tag{1.16}$$

where x_t and x_{t-1} are the (unknown and estimated) states vectors at time instants t and $t-1$, respectively. u_t is a control vector that is known, w_t is the state noise. Finally, A and B are known matrices of compatible dimensions. We also assume that w_t is an independent zero-centered normally distributed noise,

$$w_t \sim \mathcal{N}(0, Q). \tag{1.17}$$

Since w_t is a normal random variable and Formula (1.16) represents its transformation, the state variable x_t is a random variable too. The transformation is linear and preserves the distribution. That is

$$x_t \sim \mathcal{N}(Ax_{t-1} + Bu_t, Q). \tag{1.18}$$

In the Bayesian estimation theory, we represent this distribution by a probability density function $p(x_t|x_{t-1}, u_t)$.

1.3.2 Measurement model

The hidden process model describes the state variables and their evolution, but the variables themselves are not observable. The states x_t generate observable measurements y_t through a measurement model prescribed by the formula

$$y_t = Hx_t + \varepsilon_t, \tag{1.19}$$

where x_t is a state vector, H is matrix of compatible shapes.

Further, similarly to the state model, we assume ε_t is independent and from a normal distribution centered at zero

$$\varepsilon_t \sim \mathcal{N}(0, R). \tag{1.20}$$

Similar to w_t, ε_t is also a normal random variable, and Formula (1.19) represents its transformation, the measurement of y_t is also a random variable. The transformation is linear and preserves the distribution. This means

$$y_t \sim \mathcal{N}(Hx_t, R). \tag{1.21}$$

In the Bayesian estimation theory, we represent this distribution by a probability density function $f(y_t|x_t)$. The fact that both noise variables w_t and ε_t are centered at zero is extremely important because otherwise, we would get a systematic error in the estimation.

1.3.3 Example: Constant velocity model

One of the most widely used state-space model is the constant velocity model (CVM). In its basic version, it describes the position of the target on a 2D surface (direct extension to higher dimensions is straightforward). The position has two components $x_{1,t}$, $x_{2,t}$, e.g., corresponding to longitude and latitude, respectively. We model them based on the previous position at time $t-1$ and the velocities in these components $v_{1,t}$, $v_{2,t}$ during the time period Δt. All CVM state variables can be seen in 1.3. We also model the velocities, for simplicity, as a random walk. The state evolution formulas are:

$$\begin{aligned} x_{1,t} &= x_{1,t-1} + v_{x_1,t}\Delta_t + w_{x_1,t}, \\ x_{2,t} &= x_{2,t-1} + v_{x_2,t}\Delta_t + w_{x_2,t}, \\ v_{x_1,t} &= v_{x_1,t-1} + w_{v_{x_1},t}, \\ v_{x_2,t} &= v_{x_2,t-1} + w_{v_{x_2},t}, \end{aligned} \tag{1.22}$$

in the matrix notation:

$$\underbrace{\begin{bmatrix} x_{1,t} \\ x_{2,t} \\ v_{x_1,t} \\ v_{x_2,t} \end{bmatrix}}_{x_t} = \underbrace{\begin{bmatrix} 1 & 0 & \Delta_t & 0 \\ 0 & 1 & 0 & \Delta_t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{A_t} \underbrace{\begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \\ v_{x_1,t-1} \\ v_{x_2,t-1} \end{bmatrix}}_{x_{t-1}} + \underbrace{\begin{bmatrix} w_{x_1,t} \\ w_{x_2,t} \\ w_{v_{x_1},t} \\ w_{v_{x_2},t} \end{bmatrix}}_{w_t}. \tag{1.23}$$

Assume that there are noisy measurements in the same coordinate space. Then, the measurements are $y_{1,t}$ and $y_{2,t}$. Their model is

$$\begin{aligned} y_{1,t} &= x_{1,t} + \varepsilon_{x_1,t}, \\ y_{2,t} &= x_{2,t} + \varepsilon_{x_2,t}, \end{aligned} \tag{1.24}$$

in the matrix notation:

$$\underbrace{\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix}}_{y_t} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}_{H_t} \underbrace{\begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix}}_{x_t} + \underbrace{\begin{bmatrix} \varepsilon_{x_1,t} \\ \varepsilon_{x_2,t} \end{bmatrix}}_{\varepsilon_t}. \tag{1.25}$$

Since the noise variables w_t and ε_t are normally distributed, we may rewrite Formulas (1.22) and (1.24) as

$$\begin{aligned} x_t &\sim \mathcal{N}(Ax_{t-1} + Bu_t, Q), \\ y_t &\sim \mathcal{N}(Hx_t, R). \end{aligned} \tag{1.26}$$

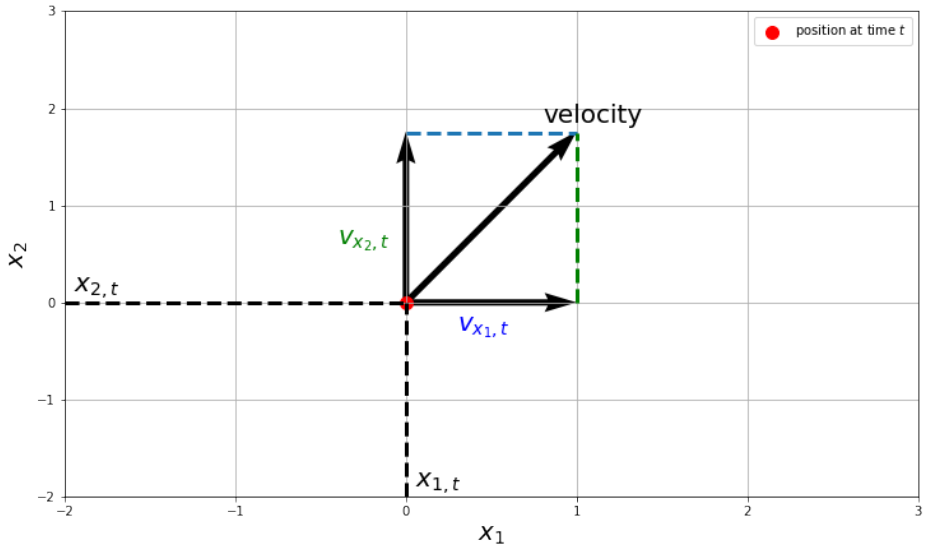


Figure 1.3 Graphical representation of CVM state variables in 2D.

Chapter 2

Kalman filter: Single target tracking

The Kalman Filter is one of the most important and common sequential estimation algorithms. It produces estimates of hidden variables based on inaccurate and uncertain measurements. Also, predicts the future system state based on past estimates. The filter is named after Rudolf E. Kálmán [9]. Navigation for the Apollo program is among the first applications of the Kalman filter. Since then, it has spread to many other areas, such as target tracking, navigation systems, control systems, computer graphics, and much more.

The Kalman filter itself is limited to linear models. We need an extended Kalman filter if we are working with non-linear models. But that is beyond the scope of our work. We work with linear models; therefore, the classical Kalman filter is sufficient. This chapter is mostly inspired by [6] and [10].

2.1 Derivation of the Kalman filter

From Section 1.3, we have the following state model

$$\begin{aligned} x_t &= Ax_{t-1} + Bu_t + w_t, & w_t &\sim \mathcal{N}(0, Q), \\ y_t &= Hx_t + \varepsilon_t, & \varepsilon_t &\sim \mathcal{N}(0, R), \end{aligned} \tag{2.1}$$

from normality, we can therefore deduce

$$\begin{aligned} x_t &\sim \mathcal{N}(Ax_{t-1} + Bu_t, Q), & &\text{with the density } p(x_t|x_{t-1}, u_t), \\ y_t &\sim \mathcal{N}(Hx_t, R), & &\text{with the density } f(y_t|x_t). \end{aligned} \tag{2.2}$$

We intend to filter sequentially estimate the state x_t in a Bayesian fashion, and we need a prior distribution for x_t. Since the model y_t is a normal distribution, the conjugate prior is a normal distribution, too; see Section 1.2. Let us denote it

$$\pi(x_t|y_{0:t-1}, u_{0:t-1}) = \mathcal{N}(x_{t-1}^+, P_{t-1}^+). \tag{2.3}$$

where x_{t-1}^+ is the mean vector and P_{t-1}^+ is the covariance matrix.

The Kalman filter, like many other filters, runs in two steps: the state prediction and the state update. The prediction uses the first of the state equations, and the update uses the second. It is important to note that if the states of x_t and the measurements of y_t were not noisy, we would not actually need the update since there would be a direct relationship between

the predicted x_t and the measured y_t. It would be enough just to use the second equation and use algebraic adjustments to get to x_t. However, since noise is present in the measurements, we use the predicted state as prior information in the Bayes theorem, where we use the measurements to correct it. Let us now focus on both steps in detail.

2.1.1 Prediction step

The prediction uses the state evolution model (the first equation in the state-space model). The state estimation x_{t-1} from the previous time instant enters the equation. We then calculate (predict) the current state x_t by computing the equation. We combine the prior distribution with the evolution model to obtain the posterior distribution [11],

$$\begin{aligned}\pi(x_t|y_{0:t-1},u_{0:t}) &= \int p(x_t|x_{t-1},u_t)\,\pi(x_{t-1}|y_{0:t-1},u_{0:t-1})\,\mathrm{d}x_{t-1}\\ &= \int \pi(x_t,x_{t-1}|y_{0:t-1},u_{0:t})\,\mathrm{d}x_{t-1},\end{aligned} \tag{2.4}$$

where the integral is taken over the space of x_{t-1}. Since we multiply two normal distributions, we get a multivariate normal distribution. Subsequently, we integrate over x_{t-1}, which gets rid of the previous x_{t-1} and leaves us again with a normal distribution $\mathcal{N}(x_t^-, P_t^-)$ with hyperparameters equal to the following expressions,

$$\begin{aligned}x_t^- &= Ax_{t-1}^+ + Bu_t,\\ P_t^- &= AP_{t-1}^+A^\intercal + Q.\end{aligned} \tag{2.5}$$

It is also worth noting that the covariance of the estimate of P_t^- expresses the measure of uncertainty in this estimate. Since we have only predicted and have not used any measurements to correct this prediction, it logically follows that our measure of uncertainty will increase.

