LUT University

DERIVATION OF THE KALMAN FILTER WITH ADAPTIVE NOISE AND PRACTICAL IMPLEMENTATION

Lappeenranta–Lahti University of Technology LUT

Degree Programme in Computational Engineering, Bachelor's thesis

2026

Lam Duc Anh Nguyen

Examiners: Associate Professor Emilia Blåsten

Abstract

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Degree Programme in Computational Engineering

Lam Duc Anh Nguyen

Derivation of the Kalman filter with adaptive noise and practical implementation

Bachelor's thesis

2026

58 pages, 10 figures, 1 tables and 3 appendices

Examiners: Associate Professor Emilia Blåsten

Keywords: Kalman filter, EM algorithm, Extended Kalman filter, Recursive least squares

The Kalman filter is used in various applications and has become the backbone technology in diverse fields including navigation, economics, space exploration, and most recently in autonomous vehicles, etc. This study shows the rigorous derivation of the Kalman filter using the recursive least squares method. We further derive the extended Kalman filter, allowing the use of the Kalman filter to a non-linear world. The problem with the Kalman filter and its extended version is fine-tuning the covariance noise matrix, which plays a crucial role affecting the performance of the filter. One of the solutions to this problem is the use of the EM algorithm, which can therefore enhance the estimation performance of these filters. Using collected real-world data, we show that the combination of the Kalman filter and the EM algorithm improves the estimation accuracy.

Acknowledgements

I would like to express my deepest gratitude to my family for their continuous support and encouragement throughout the process of writing this thesis. I am also grateful to my supervisor Emilia Blåsten for her guidance, her detailed comments, all the encouragement and support during the preparation of my thesis. With all that being said, I hope that this thesis can make a small contribution to the academic world.

Declarations

Turnitin

The originality of this thesis has been reviewed with the Turnitin similarity checking service.

Ai usage

The author of the thesis, Lam Duc Anh Nguyen, used the following AI-tools during the preparation of the thesis:

1. Claude
    a. Purpose of use: Claude is used in the drafting process of the thesis.
    b. Explanation of the use of the tool: Claude is used to polish language in the first version of the drafting thesis.

For further development of the thesis, all the language is polished by the author, Lam Duc Anh Nguyen. No AI tools are used in later version of the thesis.

Responsibility

The author, Lam Duc Anh Nguyen, takes full responsibility for the content of this thesis and has reviewed and edited the content generated by the possible use of AI tools.

Table of contents

Abstract

Acknowledgements

Declarations

Lists of figures and tables 7

1 Introduction 8

1.1 An overview of the Kalman filter 8

1.2 Development history 9

1.3 Objective and structure of the thesis 10

2 Mathematics recap 12

2.1 State vectors, state space, and dynamic system 12

2.2 Positive definite, and negative definite 13

2.3 Woodbury matrix identity 13

2.4 Some statistics definitions 14

3 Recursive Least Squares Method 16

3.1 Method of Least Square 16

3.2 Derivation of the Recursive Least Squares Method 19

3.2.1 Recursive least squares method 19

3.2.2 Example 26

4 Kalman filtering 31

4.1 Introduction on general equations 31

4.2 Priori state and posteriori state estimate 32

4.3 Discrete-time Kalman filtering 33

4.3.1 Derivation of discrete-time Kalman filtering 33

4.3.2 Example 37

4.4 Extended Kalman Filtering 39

4.4.1 Discrete-time extended Kalman filter 40

5 EM algorithm 45

5.1 Motivation 45
5.2 Prediction, filter, and smoother values 45
5.3 EM algorithm 46
5.3.1 How does the EM algorithm work? 46
5.3.2 EM algorithm in Kalman filter 46
6 Kalman filter application 51
6.1 Data collection 51
6.2 Implementation and discussion 51
7 Conclusion 56
References 57
Appendices
Appendix 1 MATLAB code for the recursive least squares method
Appendix 2 MATLAB code for the Kalman filter
Appendix 3 Kalman filter and EM algorithm practical example

List of Figures

1 Position, speed, and acceleration estimates of the truck compared with their true values.
2 Recursive Least Squares process.
3 Kalman Filtering updated process.
4 Posteriori estimate equations at time instant k.
5 Kalman Filtering process.
6 Dynamic position and speed estimates of the truck compared with their true values.
7 EM algorithm.
8 Distance and speed estimates of the truck compared with their respective filter values.
9 Noisy distance and speed measurements of the truck compared with their respective filter values and smoother values.
10 Distance and speed estimates of the truck compared with their respective new filter values and smoother values.

