Object tracking algorithms

Pavlinić, Nikola

Object tracking algorithms

Pavlinić, Nikola

UNIVERSITY OF ZAGREB
FACULTY OF SCIENCE
DEPARTMENT OF MATHEMATICS

Nikola Pavlinić

Object tracking algorithms

Master thesis

Mentor:
Prof.Dr.Sc.Bojan Basrak

Zagreb, July 2021.

This master thesis was defended on _____ in front of an examination commission composed of:

1. _____ , president
2. _____ , member
3. _____ , member

The committee evaluated the thesis with a grade _____ .

Signatures of committee members:

1. _____
2. _____
3. _____

I dedicate this master thesis to my parents and my family. I would like to thank Prof.Dr.Sc. Bojan Basrak and Dr.Sc. Drago Špoljarić for their advices and suggestions.

Contents

1 Introduction

Nowadays, object tracking technology is in various aspects of our lives. One of the first applications was in military purpose, but now we use it for surveillance, monitoring, science in every day life. Object tracking technology is used in GPS-based navigation, aircraft radars, weather monitoring and video surveillance. We need different kinds of sensors to apply the tracking algorithms, so we are interested in sensors outputs. In this master thesis, we will introduce three well-known object tracking algorithms, Kalman filter, point mass filter and particle filter. We will also consider the advantages and disadvantages of each filter and present the implementation of Kalman filter in R programming language. Since all three methods are based on recursive Bayesian logic we will start with the recursive Bayesian solution. We also note that the main literature in the thesis is the book Fundamentals of object tracking, S. Challa, M. R. Morelande, D. Mušicki, R. J. Evans [1].

2 Recursive Bayesian solution

First, we give a definition of the conditional probability of event A given observation of event B.

Definition 1. *Let A and B be two related events, the conditional probability of event A given observation of event B is*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A,B)}{\mathbb{P}(B)}. \tag{1}$$

Using (1) twice, Bayes' theorem [9] can be written as

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}. \tag{2}$$

If X and Y are two continuous random variables with densities functions $p(x)$ and $p(y)$, respectively, Bayes' theorem can be written as follows [8]:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}, \tag{3}$$

where $p(x|y)$ is the conditional density function of X given the value of Y, and $p(y|x)$ is the conditional density function of Y given the value of X.

Let $\mathbf{x}$ be a random variable of interest, for example the state of the object, and $\mathbf{y}$ be measurement related to $\mathbf{x}$, such as different types of sensors outputs. We want to use measurement $\mathbf{y}$ related to $\mathbf{x}$ to update our current knowledge about $\mathbf{x}$. We represent our knowledge about $\mathbf{x}$ with a continuous or discrete density function $p(\mathbf{x})$, depending on whether $\mathbf{x}$ takes continuous or discrete values. We update $p(\mathbf{x})$ with new sensor outputs $\mathbf{y}$ by $p(\mathbf{x}|\mathbf{y})$ as in the assertion of Bayes' theorem (3). Now we can see that for a fixed $\mathbf{y}$, $p(\mathbf{x})$ is altered by $p(\mathbf{y}|\mathbf{x})$ to become $p(\mathbf{x}|\mathbf{y})$. Once $\mathbf{y}$ is known, $p(\mathbf{y})$ is the same for all $\mathbf{x}$ values, so we can

ignore the effect of $p(\mathbf{y})$. The term $\mathcal{L}(\mathbf{x}) = p(\mathbf{y}|\mathbf{x})$ viewed as a function of $\mathbf{x}$ is called the *likelihood function.* We can write $p(\mathbf{x}|\mathbf{y}) \approx p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$. Considering $p(\mathbf{y})$ as a function of $\mathbf{x}$, $p(\mathbf{y})$ is constant and is called the normalization constant or normalization factor, which ensures that $p(\mathbf{x}|\mathbf{y})$ sums up or integrates to 1 as a function of $\mathbf{x}$. The initial distribution $p(\mathbf{x})$ is called the *prior distribution* and $p(\mathbf{x}|\mathbf{y})$ is called the *posterior distribution*, which is the new distribution of $\mathbf{x}$.

2.1 Application to object tracking

Let $\mathbf{y}^k = (\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_k)$, where $\mathbf{y}_i$ is the measurement at time i, $i \in \{1, 2, ..., k\}$, are the sensor outputs. Bayes' theorem (2) derives $p(\mathbf{S}^k|\mathbf{y}^k)$ as

$$p(\mathbf{S}^k|\mathbf{y}^k) = \frac{p(\mathbf{y}^k|\mathbf{S}^k)p(\mathbf{S}^k)}{p(\mathbf{y}^k)}, \tag{4}$$

where $p(\mathbf{S}^k) = p(\mathbf{S}_k, \mathbf{S}_{k-1}, ..., \mathbf{S}_0)$ is the joint probability density function, and $\mathbf{S}_k$ denotes a generic object state representing the single-object or multiple-object states, or the number of objects, or the identity of an object, or a combination of them at time k. The prior distribution of $\mathbf{S}^k$ is $p(\mathbf{S}^k)$. The conditional probability distribution $p(\mathbf{S}^k|\mathbf{y}^k)$ is the posterior distribution of the object state, that is, the object state after the measured values $\mathbf{y}^k$ are obtained. The likelihood function is $p(\mathbf{y}^k|\mathbf{S}^k)$ and $p(\mathbf{y}^k)$ is the normalization factor that ensures that the resulting probability distribution $p(\mathbf{S}^k|\mathbf{y}^k)$ satisfies the axioms of probability and sums up to 1. The Bayesian solution (4) can be extended to a recursive solution. The term $\mathbf{y}^k$ can be written as $\mathbf{y}^k = (\mathbf{y}_k, \mathbf{y}^{k-1})$. Now we can rewrite the terms in (4) using (1) as follows:

$$p(\mathbf{y}^k|\mathbf{S}^k) = p(\mathbf{y}_k, \mathbf{y}^{k-1}|\mathbf{S}^k) = p(\mathbf{y}_k|\mathbf{y}^{k-1}, \mathbf{S}^k)p(\mathbf{y}^{k-1}|\mathbf{S}^k),$$

where $p(\mathbf{y}^{k-1}|\mathbf{S}^k) = p(\mathbf{y}^{k-1}|\mathbf{S}^{k-1})$, because the measurements at time $k-1$ do not depend on the object state at times greater than $k-1$. Now follows:

$$\begin{aligned} p(\mathbf{y}^k|\mathbf{S}^k) &= p(\mathbf{y}_k|\mathbf{y}^{k-1}, \mathbf{S}^k)p(\mathbf{y}^{k-1}|\mathbf{S}^{k-1}), \\ p(\mathbf{S}^k) &= p(\mathbf{S}_k, \mathbf{S}^{k-1}) = p(\mathbf{S}_k|\mathbf{S}^{k-1})p(\mathbf{S}^{k-1}), \\ p(\mathbf{y}^k) &= p(\mathbf{y}_k, \mathbf{y}^{k-1}) = p(\mathbf{y}_k|\mathbf{y}^{k-1})p(\mathbf{y}^{k-1}). \end{aligned}$$

Substituting those terms in (4) we end up with the recursive form of the Bayesian solution:

$$p(\mathbf{S}^k|\mathbf{y}^k) = \frac{p(\mathbf{y}_k|\mathbf{y}^{k-1}, \mathbf{S}^k)p(\mathbf{S}_k|\mathbf{S}^{k-1})}{p(\mathbf{y}_k|\mathbf{y}^{k-1})}p(\mathbf{S}^{k-1}|\mathbf{y}^{k-1}), \tag{5}$$

where $p(\mathbf{S}^{k-1}|\mathbf{y}^{k-1}) = \frac{p(\mathbf{y}^{k-1}|\mathbf{S}^{k-1})p(\mathbf{S}^{k-1})}{p(\mathbf{y}^{k-1})}$. We can simplify the measurement likelihood function $p(\mathbf{y}_k|\mathbf{y}^{k-1}, \mathbf{S}^k)$ to $p(\mathbf{y}_k|\mathbf{S}_k)$ under the assumption that measurements at a given time depend only on the object states at the corresponding time and are conditionally independent

of measurements at other times, and that they depend only on the current states of the objects via $\mathbf{S}_k$ and not on its entire sequence of states. The majority of systems obey the Markov property, where the current state does not depend on previous states given the last state. This means that $\mathbf{S}_k$ depends only on $\mathbf{S}_{k-1}$ and not on $(\mathbf{S}_{k-2}, ..., \mathbf{S}_0)$. Consequently, $p(\mathbf{S}_k|\mathbf{S}^{k-1}) = p(\mathbf{S}_k|\mathbf{S}_{k-1})$, which brings us to the recursive form of the Bayesian solution to the object tracking problem:

$$p(\mathbf{S}^k|\mathbf{y}^k) = \frac{p(\mathbf{y}_k|\mathbf{S}^k)}{p(\mathbf{y}_k|\mathbf{y}^{k-1})} p(\mathbf{S}_k|\mathbf{S}_{k-1}) p(\mathbf{S}^{k-1}|\mathbf{y}^{k-1}). \tag{6}$$

The recursive Bayesian solution (6) yields the posterior conditional distribution $p(\mathbf{S}^k|\mathbf{y}^k)$ at time k, after obtaining the last measurement $\mathbf{y}_k$. Our interest is to know the sequence of objects and their states, and to know the number of objects and their states at a certain time k. Therefore, we will use the recursive Bayesian solution to derive **the state conditional density**:

$$p(\mathbf{S}_k|\mathbf{y}^k) = \int_{\mathbf{S}_{k-1}} ... \int_{S_0} p(\mathbf{S}^k|\boldsymbol{y^k})\, dS_0 ... d\mathbf{S}_{k-1}.$$

Applying (6) to the above equation gives:

$$p(\mathbf{S}_k|\mathbf{y}^k) = \frac{p(\mathbf{y}_k|\mathbf{S}^k)}{p(\mathbf{y}_k|\mathbf{y}^{k-1})} \int_{\mathbf{S}_{k-1}} ... \int_{S_0} p(\mathbf{S}_k|\mathbf{S}_{k-1}) p(\mathbf{S}^{k-1}|\mathbf{y}^{k-1})\, dS_0 ... d\mathbf{S}_{k-1},$$

where we simplify the integrals $\int_{\mathbf{S}_{k-2}} ... \int_{S_0} p(\mathbf{S}^{k-1}|\mathbf{y}^{k-1})\, dS_0 ... d\mathbf{S}_{k-2}$ to $p(\mathbf{S}_{k-1}|\mathbf{y}^{k-1})$, because of the fact that $p(\mathbf{S}^{k-1}|\mathbf{y}^{k-1}) = p(\mathbf{S}_{k-1}, \mathbf{S}^{k-2}|\mathbf{y}^{k-1})$, leading to **the state conditional density:**

$$p(\mathbf{S}_k|\mathbf{y}^k) = \frac{p(\mathbf{y}_k|\mathbf{S}_k)}{p(\mathbf{y}_k|\mathbf{y}^{k-1})} \int_{\mathbf{S}_{k-1}} p(\mathbf{S}_k|\mathbf{S}_{k-1}) p(\mathbf{S}_{k-1}|\mathbf{y}^{k-1}) d\mathbf{S}_{k-1}. \tag{7}$$

The integral $\int_{\mathbf{S}_{k-1}} p(\mathbf{S}_k|\mathbf{S}_{k-1}) p(\mathbf{S}_{k-1}|\mathbf{y}^{k-1}) d\mathbf{S}_{k-1}$ is the Chapman - Kolmogorov equation. Solving the recursive relation in (7) is the core of solving object tracking problems and most object tracking algorithms compute or attempt to approximate (7).