2.1.2 Update step

The update step corrects the predicted estimate based on the observed data y_t. We use the Bayes formula in the following way:

$$\pi(x_t|y_{0:t},u_{0:t}) \propto f(y_t|x_t)\,\pi(x_t|y_{0:t-1},u_{0:t}). \tag{2.6}$$

We know from Section 1.2.1 that the Bayesian update can be reduced to a sum of the hyperparameter and the sufficient statistic when the prior is conjugate. Let us, therefore, modify the distributions to exponential form. First, we adjust the measurement model,

$$\begin{aligned}f(y_t|x_t) &\propto \exp\left\{-\frac{1}{2}(y_t-Hx_t)^\intercal R^{-1}(y_t-Hx_t)\right\}\\ &= \exp\left\{\mathrm{Tr}\left(\underbrace{-\frac{1}{2}\begin{bmatrix}-1\\x_t\end{bmatrix}\begin{bmatrix}-1\\x_t\end{bmatrix}^\intercal}_{\eta}\underbrace{\begin{bmatrix}y_t^\intercal\\H^\intercal\end{bmatrix}R^{-1}\begin{bmatrix}y_t^\intercal\\H^\intercal\end{bmatrix}^\intercal}_{T(y_t)}\right)\right\}.\end{aligned} \tag{2.7}$$

Now we make the same adjustment for the prior distribution,

$$\begin{aligned}\pi(x_t|y_{0:t-1},u_{0:t}) &\propto \exp\left\{-\frac{1}{2}(x_t-x_t^-)^\intercal(P_t^-)^{-1}(x_t-x_t^-)\right\}\\ &= \exp\left\{\mathrm{Tr}\left(\underbrace{-\frac{1}{2}\begin{bmatrix}-1\\x_t\end{bmatrix}\begin{bmatrix}-1\\x_t\end{bmatrix}^\intercal}_{\eta}\underbrace{\begin{bmatrix}(x_t^-)^\intercal\\I\end{bmatrix}(P_t^-)^{-1}\begin{bmatrix}(x_t^-)^\intercal\\I\end{bmatrix}^\intercal}_{\xi_t}\right)\right\},\end{aligned}$$

where I is an identity matrix of matching dimensions. Now we can rewrite the Bayesian update using the following equations,

$$\begin{aligned}\xi_t &= \xi_{t-1} + T(y_t) \\ &= \begin{bmatrix} (x_t^-)^\intercal (P_t^-)^{-1} x_t^- + y_t^\intercal R^{-1} y_t, & (x_t^-)^\intercal (P_t^-)^{-1} + y_t^\intercal R^{-1} H \\ (P_t^-)^{-1}(x_t^-)^\intercal + H^\intercal R^{-1} y_t, & (P_t^-)^{-1} + H^\intercal R^{-1} H \end{bmatrix}.\end{aligned} \tag{2.8}$$

We can easily derive the posterior distribution hyperparameters

$$\begin{aligned} P_t^+ &= (\xi_{t;[2,2]})^{-1} \\ &= \left[(P_t^-)^{-1} + H^\intercal R^{-1} H\right]^{-1} \\ &= (I - K_t H) P_t^-, \\ x_t^+ &= (\xi_{t;[2,2]})^{-1} \xi_{t;[2,1]} \\ &= P_t^+ \left[(P_t^-)^{-1}(x_t^-)^\intercal + H^\intercal R^{-1} y_t\right] \\ &= x_t^- + P_t^+ H^\intercal R^{-1}(y_t - H x_t^-), \end{aligned} \tag{2.9}$$

where

$$K_t = P_t^- H^\intercal (R + H P_t^- H^\intercal)^{-1} \tag{2.10}$$

is the Kalman gain. In general, the greater the Kalman gain, the greater the emphasis of the new measurements. The filter is then more sensitive, but less able to filter out noise. The Formula (2.10) shows the gain that yields the optimal Kalman filter. Alternatively, it is possible to proceed with fixed gain or other variants [1]. In Formula (2.11), we can see how the hyperparameters change during one cycle of the Kalman filter.

$$\rightarrow (P_{t-1}^+, x_{t-1}^+) \xrightarrow{\text{prediction}} (P_t^-, x_t^-) \xrightarrow{\text{update}} (P_t^+, x_t^+) \rightarrow \tag{2.11}$$

2.1.3 Example

The main purpose of this example is to show the functionality of the Kalman filter. We demonstrate this by using the example of tracking the position of a falling object from a height on the ground. Atmospheric drag is neglected. Such an example has the following state equations,

$$\begin{aligned} X_t &= \begin{bmatrix} 1 & -\Delta_t \\ 0 & 1 \end{bmatrix} X_{t-1} + \begin{bmatrix} -\frac{1}{2}\Delta_t^2 \\ \Delta_t \end{bmatrix} g + \begin{bmatrix} w_{h,t} \\ w_{v,t} \end{bmatrix}, \\ Y_t &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_t + \varepsilon_t. \end{aligned} \tag{2.12}$$

We get the measured height of the object from each time instant for 100 seconds and estimate the actual height of the object based on these measurements and the Kalman filter. In Figure 2.1, we can see the actual trajectory of the object and the measurements captured by the sensor.

Before we can use the Kalman filter as explained in Section 2.1, we have to initialize the matrices A, B, H, R, and Q, which can be done according to Formula (2.12). Next, we need to initialize our prior ideas about the task, all we know is that we will observe the falling object from a height, so we decided to place our prior idea of the object up to 60000 meters above the ground. However, since this is just our guess unsupported by any data, we set our covariance matrix P to reflect the large uncertainty. This will ensure that even if we are off by a lot with the initial height, the Kalman filter will be more responsive to the measurements and will go in the right direction quickly. This fact is clearly seen in Figure 2.2. Once we have all the variables initialized, we can run the Kalman filter. In the graph, we can see our gradually improving estimates of the object height, which have already caught the actual trajectory at around 35 seconds. This is also evident in the narrowing of the confidence interval, which shows the band in which our falling object is actually located with a probability of over 95%.

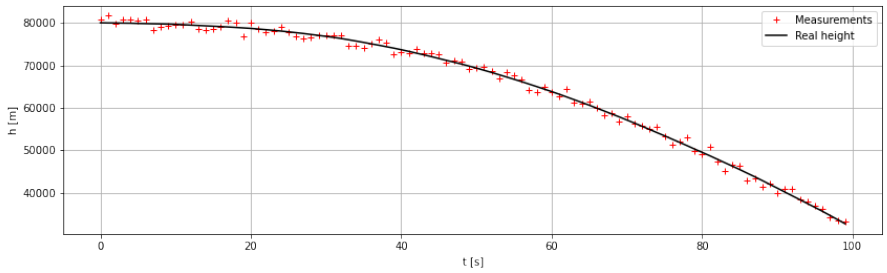


Figure 2.1 Real trajectory of an object dropped at 80000 meters above the ground and simulated for 50 time steps.

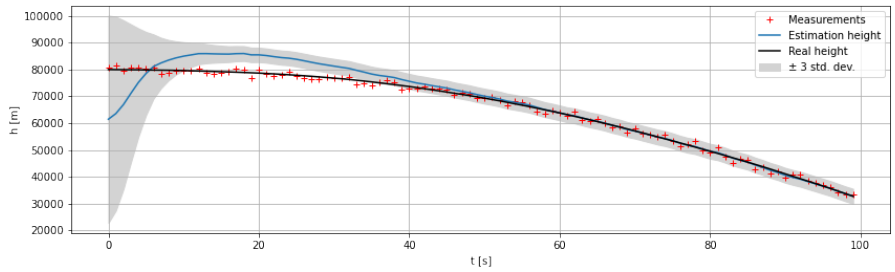


Figure 2.2 Object height estimation with plotted confidence interval.

2.2 Single-target tracking

In this section, we begin our study of target tracking. Unlike the Kalman filter, where we assumed that at every time instant we get exactly one measurement from our tracked object, in the real world of target tracking, it may happen that we get no measurements or that we get multiple measurements where some of them are false.

The standard assumption in the following part of the paper is that our tracked object generates at most one measurement. The other measurements are generated by other objects or by the clutter. This section is mostly inspired by [12].

2.2.1 Clutter

The measurements we enter into the tracking method are usually obtained from a sensor array, such as a radar image. For a given resolution cell (the smallest volume of airspace in which a radar cannot determine the presence of more than one target), we have a Boolean detector that is tasked with reporting whenever a target is in that resolution cell. Otherwise, it should report nothing. However, since no detector is perfect, there is some probability that it reports a false when no target is present. We call these false measurements clutter. In Figure 2.3 we can see the actual radar image.

Furthermore, for the purpose of our work, we assume that the detection of targets in different cells is independent and equally distributed, and the intensity of false measurements comes from a Poisson distribution.

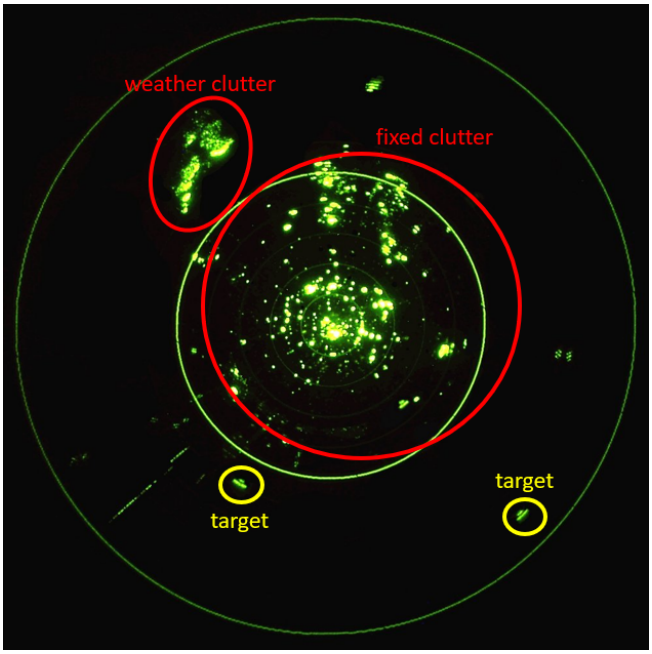


Figure 2.3 The described radar image [13].