List of Tables

1 Position measurements of the ice cream truck

1 Introduction

1.1 An overview of the Kalman filter

The Kalman filter was developed by Rudolf E. Kalman in 1960 (Kalman, 1960), and still plays an important role in both research and quantitative fields. With more than 440000 academic papers related to the Kalman filter based on Google Scholar, it is not wrong to state that this algorithm is among the most important innovations of the twentieth century and has gained significant attention from researchers around the world (Humpherys, Redd & West, 2012). The most appreciated feature of this algorithm is its recursive form that optimally estimates the unknown state variables of a dynamic system using discrete-time noisy data measurements (Chui & Chen, 2017, pp. xi-xii). By using recursion, the Kalman filter guaranties ease in its implementation and remains an indispensable technology in diverse fields, including aerospace exploration, finance, satellite navigation, radar systems (Pearson & Stear, 1974), medical science, and autonomous vehicle control (Kiencke & Nielsen, 2000) even decades after its introduction. One of its most celebrated applications was in the Apollo mission, where it played a key role in the guidance system that took astronauts to the moon and most importantly brought them back to Earth (Grewal & Andrews, 2010) (Battin, 1982).

In tracking problems, a position sensor or radar can be used to track the position of many planes in real time, display the position of a system of autonomous delivery cars, or simply show the location where we are currently at. However, these measurements data are not always perfect. Signal corruption, measurement uncertainty, the effects of the environment, or errors in the sensor itself are inevitable, leading to variation in the final results. Even if one can come up with a prediction model that uses the observed data to calculate the estimations, the noise from the environment is not a simple effect that the model can take into account, which corresponds to the estimation errors. To obtain a better approximation, taking the average of the measurement data and the estimation values is one solution. However, if the measurement data are noisier than the prediction values, giving the same weight to both the measurement and the prediction values is not a good choice. The Kalman filter is different in that it gives more gain toward the measurement data if it is more accurate compared to the prediction model and vise versa. The GPS (Global Positioning System), for instance, relies heavily on the Kalman filter, which combines measurement data from satellites with prediction data obtained from the dynamic systems model to produce better state approximation values (Grewal, Weill & Andrews, 2007).

1.2 Development history

Before diving into the theory of the Kalman filter and other estimation methods, it is helpful to understand the historical development that built the solid foundation for the emerging of the filter.

The method of least squares is credited to Carl Friedrich Gauss, who claimed to have developed it in 1795 at the age of eighteen — though it was Adrien-Marie Legendre who first published the method in 1805, sparking a long-running priority dispute between the two (Stigler, 1981). There is an interesting story behind Gauss and how he estimated the orbit of a newly discovered planet called Ceres. The story began when the Italian astronomer Giuseppe Piazzi unexpectedly discovered a new planet whose orbit lay between Mars and Jupiter. Piazzi was able to track the orbit of the new planet for 41 nights before his work was interrupted, after which he lost its position and was unable to find it again (Grewal & Andrews, 2001, pp. 5). This was a significant breakthrough in the 1800s, since the existence of a planet between Mars and Jupiter had remained a long-standing mystery. At that time, one of the most famous studies about the relationship between the distance from the sun to other planets was the Titius-Bode law. This law gained significant attention since its discovery of Uranus in 1781 and reinforced the belief that there must be a planet between Mars and Jupiter (Serio, Manara & Sicoli, 2002). The loss of Ceres brought attention to Gauss, who then used the noisy measurement data and the method of Least Squares to estimate the possible position of the missing planet allowing it to be relocated on the last day of 1801, which was almost a year after Piazzi's discovery on January 1, 1801 (Grewal & Andrews, 2001, pp. 6-7).

In fall 1959, there was a discussion between Dr. Harry J. Goett, who was the director of NASA's Goddard Space Flight Center (GSFC), and other members of the Ames Research center (ARC), including Dr. Stanley F. Schmidt, chief of the Dynamics Analysis Branch, about the future of manned spacecraft program, especially for the manned lunar mission (McGee & Schmidt, 1985, pp. 5). At that time, there was a need to find an optimal algorithm that has the ability to navigate and guide the spacecraft from Earth to the Moon and back to Earth. Dr. Schmidt and his team members had tried and reviewed some of the estimation theory at the time, such as the least-square method. However, due to the complexity of the computation for the computer on board at the time, these methods are not applicable for the mission. During this era, the Wiener filter was commonly used for the problems of guidance homing missiles, which caught the attention of Dr. Schmidt and his team. They wondered whether the Wiener filter could be applied to the navigation problem that they were trying to solve. However, the Wiener filter required assumptions for the state and measurements, which were not applicable for the lunar navigation system (Simon, 2006, pp. 493) (McGee

& Schmidt, 1985, pp. 6).