3 Object dynamics and measurement equations

Let $\mathbf{x}_k \in \mathbb{R}^{n_x}$ denote the object state at time k, where n_x denotes the dimensionality of the object state at time k. The object dynamics are modeled by a stohastic difference equation $\mathbf{x}_k = \mathbf{g}(x_{k-1}, \mathbf{v}_k)$, which defines the prior density for the object state, where $\mathbf{g} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_v} \to \mathbb{R}^{n_x}$ is C^2 continuous and $\mathbf{v}_k$ is a random noise input to the system. Object tracking algorithms usually assume an additive noise assumption for the object dynamics equation, leading to an equation

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{v}_k. \tag{8}$$

The inverse of the function $\mathbf{g}(\mathbf{x}_{k-1}, \mathbf{v}_k) = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{v}_k$ of the object dynamics equation is $\mathbf{g}^{-1}(\mathbf{x}_k, \mathbf{x}_{k-1}) = \mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1})$. Furthermore, the transition density function is

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}) = p_{\mathbf{v}_k}(\mathbf{g}^{-1}(\mathbf{x}_k, \mathbf{x}_{k-1}))|\nabla_{\mathbf{x}_k}\mathbf{g}^{-1}(\mathbf{x}_k, \mathbf{x}_{k-1})| = p_{\mathbf{v}_k}(\mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1})), \tag{9}$$

where $p_{\mathbf{v}_k}$ is posterior density function of $\mathbf{v}_k$.
Let $\mathbf{y}_k \in \mathbb{R}^{n_y}$ denote the observed measurement at time k, where n_y denotes the dimensionality of the observed measurement at time k. Usually we describe sensors for object tracking with sensor models of the form $\mathbf{y}_k = l(\mathbf{x}_k, \mathbf{w}_k)$, where $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_w} \to \mathbb{R}^{n_y}$ is C^2 continuous and $\mathbf{w}_k$ is a random variable modeling the measurement error. The likelihood function $p(\mathbf{y}_k|\mathbf{x}_k)$ is derived from the sensor measurement equation. If the object dynamics equation $\mathbf{g}$ and the measurement equation l are linear and the random variables $\mathbf{v}_k$ and $\mathbf{w}_k$ are Gaussian, which means they are normally distributed, the posterior density of $\mathbf{x}_k$ is Gaussian and can be found using the Kalman filter. Just as with additive noise assumption for the object dynamics equation, we have an additive assumption for the measurement noise, resaulting in an equation

$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{w}_k. \tag{10}$$

The inverse of the function $l(\mathbf{x}_k, \mathbf{w}_k) = \mathbf{h}(\mathbf{x}_k) + \mathbf{w}_k$ of the measurement equation is $l^{-1}(\mathbf{y}_k, \mathbf{x}_k) = \mathbf{y}_k - \mathbf{h}(\mathbf{x}_k)$. Furthermore, the likelihood function is

$$p(\mathbf{y}_k|\mathbf{x}_k) = p_{\mathbf{w}_k}(l^{-1}(\mathbf{y}_k, \mathbf{x}_k))|\nabla_{\mathbf{y}_k}l^{-1}(\mathbf{y}_k, \mathbf{x}_k)| = p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_k)), \tag{11}$$

where $p_{\mathbf{w}_k}$ is posterior density function of $\mathbf{w}_k$.
The transition density function $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ and the likelihood function $p(\mathbf{y}_k|\mathbf{x}_k)$ are used for the recursive estimation of the conditional density function:

$$p(\mathbf{x}_k|\mathbf{y}^k) = \frac{p(\mathbf{y}_k|\mathbf{x}_k)}{p(\mathbf{y}_k|\mathbf{y}^{k-1})} \int_{\mathbf{x}_{k-1}} p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{y}^{k-1})d\mathbf{x}_{k-1}. \tag{12}$$

Substituting the transition density (9) and the likelihood function (11) in (12), the posterior density $p(\mathbf{x}_k|\mathbf{y}^k)$ of object state is

$$p(\mathbf{x}_k|\mathbf{y}^k) = \frac{p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_k)) \int p_{\mathbf{v}_k}(\mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1}))p(\mathbf{x}_{k-1}|\mathbf{y}^{k-1})d\mathbf{x}_{k-1}}{\int p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_k))p(\mathbf{x}_k|\mathbf{y}^{k-1})d\mathbf{x}_k}, \tag{13}$$

where $p(\mathbf{x}_k|\mathbf{y}^{k-1}) = \int p_{\mathbf{v}_k}(\mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1}))p(\mathbf{x}_{k-1}|\mathbf{y}^{k-1})d\mathbf{x}_{k-1}$. The Bayesian approach to non - maneuvering object tracking is summarized in equation (13). It consists of two steps. The first step is the prediction step and the second step is the filtering step. The prediction step uses $p(\mathbf{x}_{k-1}|\mathbf{y}^{k-1})$, which is the conditional density at time $k-1$, and the Chapman-Kolmogorov equation to form the predicted density $p(\mathbf{x}_k|\mathbf{y}^{k-1})$. The predicted density contains the current information about $\mathbf{x}_k$ up to and including time $k-1$, before the new information $\mathbf{y}_k$ is included. In the filtering step, the new information $\mathbf{y}_k$ is passed through the likelihood function(11) to form the filtering distribution $p(\mathbf{x}_k|\mathbf{y}^k)$. This is the beginning

for our three filters, there are only differences in the assumptions used in the evaluation of (13). Here are assumptions we will use in deriving the Kalman filter:
i) The object dynamics and measurement equations are linear:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{v}_k, \tag{14}$$

$$\mathbf{y}_k = \mathbf{H}\mathbf{x}_k + \mathbf{w}_k. \tag{15}$$

ii) $\mathbf{v}_k$ and $\mathbf{w}_k$ are white, uncorrelated, Gaussian noise sequences with zero mean and covariance $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively.
iii) The posterior density of the object state $p(\mathbf{x}_{k-1}|\mathbf{y}^{k-1})$ at time $k-1$ is Gaussian with mean $\hat{\boldsymbol{x}}_{k-1|k-1}$ and covariance $\mathbf{P}_{k-1|k-1}$.

4 The Kalman filter

One of the first mathematicians who was developing Kalman filter was Rudolf Emil Kálmán (1930. - 2016.)[1] and the filter is named after him. He was not the only mathematician who studied Kalman filter. There was also Peter Swerling (1929. - 2000.)[2] who anticipated the development of the filter. Kalman filter was used in navigation computers in NASA Apollo program.

4.1 Simple example of the Kalman filter

Example 4.1. *Consider a train moving at a constant speed on the railway without maneuvering. In this case, the train moves in one dimension. Let x_k be its one-dimensional position and $\dot{x}_k$ its speed at time $t_k, k = 1, 2, 3...$ The state of the train is represented by the two-dimensional vector $\boldsymbol{x}_k = [x_k, \dot{x}_k]^T$. Under the assumptions in the example, we can write the position of the train x_k at time t_k in terms of position and speed at time t_{k-1}:*

$$x_k = x_{k-1} + \dot{x}_{k-1}T, \tag{16}$$

where $T = t_k - t_{k-1}$ is the interval between measurements and is assumed to be constant for all k. Assuming constant train speed we notice $\dot{x}_k = \dot{x}_{k-1}$. Therefore, the object dynamics equation is:

$$\boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_{k-1} + \boldsymbol{v}_k, \tag{17}$$

where $\boldsymbol{F} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}$ and $\boldsymbol{v}_k$ is Gaussian noise with zero mean and covariance $\boldsymbol{Q}_k$. The sensor provides measurements of the object position embedded in zero-mean additive Gaussian

[1] Hungarian-American electrical engineer and mathematician
[2] American mathematician and economist

noise at times $t_k = 1, 2, 3, ...$ *The measurement equation is:*

$$\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{w}_k, \tag{18}$$

where $\boldsymbol{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$ *and* $\boldsymbol{w}_k$ *is Gaussian noise with zero mean and covariance* $\boldsymbol{R}_k$.

We will first consider how we formed the object dynamic equation (17) and the transition matrix **F**. For this purpose we will use the train position x_k at time t_k as well as the assumption of a constant speed, $\dot{x}_k = \dot{x}_{k-1}$, for all k, which gives us the equations:

$$\begin{aligned} x_k &= x_{k-1} + \dot{x}_{k-1}T, \\ \dot{x}_k &= \dot{x}_{k-1}. \end{aligned}$$

We can also write these equations in matrix form:

$$\begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \dot{x}_{k-1} \end{bmatrix}.$$

By comparing the equations in matrix form with (14), we can see that $\mathbf{F} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}$. Although we cannot know the actual train position x_k, the Kalman filter algorithm provides an estimated train position $\hat{x}_k$ at time k. The Kalman filter is based on Gaussian probability density functions, and therefore we need to know the variances and covariances of the multivariate normal distribution or the normal distribution stored in the covariance matrix $\mathbf{P}_k$. To illustrate, the matrix $\mathbf{P}_k$ in Example 4.1 can be initialized to $\mathbf{P}_0 = \begin{bmatrix} 3^2 & 0 \\ 0 & 10^2 \end{bmatrix}$, where 3^2 is the variance of the train position, meaning that we are confident in our initial estimate of the train position for a drift of $3m$, and 10^2 is the variance of the train speed, meaning that we are not too confident in our initial estimate of the train speed for a drift of $10m/s$.