2.2.2 Misdetections

The second complication that makes target tracking more difficult than pure filtering is that the target may or may not be detected. If we assume that the target detection or non-detection event at time t is independent of the detection events at all other time steps, then the detection model is very simple. It is given by a Bernoulli random variable:

$$P(\delta) = \begin{cases} P_D & \text{if } \delta = 1, \\ 1 - P_D & \text{if } \delta = 0, \end{cases} \tag{2.13}$$

where P_D is the probability of detection.

The main complication is to determine the correct value of P_D. Typically this value is in the range from 0.5 (e.g., tracking of boats) to 0.95 (e.g., radar tracking of airplanes)[14].

2.2.3 Data association

It is clear from Section 2.2.1 and Section 2.2.2 that the biggest problem in tracking a single target in space is proper measurement association. In the case of using a Kalman filter, we therefore need to select the correct measurement to update our filter with.

This data association problem can be solved in many ways. In general, we have two options that, in some sense, correspond to maximum a posteriori (MAP) and the minimum mean square error (MMSE) estimation: We can choose the most probable measurement and discard all other measurements, or we can attempt to calculate a weighted average over all the possible measurements, where the weights quantify how likely the measurement is to be the one true measurement [14]. In practice, the second approach is usually adopted. A particularly popular filter averaging over plausible measurements is the probabilistic data association (PDA) and its extension to multiple targets (JPDA) [15]. Unfortunately, this filter is beyond the scope of our thesis, and we do not discuss it in detail here.

Chapter 3

Multiple target tracking

In the previous chapter, we made two key assumptions. First, there is only one target, and second, this target generates all real observations. When observing multiple targets, we allow for the possibility of more than one or even no target. Thus, the sensor responses may be caused by any one of the targets or none of them. In the second case, we refer to these measurements as clutter, cf. Section 2.2.1. We must realize we do not know how many targets are moving in the space.

Another challenge for us is the appearance and disappearance of targets. The task of multiple target tracking has a significant complexity, increased by the unknown cardinality of the set of targets. For instance, assume a close-range airport radar. If an airplane takes off, a new target has to be created (birth). On the other hand, if an airplane land or leave radar range, the target has to disappear (death).

However, the main problem in multi-target tracking is deciding which target elicited which sensor response or whether the response is a false measurement. Data association is the process of associating a sensor response with a target. If the assignment is ambiguous, the multi-target tracking problem becomes complicated to solve [16].

This chapter primarily deals with the probability hypothesis density (PHD) filter method [17], other methods for tracking multiple targets such as the joint probability data association (JPDA) [15] filter or probabilistic multiple hypothesis tracking (PHMT) [18] are not be discussed further here.

3.1 Introduction to multiple target tracking

First, let us look at the problem of multiple target tracking without false measurements, and for simplicity, let us consider that we know how many targets are moving in space. Moreover, assume that we track each of these targets using a Kalman filter that is already initialized. Then, at each time point, all we need to do is to correctly assign a measurement to a target and then use this measurement to perform the update step of the corresponding Kalman filter.

There are several ways to look at such a problem. One possible solution is to advance the state at its next $t+1$ time point for each target. And then assign the measurement that is closest to this predicted state.

The problem arises when we have two targets close together. Then the assignment of measurements to the target may not be unambiguous. However, there is a solution where we assign each measurement a probability of $0 < p < 1$ and then try to maximize the product of probabilities over all measurements, thereby choosing the most likely assignment of measurements to targets. These considerations lead to the JPDA filter [15], which uses a similar system of assign-

ing measurements to targets and can even handle false measurements, making it a full-featured filter for tracking multiple targets.

From this brief introduction, it is clear that even the simplest example of tracking multiple targets can get very complicated, and we have not addressed the problem of target birth and death at all here. Let us move on to a PHD filter that can also handle these situations.

3.2 Derivation of Probability Hypothesis Density filter

In this section, we derive all the important concepts of the PHD filter. Most of this section is inspired by [16], [17] and [19].

3.2.1 Possible target states

There are three possibilities of what can happen to one target between two-time steps:

1. A target can be born, i.e., appear in our viewing territory. Modeling such a problem may vary in different applications of the PHD filter, but in our case, we consider that a target can only originate at a known location in advance. In the analogy of the aircraft example, these locations are airports. At each time instant, the PHD filter creates a low-weight target at these birth locations. Only if there are enough measurements in the vicinity can the target actually be born and continue to other time instants.

2. An existing target can continue on its trajectory or die. We model this case using $p_{S,t}(x_{t-1})$ as the probability of the target survival and $1 - p_{S,t}(x_{t-1})$ as the probability of the target death. This probability can be written as the conditional probability x_t on the previous state x_{t-1} as the following $f_{t|t-1}(x_t|x_{t-1})$.

3. A new target may be spawned. Thus, at time t, we have a single target, and at time $t+1$, we have multiple targets that arose from a single target x_t. This problem is also modeled in the PHD filter up to the task specification. In the analogy of the aircraft example, such a situation corresponds to an aircraft carrier from which an aircraft takes off (be spawned).

3.2.2 Random finite sets

To continue, we need to give a definition of the concept of a random finite set (RFS) [20],

▶ **Definition 3.1** (Random finite sets)**.** *A random variable whose possible outcomes are sets with a finite number of unique elements.*

We need this definition mainly for easier handling of measurements and target states.

Suppose $M(t)$ is the number of targets at time t and consider that at time $t-1$, the states of the targets are $x_{t-1,1}, \ldots, x_{t-1,M(t-1)} \in \mathcal{X}$, where $\mathcal{X}$ stands for set of all states. At the next time instant, some of these targets may disappear, surviving targets evolve into new states, and new targets may appear. The result in $M(t)$ new states $x_{t,1}, \ldots, x_{t,M(t)} \in \mathcal{X}$. Note that the order of in which the states are listed have no meaning in the multi-target RFS model formulation.

Similarly, for the measurements, let us assume $N(t)$ measurements $z_{t,1}, \ldots, z_{t,N(t)} \in \mathcal{Z}$ are received at time t, where $\mathcal{Z}$ stands for set of all measurements. The origins of the measurements are not known, and therefore the order in which the measurements were taken is not known, which implies that the order has no significance. Only some of these measurements are actually generated by the targets. Moreover, they are indistinguishable from false measurements. As mentioned, the main goal of tracking multiple targets is to jointly estimate the number of targets and their states from measurements with uncertain origins.

Let X_t and Z_t denote the RFS of the multi-target states and the measurements at time step t, respectively,

$$\begin{aligned} X_t &= \{x_{t,1}, \ldots, x_{t,M(t)}\} \in \mathcal{F}(\mathcal{X}), \\ Z_t &= \{z_{t,1}, \ldots, z_{t,N(t)}\} \in \mathcal{F}(\mathcal{Z}), \end{aligned} \tag{3.1}$$

where $\mathcal{F}(\mathcal{X})$ and $\mathcal{F}(\mathcal{Z})$ are the respective collections of all finite subsets of $\mathcal{X}$ and $\mathcal{Z}$.

The key in the random finite set formulation is to treat the target set X_t, and measurement set Z_t as the multiple-target state and multiple-target observation, respectively. The multiple-target tracking problem can then be posed as a filtering problem.

Now let us list what the state model consists of. The X_t is formed by merging three RFS, namely the set of surviving targets, the set of spawn targets, and finally, the set of newborn targets.

The set of surviving targets is formed by computing, for each state $x_{t-1} \in X_{t-1}$ at time $t-1$, an RFS $S_{t|t-1}(x_{t-1})$ such that it contains either $\{x_t\}$ when the target survives or $\{\emptyset\}$ when the target dies. The set of spawn targets $B_{t|t-1}(x_{t-1})$ arises analogously. The set of newborn targets Γ_t is created by adding targets to all birthplaces as described in Section 3.2.1.

Thus, in total, X_t can be written in the following way,

$$X_t = \left[\bigcup_{\zeta \in X_{t-1}} S_{t|t-1}(\zeta)\right] \cup \left[\bigcup_{\zeta \in X_{t-1}} B_{t|t-1}(\zeta)\right] \cup \Gamma_t. \tag{3.2}$$

Similarly to the single-target tracking in Section 2.2.2, multiple-target tracking requires working with the probability of target detection. For a target $x_t \in X_t$, $p_{D,t}(x_t)$ is the probability of detecting the target, and $1 - p_{D,t}(x_t)$ is the probability of missing the target.

At time t, each state $x_t \in X_t$ generates an RFS $\Theta_t(x_t)$, that can take on either $\{z_t\}$ when target is detected or $\emptyset$ when target is not detected.

Furthermore, we need to work with clutter measurements similarly to Section 2.2.1. Thus, at each time instant, we obtain a set of K_t of false measurements. Altogether, the set Z_t is the union of the false measurements and the received detection from the targets

$$Z_t = K_t \cup \left[\bigcup_{x \in X_t} \Theta_t(x)\right]. \tag{3.3}$$

3.2.3 Multiple-target Bayes filter

Let us now rewrite the principles of filtering from Section 1.1 for the RFS variables. The optimal multi-target Bayes filter propagates the multi-target posterior density $\pi_t(.|Z_{1:t})$ conditioned on the sets of observations up to time t, with the following recursion,

$$\begin{aligned} \pi_{t|t-1}(X_t|Z_{1:t-1}) &= \int f_{t|t-1}(X_t|X)\pi_{t-1}(X|Z_{1:t-1})\mu_s(dX), && \text{(prediction)} \\ \pi_t(X_t|Z_{1:t}) &= \frac{g_t(Z_t|X_t)\pi_{t|t-1}(X|Z_{1:t-1})}{\int g_t(Z_t|X_t)\pi_{t|t-1}(X|Z_{1:t-1})\mu_s(dX)}, && \text{(update)} \end{aligned} \tag{3.4}$$

where the motion model is governed by the transition density $f_{t|t-1}(X_t|X)$ and multi-target likelihood $g_t(Z_t|X_t)$ and μ_s is an appropriate reference measure on $\mathcal{F}(\mathcal{X})$.