The discovery of Kalman filtering took place in late November 1958, when Kalman was on a late train (Grewal & Andrews, 2001, pp. 14). Despite being tired, the idea he came up with that night would go on to mark an extraordinary breakthrough in filtering theory. Two years later, in the fall of 1960, Dr. Kalman presented his work to Dr. Schmidt and the staff members of the Dynamic Analysis Branch. Since Dr. Schmidt and his team had been working on the problem, they immediately recognized it as a potential solution to the team's challenge: estimating the trajectory of the spacecraft for the Lunar mission. Schmidt and his research team later investigated the effect of nonlinearities on the Kalman filter in greater depth. He and his team found out that excellent performance could be achieved by linearizing the problem about the estimated trajectory, which was later known as the extended Kalman filtering. Even with large estimation errors due to nonlinear effects, the extended Kalman filtering was still able to converge to the true trajectory (McGee & Schmidt, 1985, pp. 8).

1.3 Objective and structure of the thesis

The objective of the thesis is to introduce a detailed derivation of the Kalman filter using the recursive least squares method, since these two algorithms have pretty much the same idea in predicting and correcting the estimation values. Also, the derivation of the Kalman filter using the RLS is considered straightforward using linear algebra and few statistics definitions. We also derive the extended Kalman filter which allows us to apply the Kalman filter to non linear problems. However, the problem with both the Kalman filter and the extended version is that the covariance noise matrix is built by experience rather than from the observing data or mathematical derivation. Therefore, a good approximation for the covariance noise matrix is required for the best performance of the algorithm. We will also introduce the EM algorithm to estimate the covariance noise matrix efficiently. Finally, we will implement a tracking problem using non-simulated data.

The structure of the thesis is as follows. First, we cover the required mathematics that is used in the derivation of the Kalman filter in the section 2. Next, we will have a brief introduction about the method of least squares which is the motivation for the recursive least squares in section 3.1. The detailed derivation of the recursive least squares is then introduced in section 3.2. And in section 3.2.2, we discuss an estimation example for the initial position and speed of an ice cream truck, and its acceleration which is assumed to be constant using the recursive least squares. The derivation of the Kalman filter is introduced in section 4 which is based on the recursive least squares derived earlier in section 3.2. In this section, we also have an example of tracking the position and speed of the ice cream truck which is an extended version of the previous example. Following up in sections 4.4 and 5 is the general

derivation of the extended Kalman filter and the EM algorithm, respectively. The final part will be the application using non-simulated data to show the power of the Kalman filer and the use of the EM algorithm in broadening the performance of the filter algorithm.

2 Mathematics recap

This section reviews some of the essential theory and definition that will become useful in a later section. First, we will go through a short introduction to state vector, state space, and dynamic system. After that, some important linear algebra lemma and definitions will be introduced next. And finally, we will go through some statistics definition, including random variable, expected value, variance, covariance, and some essential properties corresponding to these definitions.

2.1 State vectors, state space, and dynamic system

In the first section, the term "dynamic systems" was used, but what does that actually mean, and why bother formalizing it? The idea traces back to Isaac Newton, who pioneered the use of differential equations in the seventeenth century to model real-life phenomena mathematically. Applying this to the motion of the planets around the sun, Newton's equations gave us a way to precisely predict a planet's position and velocity at any point in time (Grewal & Andrews, 2001, pp. 35-36). Below is a brief introduction to some terminologies that will be used during the development of the Kalman filter.

A state represents the current condition of a system, described by one or more variables. For example, the velocity of a car at a specific time is 50 km/h. That is the state of the car. Similarly, a dynamic system is used to describe how a system's state evolves over time. A state vector is formed by stacking the state variables that describe the dynamic system. The set of all possible values that the state vector can take is called as the state space. The state space is a linear space where the state vector belongs to. To make these general definitions clearer, consider the following example.