The Kalman filter algorithm consists of two steps, prediction and measurement update. The filter equations for the prediction step are:

$$\begin{aligned} \hat{\boldsymbol{x}}_{k|k-1} &= \mathbf{F}_t \hat{\boldsymbol{x}}_{k-1|k-1}, \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^t + \mathbf{Q}_k, \end{aligned}$$

where $\mathbf{Q}_k$ is the process noise covariance matrix. The filter equations for the measurement update step are:

$$\begin{aligned} \hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k \hat{\boldsymbol{x}}_{k|k-1}), \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_{k|k-1}, \end{aligned}$$

where $\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^t(\mathbf{H}_k \mathbf{P}_{k|k-1}\mathbf{H}_k^t + \mathbf{R}_k)^{-1}$ is called the Kalman gain.

For each measurement obtained, we want to know the best estimate of the train position. We obtain information from predictions based on the last known position and speed of the train, and from measurements. To achieve the best approximation, we use both prediction

based on last known position and measurements together. Consider this in the following example. At time $k = 0$, we have an estimated state where the train position is given by a Gaussian probability density function. At the next time $k = 1$, we can estimate the new train position based on the train position and speed at time $k = 0$. The train position at time $k = 1$ is represented by a new Gaussian probability density function with new mean and variance. We also take a measurement of train position at time $k = 1$, which is also represented by another Gaussian probability density function. Now we use our knowledge from the prediction and the measurement for the best estimate of the train position. Mathematically, this means that we multiply the two corresponding Gaussian probability density functions together. In doing so, we take advantage of the fact that the product of two Gaussian densities is a Gaussian demsity and the product represents our best estimate of the train position.

In implementing the Example 4.1, we will specify the initial starting point and the covariance matrix, but we will discuss this in more detail in the final chapter.

4.2 Derivation of the Kalman filter

In deriving the Kalman filter we must use all three assumptions **i)**, **ii)**, **iii)**. We will now use the following theorem, known as the Gaussian product, to derive the Kalman filter [1, Theorem 2.1, 27. page].

Theorem 2. *For* $\boldsymbol{x}_1, \boldsymbol{\mu}_1 \in \mathbb{R}^{d_1}, \boldsymbol{H} \in \mathbb{R}^{d_2 \times d_1}, \boldsymbol{x}_2 \in \mathbb{R}^{d_2}$, *postive definite matrices* $\boldsymbol{P}_1, \boldsymbol{P}_2$, $X_1 \sim N(\boldsymbol{H}\boldsymbol{x}_1, \boldsymbol{P}_2), X_2 \sim N(\boldsymbol{\mu}_1, \boldsymbol{P}_1), X_3 \sim N(\boldsymbol{H}\boldsymbol{\mu}_1, \boldsymbol{P}_3)$ *and* $X_4 \sim N(\boldsymbol{\mu}, \boldsymbol{P})$

$$f_{X_1}(\boldsymbol{x}_2) \cdot f_{X_2}(\boldsymbol{x}_1) = f_{X_3}(\boldsymbol{x}_2) \cdot f_{X_4}(\boldsymbol{x}_1), \tag{19}$$

where $f_{X_1}, f_{X_2}, f_{X_3}, f_{X_4}$ *are probability density functions of* X_1, X_2, X_3 *and* X_4, *respectively and* $\boldsymbol{P}_3 = \boldsymbol{H}\boldsymbol{P}_1\boldsymbol{H}^T + \boldsymbol{P}_2$, $\boldsymbol{\mu} = \boldsymbol{\mu}_1 + \boldsymbol{K}(\boldsymbol{x}_2 - \boldsymbol{H}\boldsymbol{\mu}_1)$, $\boldsymbol{P} = \boldsymbol{P}_1 + \boldsymbol{K}\boldsymbol{H}\boldsymbol{P}_1$, $\boldsymbol{K} = \boldsymbol{P}_1\boldsymbol{H}^T\boldsymbol{P}_3^{-1}$.

Using (9), the transition density is:

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}) = f_{X_1}(\mathbf{x}_k), \tag{20}$$

where $s_{X_1}(\mathbf{x}_k)$ is probability density function of $X_1 \sim N(\mathbf{F}\mathbf{x}_{k-1}, \mathbf{Q}_k)$ and $\mathbf{Q}_k$ is covariance matrix. The posterior density of the object state at time $k-1$ is Gaussian so that $p(\mathbf{x}_{k-1}|\mathbf{y}^{k-1}) = f_{X_2}(\mathbf{x}_k)$, $f_{X_2}(\mathbf{x}_k)$ is probability density function of
$X_2 \sim N(\hat{\boldsymbol{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1})$. The predicted density is

$$p(\mathbf{x}_k|\mathbf{y}^{k-1}) = \int f_{X_1}(\mathbf{x}_k) f_{X_2}(\mathbf{x}_k) d\mathbf{x}_{k-1}. \tag{21}$$

Applying Theorem (2) to (21) gives

$$p(\mathbf{x}_k|\mathbf{y}^{k-1}) = f_{X_2}(\mathbf{x}_k), \tag{22}$$

where $\hat{\boldsymbol{x}}_{k|k-1} = \mathbf{F}\hat{\boldsymbol{x}}_{k-1|k-1}$ and $\mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^T + \mathbf{Q}_k$. The prediction density is Gaussian if the posterior density at time $k-1$ is Gaussian and the object dynamic equation

is linear/Gaussian which is shown in (22). This is the standard Kalman filter prediction and its pseudofunction is:

$$[\hat{\boldsymbol{x}}_{k|k-1}, \mathbf{P}_{k|k-1}] = KF_P[\hat{\boldsymbol{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}, \mathbf{F}, \mathbf{Q}], \tag{23}$$

where $\hat{\boldsymbol{x}}_{k|k-1}$ and $\mathbf{P}_{k|k-1}$ are the same as in (22).
Let's see what it would be the likelihood function and normalization factor. The likelihood function is $p(\mathbf{y}_k|\mathbf{x}_k) = f_{Y_1}(\mathbf{y}_k)$, where we use (11), fact that $p_{\mathbf{w}_k}$ is a Gaussian density with zero mean and covariance $\mathbf{R}_k$ and $f_{Y_1}(\mathbf{y}_k)$ is probability density function of $Y_1 \sim N(\mathbf{H}\mathbf{x}_k, \mathbf{R}_k)$. We can found the normalization factor using Theorem 2:

$$p(\mathbf{y}_k|\mathbf{y}^{k-1}) = f_{Y_2}(\mathbf{y}_k), \tag{24}$$

where $f_{Y_2}(\mathbf{y}_k)$ is probability density function of $Y_2 \sim N(\hat{\boldsymbol{y}}_{k|k-1}, \mathbf{S}_k)$, $\hat{\boldsymbol{y}}_{k|k-1} = \mathbf{H}\hat{\boldsymbol{x}}_{k|k-1}$ and $\mathbf{S}_k = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R}_k$. With this operation we get the mean and covariance of the object measurement predicted probability density function and its pseudofunction is:

$$[\hat{\boldsymbol{y}}_{k|k-1}, \mathbf{S}_{k|k-1}] = MP[\hat{\boldsymbol{x}}_{k|k-1}, \mathbf{P}_{k|k-1}, \mathbf{H}, \mathbf{R}], \tag{25}$$

defined by $\hat{\boldsymbol{y}}_{k|k-1} = \mathbf{H}\hat{\boldsymbol{x}}_{k|k-1}$ and $\mathbf{S}_k = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R}$.
Using 12, the conditional density function can be found as:

$$p(\mathbf{x}_k|\mathbf{y}^k) = \frac{f_{Y_1}(\mathbf{y}_k) f_{X_2}(\mathbf{x}_k)}{f_{Y_2}(\mathbf{y}_k)}. \tag{26}$$

Applying Theorem 2 on 26 gives:

$$p(\mathbf{x}_k|\mathbf{y}^k) = f_{X_3}(\mathbf{x}_k), \tag{27}$$

where

$$\begin{gathered} f_{X_3}\, is\, probability\, density\, function\, of\, X_3 \sim N(\hat{\boldsymbol{x}}_{k|k}, \mathbf{P}_{k|k}), \\ \hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1}(\mathbf{y}_k - \hat{\boldsymbol{y}}_{k|k-1}), \\ \mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1}\mathbf{H}\mathbf{P}_{k|k-1}. \end{gathered}$$

Final step is to define the Kalman filter estimation pseudo-function:

$$[\hat{\boldsymbol{x}}_{k|k}, \mathbf{P}_{k|k}] = KF_E[\mathbf{y}, \hat{\boldsymbol{x}}_{k|k-1}, \mathbf{P}_{k|k-1}, \mathbf{H}, \mathbf{R}], \tag{28}$$

defined by:

$$[\hat{\boldsymbol{y}}_{k|k-1}, \mathbf{S}_k] = MP[\hat{\boldsymbol{x}}_{k|k-1}, \mathbf{P}_{k|k-1}, \mathbf{H}, \mathbf{R}],$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1},$$

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y} - \hat{\boldsymbol{y}}_{k|k-1}),$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_{k|k-1}.$$

We can now list the three steps of the Kalman filter [1]:
1. Compute the predicted mean an covariance matrix:

$$\hat{\boldsymbol{x}}_{k|k-1} = \mathbf{F}\hat{\boldsymbol{x}}_{k-1|k-1},$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^T + \mathbf{Q}_k.$$

2. Compute the predicted measurement, innovation covariance matrix and Kalman gain:

$$\hat{\boldsymbol{y}}_{k|k-1} = \mathbf{H}\hat{\boldsymbol{x}}_{k|k-1},$$

$$\mathbf{S}_k = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R}_k,$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1}.$$

3. Compute the posterior mean and covariance matrix:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y} - \hat{\boldsymbol{y}}_{k|k-1}),$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_{k|k-1}.$$

4.3 Advantages and disadvantages

If all three assumptions of the Kalman filter are satisfied, we can obtain a very good estimate despite the uncertainty due to noisy sensor data. Moreover, the Kalman filter recursively computes the posterior mean and covariance matrix as measurements are acquired. It is this recursive structure that allows real-time execution without storing observations or past estimates.
The main drawback of the Kalman filter is that it is only applicable when three assumptions are met. However, the assumption of linear dynamic and measurement equations is unsatisfied in most tracking problems. For this reason, we have an algorithm which can be derived only under assumptions **ii.)** and **iii.)**, which is called the extended Kalman filter. We also have an alternative algorithm to the extended Kalman filter, the unscented Kalman filter. The unscented Kalman filter has the same computational complexity as the extended Kalman filter, but performs better in most cases.