The function $g_t(Z_t|X_t)$ is the joint multi-target likelihood function, or global density, of observing the set of measurements Z given the set of target states X, which is the total probability density of association between measurements in Z and parameters in X.

3.3 The Probability Hypothesis Density (PHD) filter

The computational complexity of the joint multi-target likelihood grows exponentially with the number of targets and thus becomes numerically intractable. The PHD filter was derived to provide a sub-optimal strategy for determining the set of target states at each iteration using the first-order statistical moment of the posterior distribution of multiple targets [21].

3.3.1 Intensity

For an RFS X on $\mathcal{X}$ with probability distribution P, its first-order moment is a non-negative function v on $\mathcal{X}$, called the intensity, such that for each region $S \subseteq \mathcal{X}$,

$$\int |X \cap S| P(dX) = \int_S v(x)dx, \tag{3.5}$$

where $||$ denotes the cardinality of the set. Thus, the integral v over the region S gives the expected number of targets X that are in the region S. The integral over the v then indicates the number of targets in X.

The local maxima of intensity v are the points in $\mathcal{X}$ with the highest local concentration of the expected number of targets and can therefore be used to generate estimates of the targets of X. One possible way to estimate the number of targets in X is to choose among the highest intensity peaks, those that exceed a predetermined threshold.

For the PHD filters, the RFS of X must have a Poisson distribution. This means the cardinality distribution of X is from a Poisson distribution with mean $\hat{N}$, and the elements of x from X are independently identically distributed. Their intensities can fully characterize these RFS. Therefore, the intensities can then be used as prior and posterior distributions.

3.3.2 PHD recursion

Before we move on to defining the recursion, we need to state the assumptions:

1. Each target evolves and generates observations independently of one another.
2. Clutter is Poisson-distributed and independent of target-originated measurements.
3. The predicted multiple-target RFS is Poisson-distributed.

Now we can define the recursion using the prediction and update steps. It is worth noting that the intensities $v_t(x)$ and $v_{t|t-1}(x)$ are used here as conjugate priors

$$\begin{aligned} v_{t|t-1}(x) &= \int p_{S,t}(\zeta) f_{t|t-1}(x|\zeta) v_{t-1}(\zeta) d\zeta + \int \beta_{t|t-1}(x|\zeta) v_{t-1}(\zeta) d\zeta + \gamma_t(x), \\ v_t(x) &= [1 - p_{D,t}(x)] v_{t|t-1}(x) + \sum_{z \in Z_t} \frac{p_{D,t}(x) g_t(z|x) v_{t|t-1}(x)}{\kappa_t(z) + \int p_{D,t}(\xi) g_t(z|\xi) v_{t|t-1}(\xi) d\xi}, \end{aligned} \tag{3.6}$$

- $\gamma_t(.)$ is intensity of the birth RFS Γ_t at time t.
- $\beta_{t|t-1}(.|\zeta)$ is intensity of the RFS $B_{t|t-1}(\zeta)$ spawned at time t by a target with previous state ζ.
- $p_{S,t}(\zeta)$ probability that a target still exists at time t given that its previous state is ζ.
- $p_{D,t}(x)$ probability of detection given a state x at time t.
- $\kappa_t(.)$ is the intesity of clutter RFS K_t at time t.

From the above formulas, it can be seen that the PHD filter avoids combinatorial calculations resulting from the unknown assignment of measurements to the corresponding targets. Moreover, since the posterior intensity is a function of the state space $\mathcal{X}$ with a single target, the PHD recursion requires much less computational power than Bayesian recursion on multiple targets. Unfortunately, the PHD recursion does not admit closed-form solutions, and numerical integration suffers from the "curse of dimensionality".

3.4 Linear Gaussian PHD filter

The linear Gaussian model requires the following three assumptions in addition to the PHD recursion assumptions in Section 3.3.2:

4. Each target follows a linear Gaussian dynamical model, and the sensor has a linear Gaussian measurement model. We use the following models,

$$\begin{aligned} f_{t|t-1}(x|\zeta) &= \mathcal{N}(x; F_{t-1}\zeta, Q_{t-1}), \\ g_t(z|x) &= \mathcal{N}(z; H_t x, R_t), \end{aligned} \tag{3.7}$$

   where $\mathcal{N}(.; m, P)$ denotes a Gaussian density with the mean m and the covariance P, F_{t-1} is the state transition matrix, H_t is the observation matrix, Q_{t-1} and R_t is the process noise covariance and the observation noise covariance respectively.

   Please note that these formulas are essentially the same as those in (2.2).

5. The survival and detection probabilities are state independent,

$$\begin{aligned} p_{S,t}(x) &= p_{S,t}, \\ p_{D,t}(x) &= p_{D,t}. \end{aligned} \tag{3.8}$$

6. The intensities of the birth and spawn RFS are Gaussian mixtures of the form

$$\begin{aligned} \gamma_t(x) &= \sum_{i=1}^{J_{\gamma,t}} w_{\gamma,t}^{(i)} \mathcal{N}(x; m_{\gamma,t}^{(i)}, P_{\gamma,t}^{(i)}), \\ \beta_{t|t-1}(x|\zeta) &= \sum_{j=1}^{J_{\beta,t}} w_{\beta,t}^{(j)} \mathcal{N}(x; F_{\beta,t-1}^{(j)}\zeta + d_{\beta,t-1}^{(j)}, Q_{\beta,t-1}^{(j)}), \end{aligned} \tag{3.9}$$

   where $J_{\gamma,t}, w_{\gamma,t}^{(i)}, m_{\gamma,t}^{(i)}, P_{\gamma,t}^{(i)}$, are given model parameters that determine the shape of the birth intensity, similarly $J_{\beta,t}, w_{\beta,t}^{(j)}, F_{\beta,t-1}^{(j)}, d_{\beta,t-1}^{(j)}, Q_{\beta,t-1}^{(j)}$ determine the shape of the spawning intensity of a target with previous state ζ.

We now have everything we need to derive the recursion for the linear Gaussian PHD model.

3.4.1 Prediction step

Suppose we have a posterior intensity $v_{t-1}(x)$ from time point $t-1$ and that it is a Gaussian mixture that is the following,

$$v_{t-1}(x) = \sum_{i=1}^{J_{t-1}} w_{t-1}^{(i)} \mathcal{N}(x; m_{t-1}^{(i)}, P_{t-1}^{(i)}). \tag{3.10}$$

We know that the resulting prior intensity at time t is the sum of the intensity of the surviving targets $v_{S,t|t-1}(x)$, the intensity of the spawn targets $v_{\beta,t|t-1}(x)$ and the intensity of the newborn targets $\gamma_t(x)$.

From Formula (3.9), we know the intensity of the newborn targets $\gamma_t(x)$. However, the other two intensities we have to compute. Let us first derive the intensity of the surviving targets,

$$v_{S,t|t-1}(x) = p_{S,t} \sum_{i=1}^{J_{t-1}} w_{t-1}^{(i)} \mathcal{N}(x; m_{S,t|t-1}^{(i)}, P_{S,t|t-1}^{(i)}), \tag{3.11}$$

where

$$\begin{aligned} m_{S,t|t-1}^{(i)} &= F_{t-1} m_{t-1}^{(i)}, \\ P_{S,t|t-1}^{(i)} &= Q_{t-1} + F_{t-1} P_{t-1}^{(i)} F_{t-1}^{\intercal}. \end{aligned} \tag{3.12}$$

Now we calculate the intensity of the spawned targets,

$$v_{\beta,t|t-1}(x) = \sum_{i=1}^{J_{t-1}} \sum_{j=1}^{J_{\beta,t}} w_{t-1}^{(i)} w_{\beta,t}^{(j)} \mathcal{N}(x; m_{\beta,t|t-1}^{(i,j)}, P_{\beta,t|t-1}^{(i,j)}), \tag{3.13}$$

where

$$\begin{aligned} m_{\beta,t|t-1}^{(i,j)} &= F_{\beta,t-1}^{(j)} m_{t-1}^{(i)} + d_{\beta,t-1}^{(j)}, \\ P_{\beta,t|t-1}^{(i,j)} &= Q_{\beta,t-1}^{(j)} + F_{\beta,t-1}^{(j)} P_{\beta,t-1}^{(i)} (F_{\beta,t-1}^{(j)})^{\intercal}. \end{aligned} \tag{3.14}$$

It should be noted that Formula (3.12) and Formula (3.14) are nothing but Kalman predictions. The predicted intensity can therefore be written as,

$$v_{t|t-1}(x) = v_{S,t|t-1}(x) + v_{\beta,t|t-1}(x) + \gamma_t(x). \tag{3.15}$$

In Algorithm 1, we can see the pseudocode for the prediction part of the Gaussian mixture PHD filter.

3.4.2 Update step

Now we have the predicted intensity $v_{t|t-1}(x)$ for time t is a Gaussian mixture of the form,

$$v_{t|t-1}(x) = \sum_{i=1}^{J_{t|t-1}} w_{t|t-1}^{(i)} \mathcal{N}(x; m_{t|t-1}^{(i)}, P_{t|t-1}^{(i)}). \tag{3.16}$$

In order to calculate the resulting intensity $v_t(x)$, we first have to calculate the intensity for one detected measurement $z \in Z_t$,

$$v_{D,t}(x, z) = \sum_{i=1}^{J_{t|t-1}} w_t^{(i)}(z) \mathcal{N}(x; m_{t|t}^{(i)}(z), P_{t|t}^{(i)}), \tag{3.17}$$

where

$$\begin{aligned} w_t^{(i)}(z) &= \frac{p_{D,t} w_{t|t-1}^{(i)} q_t^{(i)}(z)}{\kappa_t(z) + p_{D,t} + \sum_{j=1}^{J_{t|t-1}} w_{t|t-1}^{(j)} q_t^{(j)}}, \\ m_{t|t}^{(i)}(z) &= m_{t|t-1}^{(i)} + K_t^{(i)}(z - H_t m_{t|t-1}^{(i)}), \\ P_{t|t}^{(i)} &= \left[I - K_t^{(i)} H_t\right] P_{t|t-1}^{(i)}, \\ K_t^{(i)} &= P_{t|t-1}^{(i)} H_t^{\intercal} (H_t P_{t|t-1}^{(i)} H_t^{\intercal} + R_t)^{-1}. \end{aligned} \tag{3.18}$$