A damped harmonic oscillator can be described as:

$$\mathbf{F} = -k\mathbf{x} - c\frac{d\mathbf{x}}{dt} = m\frac{d^2\mathbf{x}}{dt^2} \tag{1}$$

where $\mathbf{F}$ is the force acting on the mass, $\mathbf{x}$ is the mass's displacement from its equilibrium position., k is the spring constant, and c is the damping coefficient. The first-order derivative of $\mathbf{x}$ with respect to time t is the velocity, and the second-order derivative of position $\mathbf{x}$ with respect to time t is the acceleration. We can also rewrite Equation (1) as an equivalent system of two first-order differential equations:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}\\ \dot{\mathbf{x}}\end{bmatrix} = \begin{bmatrix}0 & 1\\ \frac{-k}{m} & \frac{-c}{m}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \dot{\mathbf{x}}\end{bmatrix} \tag{2}$$

Let $\mathbf{s} = \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix} \in \mathbb{R}^2$. We can rewrite Equation (2) as follows:

$$\dot{\mathbf{s}} = \begin{bmatrix} 0 & 1 \\ \frac{-k}{m} & \frac{-c}{m} \end{bmatrix} \mathbf{s}$$

Here, $\mathbf{s} = \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix}$ has a size of 2 and the state space is $\mathbb{R}^2$, which is formed by stacking the state variables (position $\mathbf{x}$, and velocity $\dot{\mathbf{x}}$) that describe the damped oscillator system.

2.2 Positive definite, and negative definite

Definition 2.1. A symmetric matrix A (i.e., $A^T = A$) is called *positive definite* if

$$v^T A v > 0, \quad \text{for all } v \in \mathbb{R}^n \setminus \{0\}. \tag{3}$$

The above definition is adapted from (Strang, 2006, pp. 318-319). Similarly, A is negative definite if $v^T A v < 0$ for all nonzero vector $v \in \mathbb{R}^n$, negative semidefinite if $v^T A v \leq 0$, and positive semidefinite if $v^T A v \geq 0$.

There is also a nice property for symmetric matrix. Assume that matrix A is nonsingular and symmetric, we can prove that $A^{-1} = (A^{-1})^T$.

Proof. Since A is symmetric, $A^T = A$. Taking the inverse of both sides and using the general identity $(A^T)^{-1} = (A^{-1})^T$, we obtain

$$(A^{-1})^T = (A^T)^{-1} = A^{-1},$$

which completes the proof. □

2.3 Woodbury matrix identity

Lemma 1. *Let $A \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{k\times k}$, $U \in \mathbb{R}^{n\times k}$, and $V \in \mathbb{R}^{k\times n}$, then*

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{4}$$

if we assume that A, C, and $A+UCV$ are non singular.

2.4 Some statistics definitions

Definition 2.2. A random variable is a function that maps from the sample space to real number. Let Ω be the sample space, and random variable X,

$$X : \Omega \to \mathbb{R} \tag{5}$$

Remark 2.1. A random variable is not a variable that is taken randomly. Suppose one tosses a coin 2 times and record the result whether it is head (H) or tail (T). Let random variable X as the total of head observed. The sample space will have a size of $2^2 = 4$, which is defined as follows:

$$\Omega = \{HH, HT, TH, TT\}$$

Then,

$$X(HH) = 2,\ X(HT) = 1,\ X(TH) = 1,\ X(TT) = 0$$

Definition 2.3. (Billingsley, 2012) Let X be the random variable with finite many outcomes $x_1, \ldots, x_n$. Each of the outcome x_i has a corresponding probability value p_i. The expected value of a random variable X is given as:

$$E[X] = x_1 p_1 + x_2 p_2 + \cdots + x_n p_n \tag{6}$$

Since the sum of the probability of $p_1 + \cdots + p_n$ has to be equal to 1, the expected value of the random variable X can also be interpreted as the weighted average of the x_i values.

Definition 2.4. Let X be a random variable. The variance of a random variable X, denoted as $Var(X)$ is given as:

$$Var(X) = E[(X - \mu)^2] \tag{7}$$

where $\mu = E[X]$. The variance of a random variable X is the expectation of the square of the difference between X and its expectation μ. It can also be shown that:

$$Var(X) = E[X^2] - \mu^2$$

There are also some properties corresponding to the expected value and the variance of a

random variable X.

$$E[aX+b] = E[aX] + E[b] = aE[X] + b$$

and

$$Var(aX+b) = a^2 Var(X)$$

where a, and b are the real number.