5 The Point mass filter

Unlike the Kalman filter, we do not need assumptions for the Kalman filter to derive the point mass filter. To use the point mass filter, we need advanced or high-tech computers to achieve higher accuracy in the numerical approximation of the posterior probability density function.

At time $k-1$, a region of state space is partitioned into n hyper-cubes of equal volume. Let $\mathbf{x}_{k-1}^i$ denote the center of the cube $i, i \in \{1,2,...,n\}$. Each hyper-cube is associated with a weight $w_{k-1|k-1}^i, i \in \{1,2,...n\}$, where $w_{k-1|k-1}^i$ sums up to one as i goes from 0 to n. Hyper-cubes and weights together form a discrete approximation to the posterior probability density function at time $k-1$:

$$p(\mathbf{x}_{k-1}|\mathbf{y}^{k-1}) \approx \sum_{i=1}^{n} w_{k-1|k-1}^i \delta(\mathbf{x}_{k-1} - \mathbf{x}_{k-1}^i). \tag{29}$$

Let denote the object dynamics wtih:

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{v}_k, \tag{30}$$

where $\mathbf{v}_k$ is additive noise. The transition density is $p(\mathbf{x}_k|\mathbf{x}_{k-1}) = p_{\mathbf{v}_k}(\mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1}))$. Using transition density and the Chapman-Kolmogorov equation we can found the prediciton density:

$$p(\mathbf{x}_k|\mathbf{y}^{k-1}) = \int p_{\mathbf{v}_k}(\mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1}))p(\mathbf{x}_{k-1}|\mathbf{y}^{k-1})d\mathbf{x}_{k-1}. \tag{31}$$

Substituting (29) into (31) gives:

$$p(\mathbf{x}_k|\mathbf{y}^{k-1}) \approx \sum_{i=1}^{n} w_{k-1|k-1}^i p_{\mathbf{v}_k}(\mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1})). \tag{32}$$

A finite-dimensional representation of the prediction density is obtained by partitioning a region of the state space into n hyper-cubes of equal volume. It is not necessary to use the same number of points at each time, although this is done here for notational convenience. Let $\mathbf{x}_k^i$ denote the center of the i th hyper-cube for $i = 1,...,n$. Then the point mass filter approximation to the prediction density is:

$$p(\mathbf{x}_k|\mathbf{y}^{k-1}) \approx \sum_{i=1}^{n} w_{k|k-1}^i \delta(\mathbf{x}_k - \mathbf{x}_k^i), \tag{33}$$

where

$$w_{k|k-1}^i = \sum_{j=1}^{n} w_{k-1|k-1}^j p_{\mathbf{v}_k}(\mathbf{x}_k^i - \mathbf{f}(\mathbf{x}_{k-1}^j)), i = 1,...,n. \tag{34}$$

Measurement equation has the same form as (10), where $\mathbf{w}_k$ is additive noise and the likelihood function of $\mathbf{x}_k$ has the same form as (11). We can expand the normalization factor $p(\mathbf{y}_k|\mathbf{y}^{k-1})$ as:

$$p(\mathbf{y}_k|\mathbf{y}^{k-1}) = \int p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_k))p(\mathbf{x}_k|\mathbf{y}^{k-1})d\mathbf{x}_k. \tag{35}$$

Using (33) in (35) gives the point mass filter approximation to the normalizing factor:

$$p(\mathbf{y}_k|\mathbf{y}^{k-1}) \approx \sum_{i=1}^{n} w_{k|k-1}^i p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_{k-1}^i)). \tag{36}$$

Now we can compute the point mass filter approximation to the posterior probability density function at time k using (33), the likelihood function and (36) in (13):

$$p(\mathbf{x}_k|\mathbf{y}^k) \approx \sum_{i=1}^{n} w_{k|k}^i \delta(\mathbf{x}_k - \mathbf{x}_k^i), \tag{37}$$

where

$$w_{k|k}^i = w_{k|k-1}^i p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_{k-1}^i)) / \sum_{j=1}^{n} w_{k|k-1}^j p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_k^j)), \tag{38}$$

where $i = 1, ..., n$.
In the end we can obtain a point estimate of the state as:

$$\hat{\mathbf{x}}_{k|k} = \sum_{i=1}^{n} w_{k|k}^i \mathbf{x}_k^i. \tag{39}$$

Based on this analysis we list the point mass filter algorithm [1]:
1. Select grid points:

$$\mathbf{x}_k^1, ..., \mathbf{x}_k^n.$$

2. Compute the weights for prediction density:

$$w_{k|k-1}^i = \sum_{j=1}^{n} w_{k-1|k-1}^j p_{\mathbf{v}_k}(\mathbf{x}_k^i - \mathbf{f}(\mathbf{x}_{k-1}^j)), i = 1, ..., n.$$

3. Compute the weights for posterior probability density function at time k, $i = 1, ..., n$:

$$w_{k|k}^i = w_{k|k-1}^i p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_{k-1}^i)) / \sum_{j=1}^{n} w_{k|k-1}^j p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_k^j)).$$

4. Compute a state estimate:

$$\hat{\mathbf{x}}_{k|k} = \sum_{i=1}^{n} w_{k|k}^i \mathbf{x}_k^i.$$

5.1 Advantages and disadvantages

Unlike the Kalman filter, the point mass filter does not require any of the three assumptions. This allows its application to nonlinear dynamic and measurement equations. The point mass filter achieves higher accuracy by obtaining a discrete approximation of the posterior probability density function. Achieving higher accuracy in this way requires significant

computer resources, which is the main drawback of the point mass filter. The second step of the point mass filter algorithm computes the weights of the prediction probability density function which requires n^2 operations. Kramer and Sorenson reduced the complexity in the second step by using the fast Fourier transform, which results in $\mathcal{O}(nlog(n))$ operations. Another computational problem is that the number of grid points required for a certain level of accuracy increases exponentially with the dimension of the state, leading to a relative approximation error of $\mathcal{O}(n^{-1/n_x})$. In addition to computational complexity, another problem is the selection of the grid points for the approximation of the posterior probability density function. The grid must include all regions of interest and exclude the regions of no interest. The problem is that we do not know doubtless which regions are of interest.

6 The particle filter

The term particle filter was first used in $1996.$ by Del Moral, who was studying fluid mechanics. Particle filtering uses a set of samples (particles) to represent the posterior distribution of the stochastic process, given noisy and/or partial observations. To use the particle filter, we do not need a linear state-space model and the noise distributions do not necessarily have to be normal. Therefore, it is superior to the Kalman filter. Compared to the point mass filter, the particle filter is easier to implement because it is no longer necessary to establish rules for determining the grid points. Also, the computational complexity is lower since the error convergence does not depend on the dimension of the state as in the point mass filter. Since the basic particle filter is formulated with sequential importance sampling, it is discussed in the following subsection.

6.1 Importance sampling and sequential importance sampling

Importance sampling is a Monte Carlo method that forms a sample-based approximation. Samples are taken from the proposed distribution. The approximation uses weights to ensure the best result. We will compute the expectation of the generic function h_t, which we will consider as a test function with respect to the target distribution γ_t:

$$\gamma_t(h_t) := \mathbb{E}_{\gamma_t}[h_t(x_{1:t})]. \tag{40}$$

To make the calculation easier, we will rewrite the equation (40) as:

$$\mathbb{E}_{\gamma_t}[h_t(x_{1:t})] = \frac{1}{Z_t}\mathbb{E}_{q_t}\left[\frac{\tilde{\gamma}_t(x_{1:t})}{q_t(x_{1:t})}h_t(x_{1:t})\right] = \frac{\mathbb{E}_{q_t}\left[\frac{\tilde{\gamma}_t(x_{1:t})}{q_t(x_{1:t})}h_t(x_{1:t})\right]}{\mathbb{E}_{q_t}\left[\frac{\tilde{\gamma}_t(x_{1:t})}{q_t(x_{1:t})}\right]}, \tag{41}$$

where Z_t is normalization constant, $\tilde{\gamma}_t$ unnormalized target distribution, and q_t is the proposal distribution. Now we will estimate right-hand side of (41) using the Monte Carlo

method:

$$\mathbb{E}_{\gamma_t}[h_t(x_{1:t})] \approx \frac{\frac{1}{N}\sum_{i=1}^{N} \tilde{w}_t(x_{1:t}^i)h_t(x_{1:t}^i))}{\frac{1}{N}\sum_{j=1}^{N} \tilde{w}_t(x_{1:t}^j)}, \tag{42}$$

where $\tilde{w}_t(x_{1:t}) = \dfrac{\tilde{\gamma}_t(x_{1:t})}{q_t(x_{1:t})}$, $x_{1:t}^i$ are simulated independent and identically distributed random variables (iid) from q_t, and N is the number of the samples. Estimation (42) is usually written more compactly as:

$$\mathbb{E}_{\gamma_t}[h_t(x_{1:t})] \approx \sum_{i=1}^{N} w_t^i h_t(x_{1:t}^i), \qquad x_{1:t}^i \overset{idd}{\sim} q_t, \tag{43}$$

where the normalized weights w_t^i are defined by

$$w_t^i := \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j},$$

where $\tilde{w}_t^i$ is a shorter note for $\tilde{w}_t(x_{1:t}^i)$. As the number of samples N tends to infinity, the estimate (43) converges to the true expectation. On the other hand, we can consider importance sampling as an approximation of γ_t:

$$\gamma_t(x_{1:t}) \approx \sum_{i=1}^{N} w_t^i \delta_{x_{1:t}^i}(x_{1:t}) =: \hat{\gamma}_t(x_{1:t}),$$

where δ_X denotes the Dirac measure at X. Importance sampling also provides an approximation to the normalization constant Z_t:

$$Z_t \approx \frac{1}{N}\sum_{i=1}^{N} \tilde{w}_t^i =: \hat{Z}_t. \tag{44}$$

The weights are random variables due to the dependence on the random samples $x_{1:t}^i$. The normalization constant Z_t is unbiased, as can be seen from the fact that $x_{1:t}^i$ are iid draws from q_t and accordingly:

$$\mathbb{E}[\hat{Z}_t] = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}\left[\frac{\tilde{\gamma}_t(x_{1:t}^i)}{q_t(x_{1:t}^i)}\right] = \frac{1}{N}\sum_{i=1}^{N} \int \frac{\tilde{\gamma}_t(x_{1:t}^i)}{q_t(x_{1:t}^i)} q_t(x_{1:t}^i) dx_{1:t}^i = Z_t, \tag{45}$$

since $Z_t = \int \tilde{\gamma}_t(x_{1:t}) dx_{1:t}$. Now we will sumarized the importance sampling method in algorithm [7].