Algorithm 1 Pseudocode of the prediction step of the Gaussian Mixture PHD filter

given $\left\{w_{t-1}^{(i)}, m_{t-1}^{(i)}, P_{t-1}^{(i)}\right\}_{i=1}^{J_{t-1}}$.

step 1. (prediction for newborn targets)
$i = 0$.
for $j = 1, \ldots, J_{\gamma,t}$:
$i := i + 1$.
$w_{t|t-1}^{(i)} = w_{\gamma,t}^{(j)}$.
$m_{t|t-1}^{(i)} = m_{\gamma,t}^{(j)}$.
$P_{t|t-1}^{(i)} = P_{\gamma,t}^{(j)}$.
end
step 2. (prediction for spawn targets)
for $j = 1, \ldots, J_{\beta,t}$:
for $l = 1, \ldots, J_{t-1}$:
$i := i + 1$.
$w_{t|t-1}^{(i)} = w_{t-1}^{(l)} w_{\beta,t}^{(j)}$.
$m_{t|t-1}^{(i)} = d_{\beta,t-1}^{(j)} + F_{\beta,t-1}^{(j)} m_{t-1}^{(l)}$.
$P_{t|t-1}^{(i)} = Q_{\beta,t-1}^{(j)} + F_{\beta,t-1}^{(j)} P_{t-1}^{(l)} (F_{\beta,t-1}^{(j)})^{\intercal}$.
end
end
step 3. (prediction for surviving targets)
for $j = 1, \ldots, J_{t-1}$:
$i := i + 1$.
$w_{t|t-1}^{(i)} = p_{S,t} w_{t-1}^{(j)}$.
$m_{t|t-1}^{(i)} = F_{t-1} m_{t-1}^{(j)}$.
$P_{t|t-1}^{(i)} = Q_{t-1} + F_{t-1} P_{t-1}^{(j)} (F_{t-1})^{\intercal}$.
end
$J_{t|t-1} = i$.

output $\left\{w_{t|t-1}^{(i)}, m_{t|t-1}^{(i)}, P_{t|t-1}^{(i)}\right\}_{i=1}^{J_{t|t-1}}$.

It is worth noting that the Formula (3.18) are just alternative transcriptions of the Formula (2.9) from the Kalman update.

The resulting intensity is then obtained by summing the intensity from the prediction step multiplied by the probability that no target was detected and the sum of the intensities over all measurements. We can write this equation in the following way:

$$v_t(x) = (1 - p_{D,t}) v_{t|t-1}(x) + \sum_{z \in Z_t} v_{D,t}(x, z). \tag{3.19}$$

In Algorithm 2, we can see the pseudocode for the update part of the Gaussian mixture PHD filter.

Algorithm 2 Pseudocode of the update step of the Gaussian Mixture PHD filter

given $\left\{w_{t|t-1}^{(i)}, m_{t|t-1}^{(i)}, P_{t|t-1}^{(i)}\right\}_{i=1}^{J_{t|t-1}}$, and the measurement set Z_t.

step 1. (counting the auxiliary variables)
for $j = 1, \ldots, J_{t|t-1}$:
$\eta_{t|t-1}^{(j)} = H_t m_{t|t-1}^{(j)}$.
$S_t^{(j)} = R_t + H_t P_{t|t-1}^{(j)} (H_t)^\intercal$.
$K_t^{(j)} = P_{t|t-1}^{(j)} (H_t)^\intercal \left[S_t^{(j)}\right]^{-1}$.
$P_{t|t}^{(j)} = \left[I - K_t^{(j)} H_t\right] P_{t|t-1}^{(j)}$.
end
step 2. (update)
for $j = 1, \ldots, J_{t|t-1}$:
$w_t^{(j)} = (1 - p_{D,t}) w_{t|t-1}^{(j)}$.
$m_t^{(j)} = m_{t|t-1}^{(j)}$.
$P_t^{(j)} = P_{t|t-1}^{(j)}$.
end
$l := 0$
for each $z \in Z_t$:
$l := l + 1$.
for $j = 1, \ldots, J_{t|t-1}$:
$w_t^{(lJ_{t|t-1}+j)} = p_{D,t} w_{t|t-1}^{(j)} \mathcal{N}(z, \eta_{t|t-1}^{(j)} S_t^{(j)})$.
$m_t^{(lJ_{t|t-1}+j)} = m_{t|t-1}^{(j)} + K_t^{(j)} (z - \eta_{t|t-1}^{(j)})$.
$P_t^{(lJ_{t|t-1}+j)} = P_{t|t}^{(j)}$.
end
for $j = 1, \ldots, J_{t|t-1}$:
$w_t^{(lJ_{t|t-1}+j)} := \frac{w_t^{(lJ_{t|t-1}+j)}}{\kappa_t(z) + \sum_{i=1}^{J_{t|t-1}} w_t^{(lJ_{t|t-1}+i)}}$.
end
end
$J_t = lJ_{t|t-1} + J_{t|t-1}$.

output $\left\{w_t^{(i)}, m_t^{(i)}, P_t^{(i)}\right\}_{i=1}^{J_t}$.

Chapter 4
Implementation

In this chapter, we show the important parts of our Gaussian mixture PHD filter implementation. We first discuss the GaussianMix class and then the PHD class.

4.1 GaussianMix class

The class is only used to encapsulate variables. It has only three arguments, an array of weights, an array of means, and an array of covariance matrices. Together, these three arrays represent a Gaussian mixture that represents the RFS of the target states at each time instant t.

This class has only one method, newborn_targets, which does nothing more than create a copy of its instance. This method comes in handy in the prediction step and is purely to make the code cleaner.

4.2 PHD class

This class is much more complicated than the GaussianMix class, as it controls the entire filtering process. When creating a PHD instance, you need to pass a dictionary argument that contains all the required variables for the task. This dictionary must contain the following:

- p_s: probability of target survival
- p_d: probability of target detection
- F: state transition matrix
- Q: process noise covariance matrix
- H: measurement matrix
- R: measurement noise covariance matrix
- $Birth$: list of parameters for newborn targets
- w, F, d, Q: parameters for spawning targets
- T, U, J_max: parameters for truncation
- $clutter\_rate$: clutter intensity, the average number of false detections
- $region\_size$: the size of the measuring area where we work

Code listing 1 Code of the filter_data method

```
def filter_data(self, Z: List[List[np.ndarray]]):
        v = GaussianMix([], [], []), estimates = []
        for z in Z:
            v = self.prediction(v)
            v = self.update(v, z)
            v = self.pruning(v)
            est = self.state_estimation(v)
            estimates.append(est)
        return estimates
```

Thus initialized class can already serve as a full-fledged Gaussian Mixture PHD filter just call the method filter_data Code1 to which we pass all the measurements that we have measured at each point in time. Inside this method, we first create an empty Gaussian mixture since we have no information yet. We then loop this mixture through all the methods. We do the following sequentially: prediction, update using the measurements at a given point in time, then prune those peaks that are not probable, and finally make an estimate of the number of targets and store their parameters in an array. After going through all the measurements, we return an array of estimates where each index contains the number of estimates at a given time instant.

4.2.1 Prediction method

In the prediction method Code 2, the following happens, we prepare a Gaussian mixture of newborn targets, which is just a copy of the Gaussian mixture $self.birth$. Next, we prepare a mixture of surviving targets and a mixture of spawned targets. Finally, we only need to combine these three mixtures together as the resulting Gaussian mixture.

Code listing 2 Code of the prediction method

```
    def prediction(self, x: GaussianMix) -> GaussianMix:
        v_birth = self.birth.newborn_targets()
        v_survived = compute_survivals(x, self.p_s, self.F, self.Q)
        v_spawn = self.spawn(x)
        return GaussianMix(v_birth.w + v_survived.w + v_spawn.w,
                           v_birth.m + v_survived.m + v_spawn.m,
                           v_birth.P + v_survived.P + v_spawn.P)
```

The calculation of surviving targets and spawned targets are shown in Code 3 and Code 4. This code fully corresponds to equations (3.12) and (3.14) and therefore does not need further explanation.

Code listing 3 Code of the compute_survivals function

```
def compute_survivals(x: GaussianMix, p, F: np.ndarray, Q: np.ndarray):
    w = list(np.multiply(copy.deepcopy(x.w), p)), m = [], P = []
    for target in x.m:
        m.append(F @ target)
    for matrix in x.P:
        P.append(Q + F @ matrix @ F.T)
    return GaussianMix(w, m, P)
```

Code listing 4 Code of the spawn method

```
def spawn(self, x: GaussianMix) -> GaussianMix:
    w = [], m = [], P = []
    for i, w_v in enumerate(x.w):
        for j, w_spawn in enumerate(self.w_spawn):
            w.append(w_v * w_spawn)
            m.append(self.F_spawn[j] @ x.m[i] + self.d_spawn[j])
            P.append(self.Q_spawn[j] + self.F_spawn[j] @ x.P[i] @ self.F_spawn[j].T)
    return GaussianMix(w, m, P)
```

Code listing 5 Code of the update method

```
    def update(self, x: GaussianMix, Z: List[np.ndarray]) -> GaussianMix:
        # Counting auxiliary parameters
        v_tmp = compute_survivals(x, self.p_s, self.H, self.R)
        eta = v_tmp.m, S = v_tmp.P
        S_inv = inversion_matrices_in_list(S)
        K_k = [], P_kk = []
        kappa = self.clutter / (self.region_size * 2)**2
        for i in range(len(x.w)):
            k = (x.P[i] @ self.H.T @ S_inv[i])
            P_kk.append(x.P[i] - k @ self.H @ x.P[i])
            K_k.append(k)

        # In case no measurement is from the target.
        w = [], m = copy.deepcopy(x.m), P = copy.deepcopy(x.P)
        for item in x.w:
            w.append((1 - self.p_d) * item)

        for z in Z:
            if z == []:
                continue
            values = []
            for j in range(len(x.w)):
                values.append(self.p_d * x.w[j] * mvn.pdf(z, eta[j], S[j]))
                m.append(m[j] + K_k[j] @ (z - eta[j]))
                P.append(P_kk[j])
            denominator = np.sum(values) + kappa
            for j in range(len(x.w)):
                numerator = values[j]
                w.append(numerator / denominator)
        return GaussianMix(w, m, P)
```

4.2.2 Update method

This key PHD filter method, which can be seen in Code 5, takes as parameters the Gaussian mixture after the prediction step and the measurement field measured at a given time instant. In the first part of the method, we need to compute auxiliary parameters that are used later for the update. These equations correspond to Equations (3.18).