The main drawback of importance sampling is that it is hard to choose a good proposal for multidimensional models. When we do find a satisfactory proposal,it is usually more heavy-tailed than the target. We will solve these shortcomings by upgrading the importance sampling, which is called sequential importance sampling. Sequential importance

Algorithm 1: The importance sampling

Input: Unnormalized target distribution $\tilde{\gamma}_t$, proposal q_t and number of samples N

for *i=1,...,n* **do**

sample $x_{1:t}^i \sim q_t$

set $\tilde{w}_t^i = \dfrac{\tilde{\gamma}_t(x_{1:t}^i)}{q_t(x_{1:t}^i)}$

set $w_t^i = \dfrac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}, \quad for \quad i = 1, ..., N$

Result: samples and weights $(x_{1:t}^i, w_t^i)_{i=1}^N$ approximating γ_t

sampling uses a proposal distribution that has an autoregressive structure and the weights are computed recursively. If we select a proposal defined by:

$$q_t(x_{1:t}) = q_{t-1}(x_{1:t-1})q_t(x_t|x_{1:t-1}),$$

we can decompose the proposal problem into T conditional distributions. This allows us to get samples $x_{1:t}^i$ by reusing $x_{1:t-1}^i$ and append a new sample x_t^i which is simulated from $q_t(x_t|x_{1:t-1}^i)$. The unnormalized weights are:

$$\tilde{w}_t(x_{1:t}) = \tilde{w}_{t-1}(x_{1:t-1})\frac{\tilde{\gamma}_t(x_{1:t})}{\tilde{\gamma}_{t-1}(x_{1:t-1})q_t(x_t|x_{1:t-1})}. \tag{46}$$

The weights in (46) are computed recursively.
The sequential importance sampling is summarized in the following algorithm [7] in which we state $q_1(x_1|x_{1:0}) = q_1(x_1)$ and $\tilde{w}_0 = \tilde{\gamma}_0 = 1$.

Algorithm 2: The sequential importance sampling

Input: Unnormalized target distributions $\tilde{\gamma}_t$, proposal q_t, number of samples N

for *t=1,...,T* **do**

for *i=1,...,n* **do**

sample $x_t^i \sim q_t(x_t|x_{1:t-1}^i)$

append $x_{1:t}^i = (x_{1:t-1}^i, x_t^i)$

set $\tilde{w}_t^i = \tilde{w}_{t-1}^i \dfrac{\tilde{\gamma}_t(x_{1:t}^i)}{\tilde{\gamma}_{t-1}(x_{1:t-1}^i)q_t(x_t^i|x_{1:t-1}^i)}$

set $w_t^i = \dfrac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^i}$

Result: Samples and weights $(x_{1:t}^i, w_t^i)_{i=1}^N$ approximatinh γ_t, for $t = 1, ..., T$

We can get the estimate of the ratio of normalization constants as follows:

$$\frac{Z_t}{Z_{t-1}} = \mathbb{E}_{\gamma_t(x_{1:t-1})q_t(x_t|x_{1:t-1})}\left[\frac{\tilde{\gamma}_t(x_{1:t})}{\tilde{\gamma}_{t-1}(x_{1:t-1})q_t(x_t|x_{1:t-1})}\right] \approx \sum_{i=1}^{N} w_{t-1}^i \frac{\tilde{w}_t^i}{\tilde{w}_{t-1}^i}$$

Although the estimate of the ratio is consistent, generally it is unbiased.

6.2 The basic particle filter

At the beginning, we state the basic particle filter alogrithm [1] formulated using the sequential importance sampling method.

Algorithm 3: The basic particle filter

for *i=1,..,n* **do**

Draw samples$(\mathbf{x}_k^i, t^i) \sim q$

Compute the weight update factor:

$$e_k^i = \frac{p(\mathbf{y}_k|\mathbf{x}_k, \mathbf{y}_{1:k-1})p(\mathbf{x}_k|\mathbf{x}_{k-1}^{t^i}, \mathbf{y}_{1;k-1})}{q(\mathbf{x}_k^i, t^i)}$$

Compute the updated weights:

$$\omega_k^i = \frac{\omega_{k-1}^i e_k^i}{\sum_{j=1}^n \omega_{k-1}^j e_k^j}, \qquad i = 1, ..., n.$$

This algorithm generates weights (ω_k^i) and samples ($\mathbf{x}_k^i$) representing the posterior probability density function at time k. It is often sufficient to know only the sample and weights at the previous time $k-1$ to compute the sample and weights at time k. In order to use the particle filter for tracking problems, we need to carefully choose and derive the importance density q. To improve these poor performances, we need to increase the sample size, which is often not possible. In this case we need to choose a more suitable importance density q. In this master thesis, we will consider two different versions of the importance density q for the problem of single-object tracking. We will use dynamic and measurement equations that include additive noise:

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{v}_k, \tag{47}$$

$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{w}_k. \tag{48}$$

6.3 The bootstrap filter

The importance density function q for the bootstrap filter is:

$$q(\mathbf{x}_k, t) = \omega_{k-1}^t p_{\mathbf{v}_k}(\mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1}^t)). \tag{49}$$

The main advantages of the bootstrap filter are relatively simple implementation and broad applicability. The disadvantage of the bootstrap filter is that the samples of the mixture index and the object state are drawn without considering the current measurement, since the current measurement is only used to assess the quality of the samples. Therefore, to achieve a precise approximation, we need a large number of samples.
We now list the algorithm for the bootstrap filter for single-object tracking [1] in which we will see the use of importance density (49).

Algorithm 4: The bootstrap filter

for *i=1,...,n* **do**
- Draw a mixture index t^i such that $\mathbb{P}(t^i = l) = \omega_{k-1}^l$
- Draw $\mathbf{v}_k^i \sim p_{\mathbf{v}_k}$ and compute the sample object state $\mathbf{x}_k^i = \mathbf{f}(\mathbf{x}_{k-1}^{t^i}) + \mathbf{v}_k^i$
- Compute the wight update $e_k^i = p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_k^i))$

Compute the updated weights:

$$\omega_k^i = \omega_{k-1}^i e_k^i / \sum_{j=1}^{n} \omega_{k-1}^j e_k^j, \qquad i = 1, ..., n.$$

Compute a state estimate:

$$\hat{\boldsymbol{x}}_{k|k} = \sum_{j=1}^{n} \omega_k^i \mathbf{x}_k^i.$$

6.4 The auxiliary bootstrap filter

The importance density function q for the auxiliary bootstrap filter is:

$$q(\mathbf{x}_k, t) = \xi_k^t \, p_{\mathbf{v}_k}(\mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1}^t)), \tag{50}$$

where

$$\xi_k^t = \omega_{k-1}^t p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\boldsymbol{\mu}_k^t)) \, / \, \sum_{s=1}^{n} \omega_{k-1}^s p_{\mathbf{w}_{\rho_k^s}}(\mathbf{y}_k - \mathbf{h}_{\rho_k^s}(\boldsymbol{\mu}_k^{\boldsymbol{s}})), \tag{51}$$

where $\boldsymbol{\mu}_k^t = \mathbf{f}(\mathbf{x}_{k-1}^t) + \mathbf{v}_k^t$, $\mathbf{v}_k^t \sim p_{\mathbf{v}_k}$.
The auxiliary bootstrap filter is better performing and applicable to all samples for which we use the bootstrap filter. This gives us better approximation accuracy combined with a slightly more complicated implementation. The following algorithm represents a recursion of the auxiliary bootstrap filter [1].

7 Implementation of Kalman filter in R

We will use the programming language R [5] to implement the Kalman filter. R is mainly used for statistical computing and graphics. It is similar to the language and environment

Algorithm 5: The auxiliary bootstrap filter

for *i=1,...,n* **do**

Draw $\tilde{\boldsymbol{v}}_k^i \sim p_{\mathbf{v}_k}$ and compute $\boldsymbol{\mu}_{\boldsymbol{k}}^{\boldsymbol{i}} = \mathbf{f}(\mathbf{x}_{k-1}^i) + \tilde{\boldsymbol{v}}_k^i$

Compute the first stage weight update $a_k^i = p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\boldsymbol{\mu}_k^i))$

Compute thefist stage weights:

$$\xi_k^t = \omega_{k-1}^t a_k^t \,/\, \sum_{i=1}^{n} \omega_{k-1}^i a_k^i, \qquad t = 1, ..., n$$

for *i=1,...,n* **do**

Draw a mixture index t^i such that $Pr(t^i = l) = \xi_k^l$

Draw $\mathbf{v}_k^i \sim p_{\mathbf{v}_k}$ and compute the sample object state $\mathbf{x}_k^i \sim \mathbf{f}(\mathbf{x}_{k-1^{t^i}}) + \mathbf{v}_k^i$

Compute the un-normalized weight:

$$\tilde{\omega}_k^i = \frac{p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\mathbf{x}_k^i))}{p_{\mathbf{w}_k}(\mathbf{y}_k - \mathbf{h}(\boldsymbol{\mu}_k^{t^i}))}$$

Normalize the weights:

$$\omega_k^i = \tilde{\omega}_k^i \,/\, \sum_{j=1}^{n} \tilde{\omega}_k^j, \qquad i = 1, ..., n$$

Compute a state estimate:

$$\hat{\boldsymbol{x}}_{k|k} = \sum_{i=1}^{n} \omega_k^i \mathbf{x}_k^i.$$

of S and most of the codes written for S run in R. One of the main advantages of R is the ease with which plots can be created and the variety of mathematical symbols or formulas. We have implemented Example 4.1 in R, which code is in Appendix A, where the Kalman filter is implemented by definition. Such an implementation has the drawback that if the process noise covariance Q has values close to zero, a rounding error may cause a small positive eigenvalues are computed as a negative number, in which case the covariance matrix P becomes an indefinite matrix, contradicting the statement that P is positive definite. The shortcoming can be solved by a different implementation of the matrix P [3, 2]. Since P is a positive definite matrix, it can be written as $P = LL^t$, where L is a lower triangular matrix with real and positive elements on the diagonals. Such a decomposition of a matrix is called a Cholesky decomposition. With the Cholesky decomposition, we have ensured that the matrix P is always positive definite. The Cholesky decomposition can be computationally expansive because it involves extracting square roots, so the LDL decomposition is more commonly used. The LDL decomposition avoids extracting square roots. Applying the LDL decomposition to the covariance matrix P gives $P = CDC^t$, where C is a lower unit triangular matrix and D is a diagonal matrix.

Consider again Example 4.1, where the sensor is placed on the railway and measures the distance between the sensor and the train assuming that the train speed is constant. The start position is the distance from where the sensor starts the measurement and we have placed it at 500 m, the train speed is set to 50 m/s. Also the time between measurements is set to $T = 0.1s$ and assumed to be constant. The Kalman filter is initialized to $600m$ and $-65m/s$ since the vector x_k is two dimensional. The origin of the coordinate system is located in the sensor as shown in Figure 1, so the vector x is in the same direction as the train speed vector, but has an opposite orientation so that the train speed is negative.

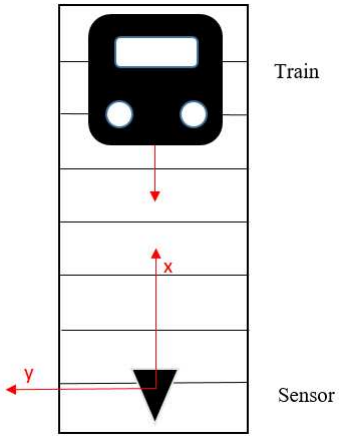

Figure 1: Position of the train and sensor

Furthermore, the covariance matrices P, process noise Q, measurement noise R and the sensor noise are tuned. By tuning the matrix R we try to approximate the sensor noise. In the following examples, we know the exact sensor noise because we have tuned it, but in the tracking problems, we do not know the exact sensor noise. In the following examples, we will tune the sensor noise and the measurement noise and observe how they affect the estimation of position and speed.

Example 7.1. *In this example we will set covariance matrix* P *on* $P = \begin{bmatrix} 3^2 & 0 \\ 0 & 10^2 \end{bmatrix}$, *process noise on* $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, *measurement noise on* $R = [1]$, *and sensor noise on random normal variable with* $mean = 0$ *and standard deviation* $sd = 1$.

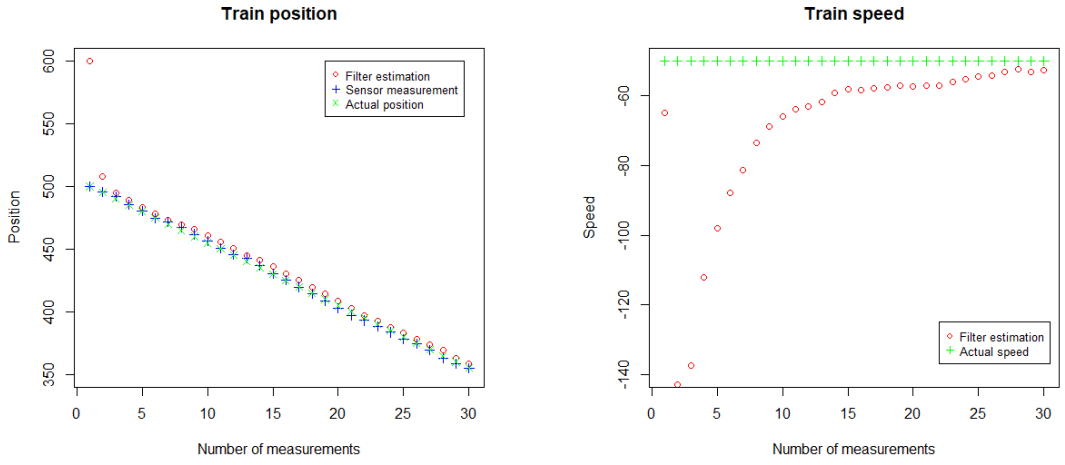

Figure 2: Position and speed of the train in the Example 7.1

In Example 7.1, we get a very good estimate of the train position due to the high-quality sensor. In the first few measurements, there was a big difference between the estimated speed and the actual speed because the initialization of speed and distance was inaccurate, but after the seventeenth measurement, we get a very accurate speed. In the initial measurements, we have a large difference in the train speed because the filter is set to strongly trust the measurements and after receiving the next measurement it tries to quickly converge to the measured train position. The following example tests what would happen to the filter estimate if we only change the sensor noise.

Example 7.2. *Let all parameters be set as in the example 7.1 except sensor noise in which we will set* $mean = 1$ *and* $sd = 1$.

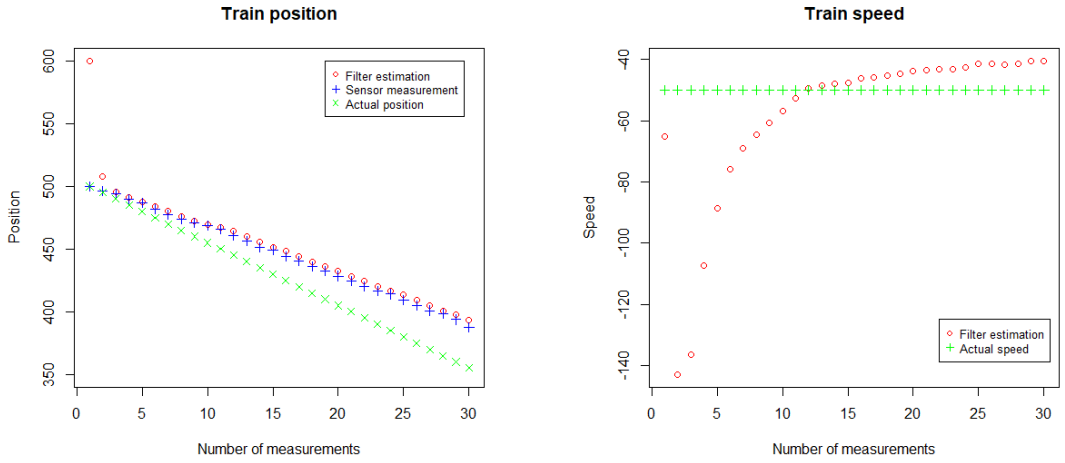

Figure 3: Position and speed of the train in the Example 7.2

Example 7.2 shows that if we had a well-tuned filter as in Example 7.1 and change the $mean$ from 0 to 1, which means that we have a bias in the measurement, the result is a worse estimate. This means that, the estimated position is different from the actual position at the end of the measurement, and instead of convergence, divergence occurs. Divergence occurs because we have a bias in the measurements and the standard deviation is set to 1, which means that the filter trusts the measurements strongly. This confirms that we are evaluating speed below the actual speed, which means that we are deviating more from the actual train position with each measurements. Looking again at Example 7.1, we can ask what happens to the change in standard deviation, and we will consider this in the third example.

Example 7.3. *Let all parameters be set as in the example 7.1 except sensor noise in which we will set* $mean = 0$, $sd = 5$ *and* $R = [25]$.

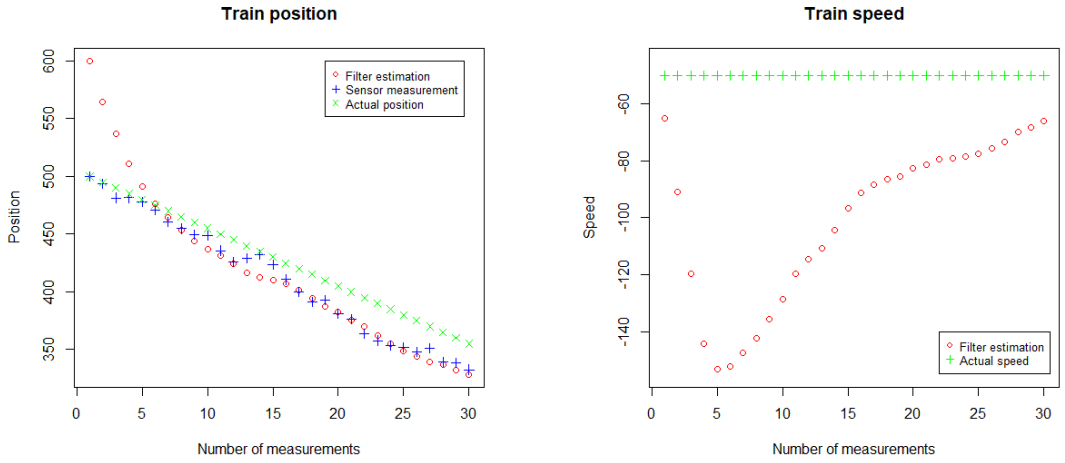

Figure 4: Position and speed of the train in the Example 7.3

We notice slower filter adaptation especially in speed estimation where we did not reach an actual speed of $-50m/s$ in thirty measurements. Let us consider an example similar to Example 7.1, but with a different measurement noise R.

Example 7.4. *Let all parameters be set as in the example 7.1 except measurement noise* $R = [10]$.

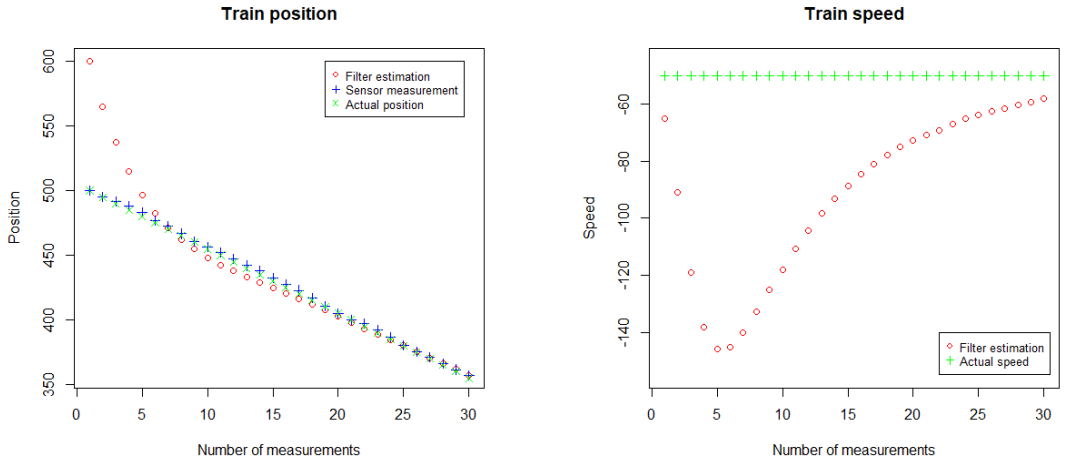

Figure 5: Position and speed of the train in the Example 7.4

Compared to Example 7.1, we notice a slower convergence in the train position, but in the end we get an accurate estimate. The train speed also converges slower to $-50m/s$ compared to the train speed in Example 7.1. Example 7.4 shows that we get a slower filter fit when we have high-quality sensor but do not really believe the measurement ($R = [10]$).