Once we have all the auxiliary variables ready, the update step is divided into two variants. First, the case where all of the measurements we got do not correspond to any of the objects and are, therefore, all false. In this case, we just need to copy the values from the prediction step and reduce the weights on these measurements accordingly, which corresponds to the first part of Equation (3.19). The second option corresponds to the interpretation of our measurements from the targets, and we update our components with them. It is important to note that every single measurement updates all the components from the prediction step. This exponentially increases the number of components in our space. However, only the components with high weights have a real value. The other components just cover all the possibilities that may occur.

From the above, it is clear that such an algorithm would be computationally feasible. Therefore a simple pruning method is added to the implementation to only let some components into the next time instant.

4.2.3 Pruning and state_estimation method

The main purpose of this method is to make the overall complexity of the algorithm computable and not exponential. Code 6 starts by selecting components from the Gaussian set that have weights higher than the threshold T. Those components that have lower weights are already so unlikely that there is no point in propagating them to the next time step.

Once we have selected the components, we iterate them sequentially, starting from the ones with the highest weights to the smallest ones. At each iteration, we try to see if the current component has other components that are close to it. If it does have such components, they were probably created from the same target, and there is no point in keeping them all in memory, so we merge them into one component by averaging the values of those components. Then we just remove all the components we worked with in that iteration from the set I and continue to the next iteration of the loop.

Once set I is empty, we look at how many components we have, and if there are more than J_max, we only select the J_max component with the highest weight. The result of this method is a pruned Gaussian mixture that makes our algorithm computable.

The last method we want to mention here is the state_estimation method. Its purpose is simple: select the components with the highest weights and declare them as real targets, thus determining the number of targets on the surface and their position. Indeed, if we did not have this method, we would most likely label all the components of the Gaussian mixture that survived the pruning method as targets. This would certainly not be a correct solution since some of these components may have survived the pruning method, but their weights may still be very small and, therefore, improbable. The implementation itself is then very simple. We declare as targets those components that have weights higher than a predefined threshold. In our code, we use a threshold value of 0.5.

Code listing 6 Code of the pruning method

```
    def pruning(self, x: GaussianMix):
        Iw = [], Im = [], IP = [], idxI = [], cnt = 0
        for i in range(len(x.w)):
            if x.w[i] > self.T:
                Iw.append(x.w[i]), Im.append(x.m[i]), IP.append(x.P[i])
                idxI.append(cnt)
                cnt += 1
        invIP = inversion_matrices_in_list(IP)

        w = [], m = [], P = []
        while len(idxI) > 0:
            # Finding argmax
            j = idxI[0]
            for i in idxI:
                if Iw[i] > Iw[j]:
                    j = i

            # Gaussian components that are close together
            L = []
            for i in idxI:
                if ((Im[i] - Im[j]) @ invIP[i] @ (Im[i] - Im[j])) <= self.U:
                    L.append(i)

            w_new = 0.0, m_new = np.zeros(len(Im[0]))
            P_new = np.zeros((len(Im[0]), len(Im[0])))

            # Merging components
            for i in L:
                w_new += Iw[i]
                m_new += (Iw[i] * Im[i].T).T
            m_new /= w_new
            for i in L:
                P_new += Iw[i] * (IP[i] + np.outer(m_new - Im[i], m_new - Im[i]))
            P_new /= w_new
            w.append(w_new), m.append(m_new), P.append(P_new)

            # Reducing the set I
            new_I = []
            for element in idxI:
                if element not in L:
                    new_I.append(element)
            idxI = new_I

        if len(w) > self.J_max:
            tmp = np.array(w).argsort()[-self.J_max:]
            w = [w[i] for i in tmp]
            m = [m[i] for i in tmp]
            P = [P[i] for i in tmp]
        return GaussianMix(w, m, P)
```

Chapter 5

Experiments

This chapter provides simulation examples to demonstrate that the described implementation is feasible and robust. The examples show that the filter is able to localize the targets under various conditions. We stress that the goal is not to study the estimation performance itself.

All examples are based on the CVM model known from Section 1.3.3. We think of these examples as tracking aircraft on a 2D surface using a radar instrument. The radar measuring period is one second. The state space model is characterized as follows: The transition matrix F and the process noise covariance matrix Q have the form

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad Q = q^2 \begin{bmatrix} \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & 0 & \Delta t & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 & \Delta t \end{bmatrix}, \tag{5.1}$$

where Δt is the time step, in this case, one second, and q is the parameter of the noise matrix, which we consider as $q = 5$.

The measurement matrix H and the measurement noise covariance matrix R are

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \qquad R = r^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{5.2}$$

where r is the parameter of the noise matrix, using this parameter we know how inaccurate our sensor is. In our case, we set $r = 5$.

Areas for the birth of new targets, which we use for all subsequent simulations, are as follows: We consider that targets can only be born at coordinates $[0, 0]^\intercal$ and $[20, 20]^\intercal$. We can think of these points as two airports from which planes take off, i.e., our new targets. These locations have the same scatter matrix P and weight w with which the new targets are born. In addition, we describe everything using the following equations,

$$w_1 = w_2 = 0.05, \qquad m_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, m_2 = \begin{bmatrix} 20 \\ 20 \\ 0 \\ 0 \end{bmatrix}, \qquad P_1 = P_2 = \begin{bmatrix} 100 & 100 & 100 & 100 \end{bmatrix}. \tag{5.3}$$

These parameters give a Gaussian mixture Birth, which we then pass to the PHD class.

The parameters used in the pruning procedure are as follows: The parameter T is a threshold, once the weight of the component w falls below this threshold, the component is removed. The parameter U determines the threshold of the distance between two components, where if the

distance is smaller then the components are merged into one. Finally, the parameter J_max determines the maximum number of components that can pass to the next step. The values of these parameters are following,

$$T = 0.00005, \qquad U = 4, \qquad J\_max = 100. \tag{5.4}$$

Finally, the last two parameters that we consider the same for all examples are the survival probability of the component p_S and the detection probability of the target p_D, their values are following,

$$p_S = 0.98, \qquad p_D = 0.95. \tag{5.5}$$

5.1 Example 1: One target with clutter

The first example demonstrates the movement of a single target in the monitored area, namely a square $[-500, -500]^\intercal \times [500, 500]^\intercal$. The simulation starts at time 0 seconds and ends at time 50 seconds. The clutter_rate is set to 5. We simulated that the object is created at time 0 at position $[0, 0]^\intercal$.

This experiment demonstrates that if only one object exists in the space, then the PHD filter behaves like a Kalman filter. The results of our experiment can be seen in Figure 5.1. In the

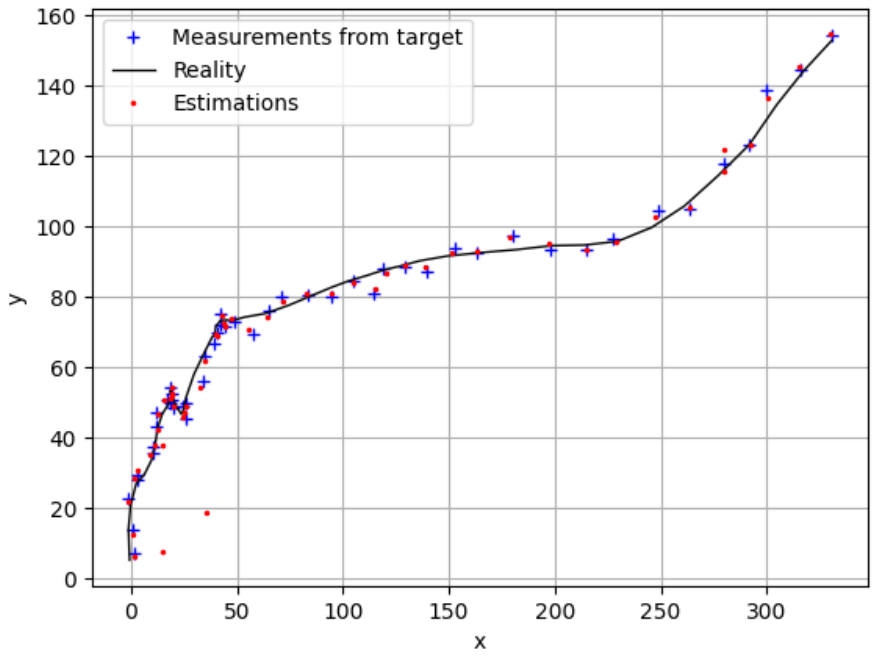


Figure 5.1 Real trajectory of an airplane, started at point $[0, 0]^\intercal$ and PHD filter estimations of position, simulated for 50 seconds. Only the interesting part of the space is depicted. The clutter measurements are not depicted as they differ at each time instant.

graph, we can observe that our PHD filter was able to determine the actual position of the aircraft quite accurately despite the clutter measurements. We can also observe that sometimes

over time, more clutter measurements probably occurred around our birth locations (airports), causing two erroneous estimates in the area around the coordinates $[30, 30]^\intercal$. These estimates most likely arose from clutter measurements. It is noticeable that these estimates were not further supported by the measurements leading to their weight dropping so much that they were removed.

5.2 Example 2: Three objects without clutter

This example considers the simulation of three objects in space with the clutter measurement turned off. It follows that if we take any measurements, they surely belong to one of our real objects in space.