In the following example we will look at what happens to the filter when we change $sd = 5$ and $R = [20]$ in Example 7.2.

Example 7.5. *Let all parameters be set as in the example 7.2 except measurement noise* $R = [20]$ *and standard deviation* $sd = 5$.

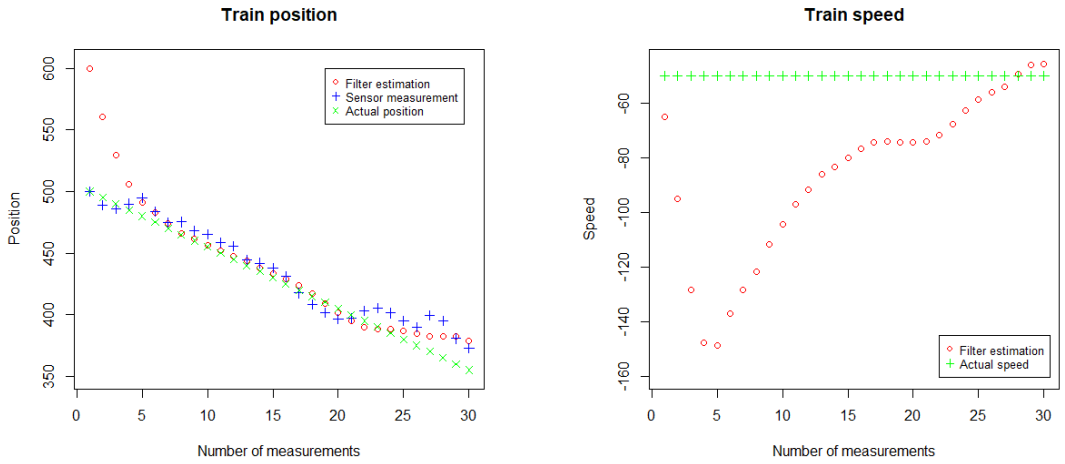

Figure 6: Position and speed of the train in the Example 7.5

Compared to Example 7.2, we notice that in Example 7.5 we have a better estimate of the train position and there is no divergence as in Example 7.1. Also in Example 7.5, we have a better speed convergence, but still do not achieve the actual value $-50m/s$. Let's look at an example similar to Example 7.3, except that we set the measurement noise to $R = [5]$.

Example 7.6. *Let all parameters be set as in the example 7.3 except measurement noise,* $R = [5]$.

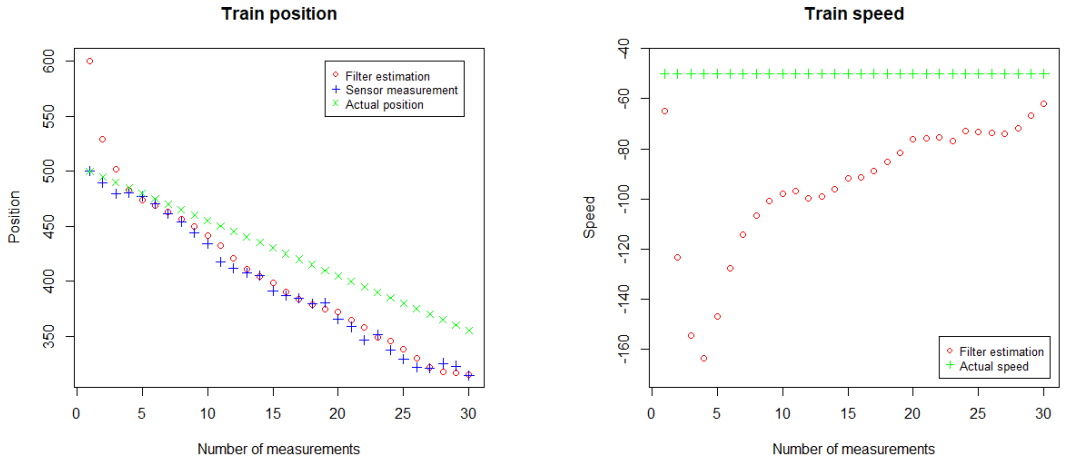

Figure 7: Position and speed of the train in the Example 7.6

Compared to Example 7.3, we notice that in Example 7.3 we have a better estimate of the train position, but even this estimate is not good compared to the actual train position. The estimated train speed is also inaccurate in both examples.

7.1 Differences in filter estimation of position and speed and actual position and speed

We will run the code found in Appendix A 1000 times and calculate the absolute difference between the filter estimate of the train position and the actual train position, and the absolute difference between the filter estimate of the train speed and the actual train speed. This will generate 30000 differences for the train position and speed, as the algorithm generates 30 measurements each time. To get more accurate results, the first five measurements are not considered, so we get 25000 measurements. Such initialization is often used in tracking problems.
We will apply this procedure to examples 7.1 and 7.2 and draw the corresponding histograms.

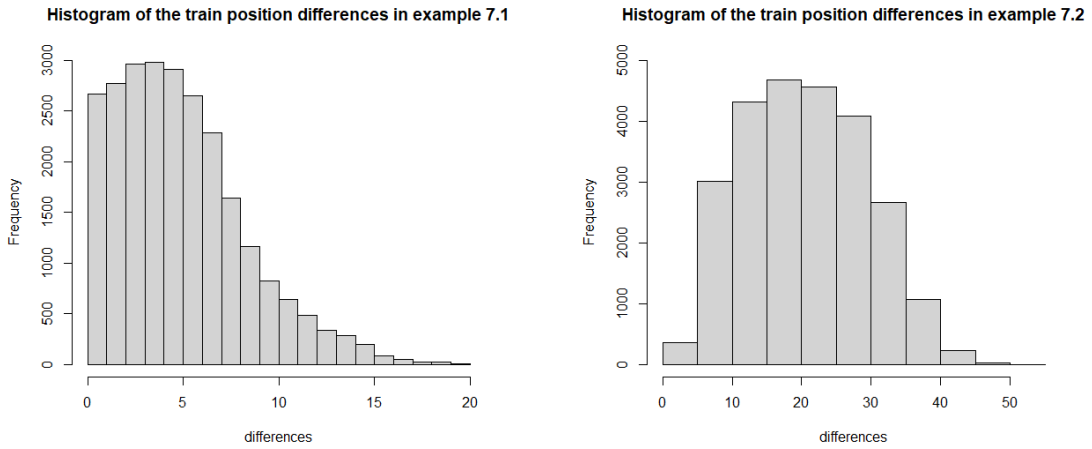

Figure 8: Histogram of the absolute differences of the filter estimate of the train position and the actual train position in the Examples 7.1 and 7.2

The mean of the train position difference data in Example 7.1 is 4.88, while in Example 7.2 it is 20.48, and we notice that we have smaller deviations from the actual train position in Example 7.1 than in Example 7.2. The variance of the train position difference data in Example 7.1 is 11.29, while in Example 7.2 it is 77.51 and we find that we have larger dispersion of the data in Example 7.2 than in Example 7.1. According to the mean and variance, Example 7.1 has a better estimate of the train position.

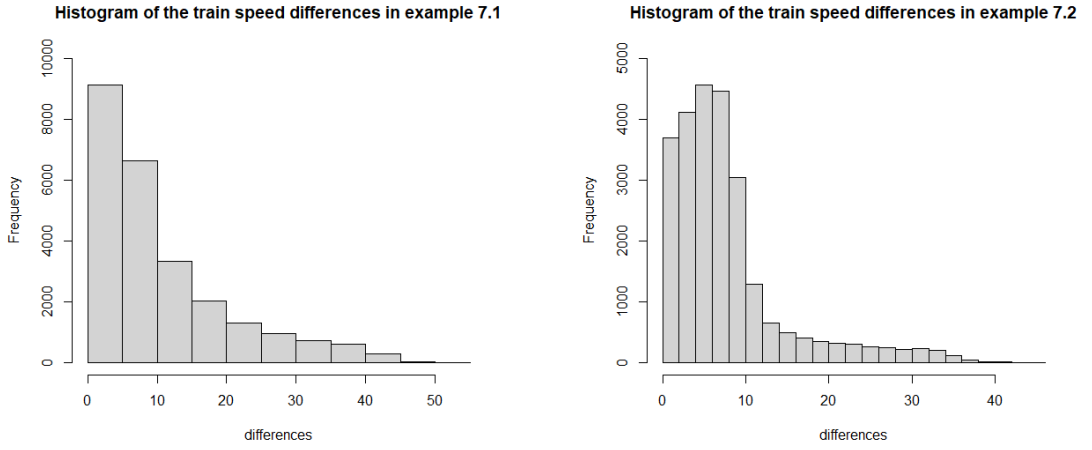

Figure 9: Histogram of the absolute differences of the filter estimate of the train speed and the actual train speed in the Examples 7.1 and 7.2

The mean of the train speed difference data in Example 7.1 is 10.5, while in Example 7.2 it is 7.73 and we notice that we have smaller deviations from the actual train speed in Example 7.2 than in Example 7.1. The variance of the train speed difference data in Example 7.1 is 94, while in Example 7.2 it is 49.06, and we find that we have a larger

dispersion of data in Example 7.1 than in Example 7.2. According to the mean and variance, Example 7.2 has a better estimate of the train speed.

We will now apply the same procedure as for examples 7.1 and 7.2 to examples 7.2 and 7.5.