Figure 5.2 shows three trajectories of the targets. All targets originated at time 0 at position $[0, 0]^\intercal$ and moved for 50 seconds. Since all measurements originated from real targets, we see that all red dots (estimates) are close to one of the real trajectories. It can be seen that the

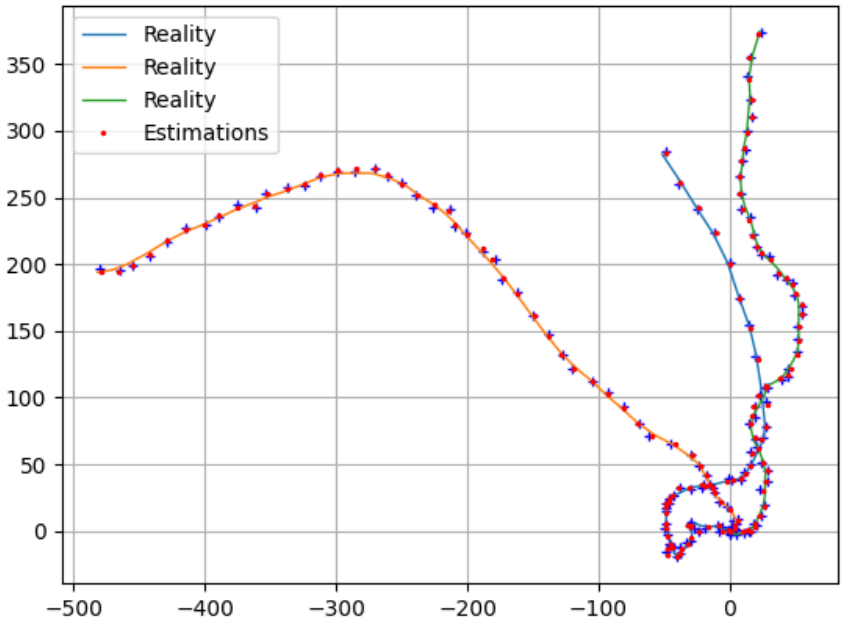


Figure 5.2 Real trajectories of three airplanes, started at point $[0, 0]^\intercal$ and PHD filter estimations of position, simulated for 50 seconds. Only the interesting part of the space is depicted. The clutter measurements are not depicted as they differ at each time instant.

PHD filter easily handled the situation at the beginning of the simulation. All three targets were in close vicinity. There were measurements coming in that could not be clearly identified as belonging to which target. A simple Kalman filter could not solve this.

5.3 Example 3: Three objects with clutter

Let us simulate the same trajectories as in Example 5.2, but this time add clutter measurements. We simulate on a square surface with corners $[-500, -500]^\intercal$ to $[500, 500]^\intercal$. We set the *clutter_rate* to 200, so at each time instant, we get approximately 200 false measurements uniformly spread over our surface. As in the previous examples, we let the simulation run for 50 seconds.

The evolution of the situation is depicted in Figures 5.3, 5.4 and 5.5 for specific time instants 10s, 30s, and 50s. It can be seen that the initialization went well since each trajectory has at least one component (red dots) connected. Moreover, we can say that the estimation is accurate since it corresponds to the real trajectory of the objects. We can also see that the yellow trajectory has two components bound since we received two measurements near this trajectory at time $t = 10$.

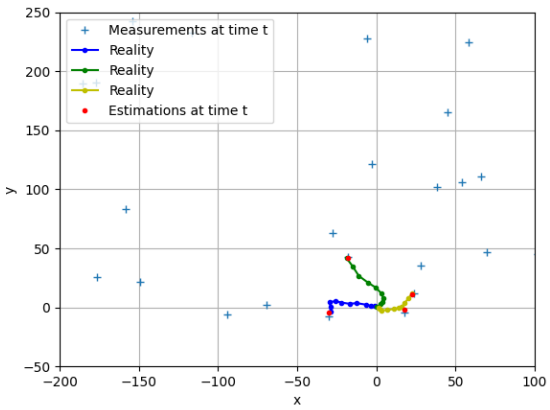


Figure 5.3 Simulation of three aircraft at time instant $t = 10$.

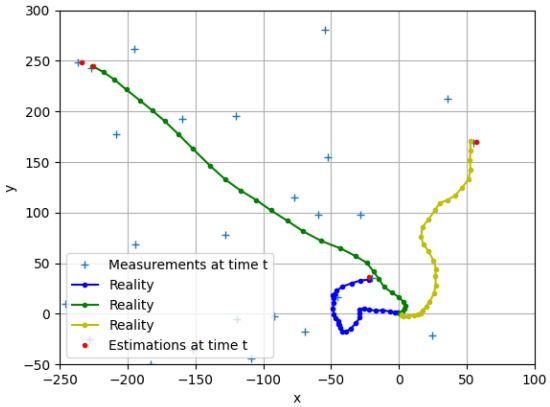


Figure 5.4 Simulation of three aircraft at time instant $t = 30$.

In Figure 5.6 we then see the complete trajectories with all estimates. Many times our PHD filter makes more estimates than how many targets are actually in the space which we unfortunately have to accept. With this example, we have shown that our PHD filter is robust enough to handle spurious measurements.

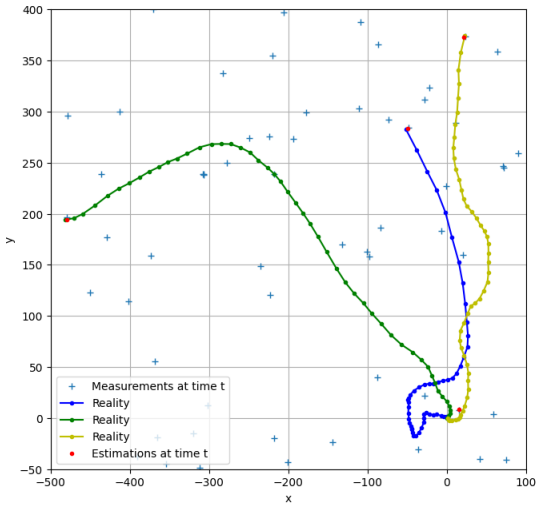


Figure 5.5 Simulation of three aircraft at time instant $t = 50$.

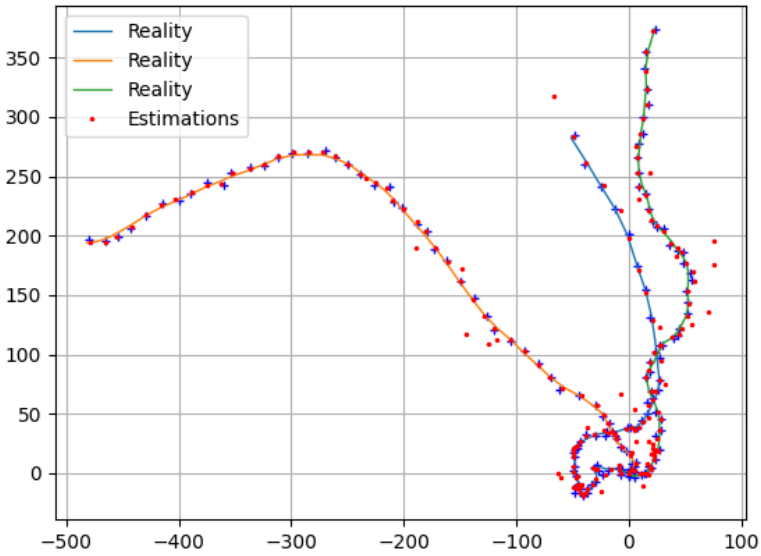


Figure 5.6 Real trajectories of three airplanes with clutter, started at point $[0, 0]^\intercal$ and PHD filter estimations of position, simulated for 50 seconds.

5.4 Example 4: Birth, Spawn, and Dead of objects

In the last example, we show the maximum coverage of our filter. We consider that not all objects are born at time 0. Next, we consider that objects can be terminated before the simulation ends. Finally, we assume that other objects can be spawned from one object. In the real world, this example can be thought of as an aircraft carrier from which planes take off (are spawned) sequentially.

First, let us describe all terms that drive the simulation. The hidden process model is characterized by the spawning matrices F_{spawn} and Q_{spawn} these are identical to the standard target process model variables F and Q, respectively. This corresponds to the idea that the spawned targets inherit the behavior of their parents. The initial weight of spawned targets is $w_{spawn} = 0.05$, and the vector driving the distance between the spawned target and its parent is $d_{spawn} = [5, 5, 5, 5]^\intercal$. Similarly to the previous cases, the simulation area is a square $[-400, 400] \times [-400, 400]$. The clutter rate is 150. The simulation horizon is 90 seconds. The events are as follows:

- $t = 1, ..., 9$ – no target,
- $t = 10$ – a target (aircraft carrier) is detected at the position $[0, 0]^\intercal$,
- $t = 30$ – a newly spawned target occurs (the first aircraft takes off),
- $t = 40$ – another spawned target occurs (the second aircraft takes off),
- $t = 60$ – the aircraft carrier disappears from the area,
- $t = 90$ – both spawned targets (aircrafts) disappear and the simulation ends.

In Figure 5.7 we see the current situation of the space at time $t = 5$. So far, no objects have appeared, and all measurements are from clutter. Our filter estimates correctly, there are no objects in the area.

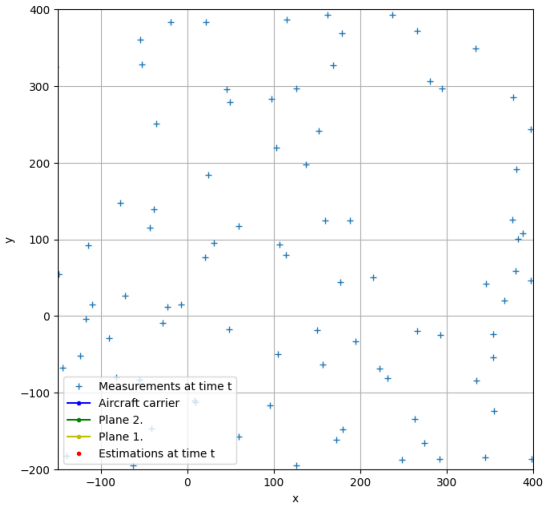


Figure 5.7 Simulation of the aircraft carrier and two planes at time $t = 5$.

In Figure 5.8 we can see the situation at time $t = 38$. Two trajectories are visible. Our PHD filter has captured both of these trajectories since it estimates that there are targets. Then in Figure 5.9 we see the situation at time $t = 65$. The aircraft has already disappeared, and the PHD filter no longer estimates its position while it tracks both born aircrafts.

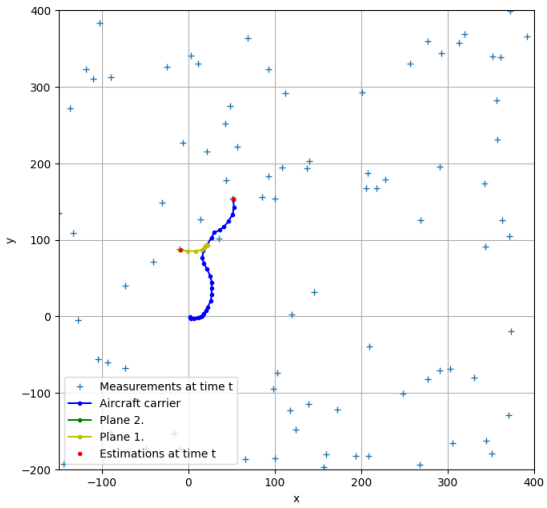


Figure 5.8 Simulation of the aircraft carrier and two planes at time $t = 38$.

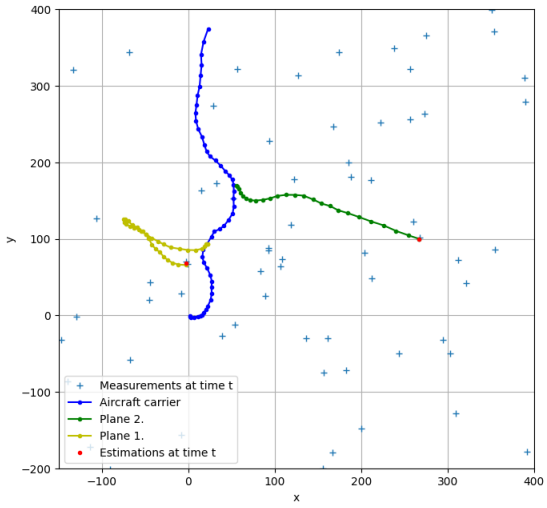


Figure 5.9 Simulation of the aircraft carrier and two planes at time $t = 65$.

In Figure 5.10 we can see the estimates of the number of targets on the surface at each time instant made by our PHD filter and also the real number of objects. With this graph, we want

to show that the PHD filter can be used to estimate the number of targets quite accurately. The filter shares the common feature of the multiple target tracking algorithms: the number of estimated targets is often higher than their true number. However, the false (nonexistent) targets do not follow the prescribed state-space model, which makes them improbable at the next iteration of the algorithm. These false targets are hence quickly pruned.

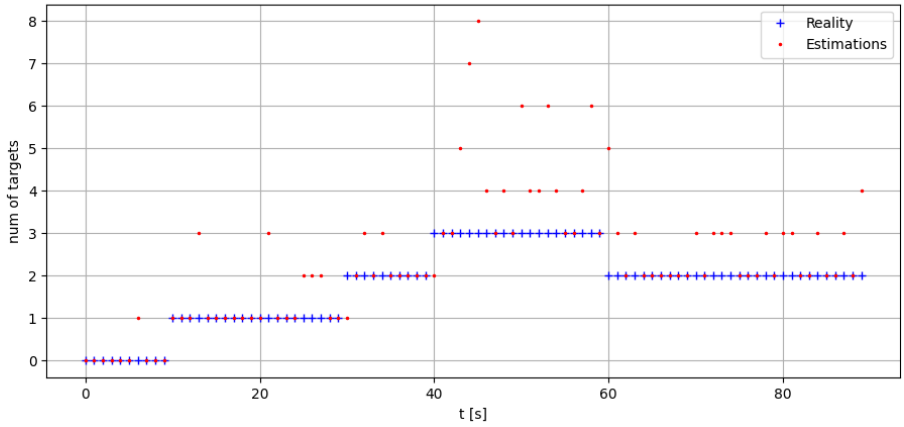


Figure 5.10 Estimation of the number of targets on the surface during the time and real number of targets.

Conclusion

In this final chapter, we summarise what has been done and also what improvements could be made.

Summary of thesis

The main aims of this thesis were:

1. Explore the area of single-target tracking using the Kalman filter.
2. Study all the necessary knowledge to track multiple targets using the PHD filter.
3. Implement our own linear Gaussian PHD filter and experimentally verify its performance.

The necessary prerequisites to derive the Kalman filter are explained in Chapter 1. Then, in Chapter 2, the Kalman filter itself is derived and its application to single-target tracking is shown. With these two chapters, we have covered the first objective of our thesis. In Chapter 3, we explained all the important concepts for understanding the PHD filter in a step-by-step manner. We pointed out the high connection between the PHD filter and the Kalman filter. This fully covered the second objective of our thesis. Next, in Chapter 4, we explained the main parts of our implementation of the PHD filter. We have described the imperfections of the PHD filter, including the necessity of a pruning function. Finally, in Chapter 5, we demonstrated the functionality of our implementation on simulated data and then discussed the simulation results. These chapters then covered the third goal of our thesis.

Possible future works

As mentioned earlier, the main goal of this work was to understand linear Gaussian PHD filtering. However, we can still extend our work to nonlinear PHD filtering models. This would mean that the process of states and measurements would be nonlinear. Such a model would then use the following equations:

$$\begin{aligned} x_t &= \varphi_t(x_{t-1}, \nu_{t-1}), \\ z_t &= h_t(x_t, \varepsilon_t), \end{aligned} \tag{5.6}$$

where φ_t and h_t are known nonlinear functions and ν_{t-1} and ε_t are zero-mean Gaussian process noise and measurement noise with covariances Q_{t-1} an R_t, respectively [17]. These equations would then lead to the extended Kalman PHD filter, which is a nonlinear PHD filter.

Bibliography

1. BISHOP, Christopher M; NASRABADI, Nasser M. *Pattern recognition and machine learning*. Vol. 4. Springer, 2006. No. 4.
2. BENHAMOU, Eric; SALTIEL, David; VEREL, Sebastien; TEYTAUD, Fabien. BCMA-es: A Bayesian approach to CMA-ES. *SSRN Electronic Journal*. 2019. Available from DOI: 10.2139/ssrn.3365449.
3. SCHLAIFER, Robert; RAIFFA, Howard. *Applied statistical decision theory*. Harvard University, 1961.
4. KOOPMAN, Bernard O. On distributions admitting a sufficient statistic. *Transactions of the American Mathematical Society*. 1936, vol. 39, no. 3, pp. 399–409. Available from DOI: 10.1090/s0002-9947-1936-1501854-3.
5. CASELLA, George; BERGER, Roger L. *Statistical Infence*. Duxbury Press, 2002.
6. DEDECIUS, Kamil. *Laboratory of Statistical Modelling - Sequential estimates* [[online]]. 2023. Available also from: https://gitlab.fit.cvut.cz/dedeckam/ni-lsm-cviceni. Accessed: 2023-03-27.
7. MARKOV, Konstantin; MATSUI, Tomoko; SEPTIER, Francois; PETERS, Gareth. Dynamic speech emotion recognition with state-space models. In: *2015 23rd European Signal Processing Conference (EUSIPCO)*. 2015, pp. 2077–2081. Available from DOI: 10.1109/EUSIPCO.2015.7362750.
8. BROGAN, William L. *Modern Control Theory*. [3rd ed.] Prentice Hall, 1991.
9. KALMAN, Rudolf E. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*. 1960, vol. 82, no. 1, pp. 35–45. Available from DOI: 10.1115/1.3662552.
10. THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic Robotics*. [1st ed.] Cambridge: The MIT Press, 2006.
11. GURAJALA, Ramakrishna; CHOPPALA, Praveen B.; MEKA, James Stephen; TEAL, Paul D. Derivation of the Kalman filter in a Bayesian filtering perspective. In: *2021 2nd International Conference on Range Technology (ICORT)*. 2021, pp. 1–5. Available from DOI: 10.1109/ICORT52730.2021.9581918.
12. BREKKE, Edmund. *Fundamentals of Sensor Fusion*. [1st ed.] Norwegian University of Science and Technology, 2020.
13. WOLFF, Christian. *Radar basics* [[online]]. 2023. Available also from: https://www.radartutorial.eu/11.coherent/co04.en.html. Accessed: 2023-04-28.
14. BLACKMAN, Samuel S. *Multiple-target tracking with radar applications*. Artech House, 1986.

15. BAR-SHALOM, Yaakov; FRED, Daum.; HUANG, Jim. The probabilistic data association filter. *IEEE control systems*. 2010, vol. 29, no. 6, pp. 82–100. Available from DOI: 10.1109/MCS.2009.934469.

16. STONE, Lawrence David; STREIT, Roy L.; CORWIN, Thomas L.; BELL, Kristine L. *Bayesian multiple target tracking*. [2st ed.] Artech House, 2014.

17. VO, Ba-Ngu; MA, Wing-Kin. The Gaussian Mixture Probability Hypothesis Density Filter. *IEEE Transactions on Signal Processing*. 2006, vol. 54, no. 11, pp. 4091–4104. Available from DOI: 10.1109/TSP.2006.881190.

18. STREIT, Roy L.; LUGINBUHL, Tod E. Point target probabilistic multiple hypothesis tracking. *NUWC-NPT Technical Report*. 1995.

19. PRIHODA, Zachary; JAMGOCHIAN, Arec; MOORE, Ben; LANGE, Bernard. Probability Hypothesis Density Filter Implementation and Application. *Stanford University*. 2019.

20. XIA, Yuxuant. *Lecture 4: Random finite sets* [[online]]. 2021. Available also from: https://chalmersuniversity.app.box.com/s/kbkmglktznkb2tjlr9pqefz3ezbiyw8p. Accessed: 2023-04-13.

21. MAHLER, Ronald P.S. Multitarget Bayes filtering via first-order Multitarget Moments. *IEEE Transactions on Aerospace and Electronic Systems*. 2003, vol. 39, no. 4, pp. 1152–1178. Available from DOI: 10.1109/taes.2003.1261119.

Enclosed medium contents

- `readme.txt` ..... the file with media contents description
- `implementation` ..... the directory of code of the implementation
- `src` ..... the directory of source codes
  - `thesis` ..... the directory of LaTeX source codes of the thesis
  - `thesis.pdf` ..... the thesis text in PDF format