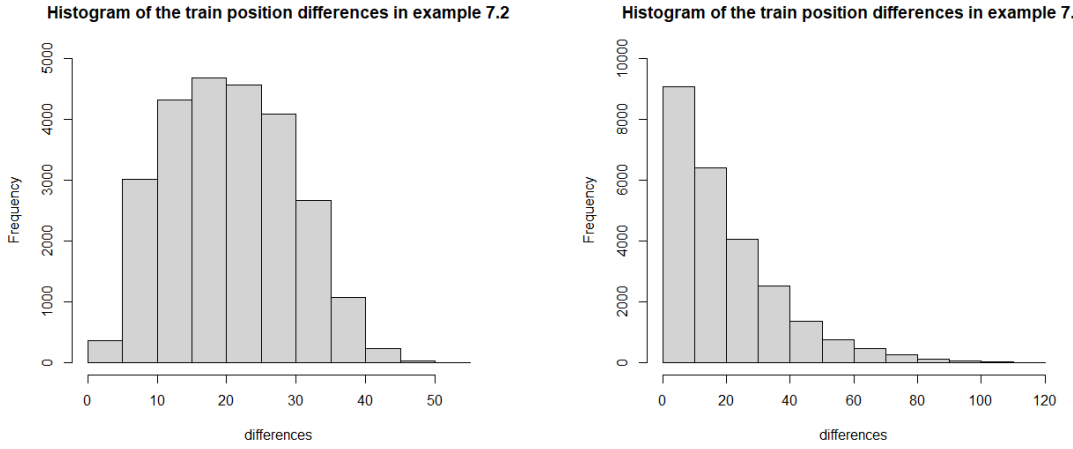

Figure 10: Histogram of the absolute differences of the filter estimate of the train position and the actual train position in the Examples 7.2 and 7.5

The mean of the train position difference data in Example 7.2 is 20.48, while in Example 7.5 it is 19.51, and we notice that we have smaller deviations from the actual train position in Example 7.5 than in Example 7.2. The variance of the train position difference data in Example 7.2 is 77.51, while in Example 7.5 it is 288.6 and we find that we have a larger dispersion of the data in Example 7.5 than in Example 7.2. Although the mean of Example 7.5 is slightly smaller, Example 7.2 has a better estimate of the train position because of the smaller variance.

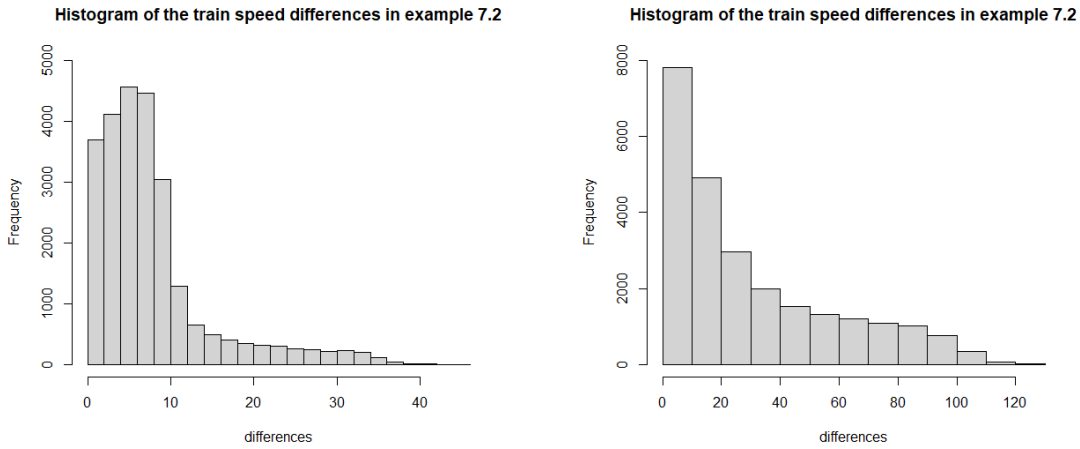

Figure 11: Histogram of the absolute differences of the filter estimate of the train speed and the actual train speed in the Examples 7.2 and 7.5

The mean of the train speed difference data in Example 7.2 is 7.73, while in Example 7.5 it is 30.08 and we notice that we have smaller deviations from the actual train speed in Example 7.2 than in Example 7.5. The variance of the train speed difference data in Example 7.2 is 49.06, while in Example 7.5 it is 30.08, and we fing that we have a larger dispersion of the data in Example 7.2 than in Example 7.5. Although the variance of Example 7.5 is smaller, Example 7.2 has a better estimate of the train speed because od the smaller mean.

A R code

The following code partially follows [6]:

```
prediction_kf <- function(X, P, F, Q){
        X<-F%*%X
        pom<- P%*%t(F)
        P<- F%*%pom + Q
        return(list(X,P))
}

pdf <- function(X,M,S){
        if(dim(M)[2]==1){
                Dx<- X - rep(M, times = dim(X)[2])
                pom2<-solve(S) %*% Dx
                E<- 0.5 * apply ((Dx*pom2),1,sum)
                E<- E+0.5*dim(M)[1]*log(2*pi)+ 0.5*log(det(S))
                P<-exp(-E)
```

```
        }
        else if(dim(X)[1]==1){
                Dx<- rep(X, dim(M)[2])-M
                pom2<-solve(S) %*% Dx
                E<- 0.5 * apply ((Dx*pom2),1,sum)
                E<-E + 0.5*dim(M)[1]*log(2*pi)+0.5*log(det(S))
                P<-exp(-E)
        }
        else{
                Dx<- X-M
                pom2<-solve(S) %*% Dx
                E<- 0.5 * apply ((t(Dx)*pom2),1,sum)
                E<- E + 0.5*dim(M)[1]*log(2*pi)+ 0.5*log(det(S))
                P<-exp(-E)
        }
        return(list(P[1],E[1]))
}

update <- function(X,P,Y,H,R){
        Im<- H %*% X
        pom3<- P %*% t(H)
        Is<- R + H %*% pom3
        pom4<- t(H) %*% solve(Is)
        K<- P %*% pom4
        X<- X + K %*% (Y-Im)
        pom5<- K %*% H
        P<- P - pom5 %*% P
        return(list(X,P,K,Im,Is))
}

X<-matrix(0,ncol=1,nrow=2)
#initialization of the starting position on 500m and
#speed on 50m/s
X[1,1]<-600
X[2,1]<- -65

P<-matrix(0,nrow=2,ncol=2)
P[1,1]<-3^2
P[2,2]<-10^2

T<-0.1
F<-diag(1,ncol=2,nrow=2)
```

```
F[1,2]<-T
F[2,1]<-0

Q<-diag(1,nrow = dim(X)[1])

num_of_meas <- 30
Y<-matrix(0,ncol=num_of_meas,nrow=1)
Y[1,1] <- 500
train_speed <- -50
real_tra<-numeric(num_of_meas)
real_tra[1]<-500
for(i in 2:num_of_meas)
{
  Y[1,i] <- Y[1, i-1]+train_speed*T+rnorm(1, mean=0, sd=1)
  real_tra[i]<-real_tra[i-1]+train_speed*T
}

H<-matrix(0,ncol=2,nrow=1)
H[1,1]<-1

R<- diag(1, nrow = 1)

c<-numeric(num_of_meas)
d<-numeric(num_of_meas)
for(i in 1:num_of_meas){
        c[i]<-X[1,1]
        d[i]<-X[2,1]
        z1<- prediction_kf(X,P,F,Q)
        z2<-update(z1[[1]],z1[[2]],Y[1,i],H,R)
        X<-z2[[1]]
        P<-z2[[2]]
}
index <- c(1:num_of_meas)
plot(index,c,col="red",type="p",xlab="Number of measurements",
     ylab="Position",main = "Train position")
points(index,Y[1,],col="blue",pch =3)
points(index,real_tra,col="green",pch=4)
legend(17, 600, legend=c("Filter estimation",
       "Sensor measurement","Actual position"),
       col=c("red", "blue","Green"), pch = c(1,3,4), cex=0.8)

plot(index,d,col="red",type="p",xlab="Number of measurements",
```

```
     ylab="Speed",main="Train speed")
points(index,rep(-50, times = num_of_meas),col="green",pch =3)
legend(20, -125, legend=c("Filter estimation", "Actual speed"),
       col=c("red", "green"), pch=c(1,3), cex=0.8)
```

X - state estimate of the previous step
P - the state covariance matrix of the previous step
F - the transition matrix
K - the Kalman Gain
Im - the mean of predictive distribution of Y
Is - the covariance or predictive mean of Y

References

[1] S. Challa, M. R. Morelande, D. Mušicki, R. J. Evans, *Fundamentals of object tracking*, Cambridge University Press, New York, 2011.

[2] Wikipedia, *Kalman filter*, available at:
`https://en.wikipedia.org/wiki/Kalman_filter#History` (May 2021.).

[3] M. Verghaegen, P. V. Dooren, *Numerical Aspects of Different Kalman Filter Implementations*, IEEE Transactions on automatic control, vol. ac-31. no. 10, October 1986.

[4] Wikipedia, *Particle filter*, available at:
`https://en.wikipedia.org/wiki/Particle_filter` (May 2021.).

[5] *R language*, available at:
`https://www.r-project.org/about.html` (May 2021.).

[6] M. Laaraiedh, *Implementation of Kalman Filter with Python Language*, available at:
`https://www.academia.edu/2637009/Implementation_of_Kalman_Filter_with_Python_Language` (May 2021.).

[7] C. A. Naesseth, F. Lindsten, T. B. Schön, *Elements of sequential Monte Carlo*, March 2019, available at:
`https://arxiv.org/abs/1903.04797`

[8] M. Huzak, *Matematička statistika*, available at:
`https://web.math.pmf.unizg.hr/nastava/ms/files/ms1v8.pdf`

[9] N. Sarapa, *Teorija vjerojatnosti*, Školska knjiga, Zagreb, 1992.

Sažetak

U ovom diplomskom radu predstavljene su tri poznate metode za praćenje objekata, Kalmanov filter, point mass filter te filter čestica (particle filter). Kako su sva tri filtera zasnovana na Bayesovoj logici na početku se gradila teorija koja je potom primjenjena na sva tri filtera. Naposljetku smo naveli primjenu Kalmanovog filtera na simuliranim podacima.

Summary

This master thesis is presenting three well-known methods for monitoring objects, Kalman filter, point mass filter and particle filter. The theory was initially built through recursive Bayesian solution because all three methods are based on Bayesian Logic. Finally, we present an application of Kalman filter on a simulated data set under several different scenarios.

Curriculum Vitae

I was born on December 21st, 1996 in Zagreb. I finished X. gymnasium „Ivan Supek“ high school in Zagreb, a natural science course with excellent success for all four years. In 2015 I enrolled the university of Zagreb, Faculty of Science, Department of Mathematics. During the study in co-operation with Professor Zrinka Franušić, Faculty of Science, university of Zagreb, Department of Mathematics, I wrote two popular articles *O distribuciji prostih brojeva*, Acta mathematica Spalatensia, Series didactica, 2018 and *Raznovrsni prosti brojevi*, Acta mathematica Spalatensia, Series didactica, 2020. Also, I wrote a popular article *Marie-Sophie Germain*, Matematika i škola, 2019.