Definition 2.5. (Chui & Chen, 2017) Let X and Y be random n- and m-vectors, respectively. The covariance of X and Y is defined as

$$Cov(X,Y) = E[(X - E[X])(Y - E[Y])^T] \tag{8}$$

When $Y = X$, then

$$Var(X) = Cov(X,Y) \tag{9}$$

3 Recursive Least Squares Method

The main point of this section is to derive the recursive least squares method, which can be seen as the recursive form of the ordinary least squares method (Haykin, 2014). The ordinary least squares is also known as the batch processing method, which uses all available data to estimate unknown constants (Zarchan et al., 2009, pp.91). As more data are collected, the computation for the least square algorithm will become more expensive and time consuming. This is because the least squares algorithm requires the inverse of a matrix whose dimension will get larger as more data is used.

The recursive least squares method is a widely used algorithm in systems and control theory to estimate fixed constant parameters using new incoming data to continuously update and improve previous estimation (Islam & D. Bernstein, 2019) (Dumortier, 1985) (Lai & D. S. Bernstein, 2024), and also a solution to the computation problem of the ordinary least squares. In other words, recursive least squares uses new incoming data to continuously update the estimation of the unknown constant state values. The idea of correcting the old state estimate using the new incoming data of the recursive least squares is the same as that of the Kalman filter, and the derivation of the Kalman filter in section 4 is based on this method. Thus, it is important to understand the RLS thoroughly before deriving the Kalman filter.

3.1 Method of Least Square

The Method of Least Squares, which was first developed by Gauss during the 18th century, is used as the fundamental theory for many curve-fitting programs and still remained as an important tool in mathematics theory up until now. The objective of utilizing this method is to determine a curve that provides the best approximation as close as possible to the actual data. We assume that the fitting curve is described by a polynomial model. Our goal is to find the coefficients of this model such that the sum of squared residuals which is the difference between the measured value and the value estimated by the model, is minimized. The least-squares problem can be written as follows:

$$R = \sum_{k=1}^{n} (\hat{x_k} - x_k^*)^2 \tag{10}$$

where R $\in \mathbb{R}$ is the sum of square residuals we want to minimize, $\hat{x}_k \in \mathbb{R}$ is the output from our polynomial model, and $x_k^* \in \mathbb{R}$ is our actual data (Zarchan et al., 2009, chap. 2, pp. 41–42).

Suppose we are trying to fit a straight line to the data,

$$y_i = a + bx_i \tag{11}$$

where $y_i \in \mathbb{R}$ is the i^{th} measurement, $a \in \mathbb{R}$ is the intercept, $b \in \mathbb{R}$ is the slope of the line, and $x_i \in \mathbb{R}$ is the predictor of our model. For the first measurement, we have:

$$y_1 = a + bx_1$$

For the second measurement, we have:

$$y_2 = a + bx_2$$

Up to the k^{th} measurement, we have:

$$y_k = a + bx_k$$

We can also write this in matrix form as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_k \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \tag{12}$$

Similarly, suppose we wish to fit an n-th order polynomial. In this case, we have:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}. \tag{13}$$

Or we can rewrite as:

$$Y = Ha \tag{14}$$

where $Y \in \mathbb{R}^k$ is our vector of measurements, $H \in \mathbb{R}^{k \times (n+1)}$ is our predictor matrix and we assume that H has full column rank, and vector $a \in \mathbb{R}^{(n+1)\times 1}$ is our vector of polynomial coefficients. Notice that the first column of H consists entirely of ones because the intercept coefficient a_0 is multiplied by one in each observation.

Our objective is to minimize the sum of squared errors. By doing that, we are trying to fit the curve as close as possible to the actual measurements. The sum squared errors can be

written as:

$$J = (Y - Ha)^T (Y - Ha) = Y^T Y - Y^T Ha - a^T H^T Y + a^T H^T Ha$$

Since $Y^T Ha$ and $a^T H^T Y$ are both scalars, and a scalar is equal to its own transpose, it follows that $Y^T Ha = (Y^T Ha)^T = a^T H^T Y$. Substituting this back, we obtain:

$$J = Y^T Y - 2a^T H^T Y + a^T H^T Ha \tag{15}$$

Taking the gradient of J with respect to a, we have:

$$\nabla_a J = -2H^T Y + 2H^T Ha. \tag{16}$$

Setting the gradient equal to zero yields:

$$-2H^T Y + 2H^T Ha = 0 \quad \Leftrightarrow \quad a = (H^T H)^{-1} H^T Y, \tag{17}$$

which is the ordinary least-squares estimator.

To verify that this estimated point corresponds to a minimum, we compute the Hessian of J:

$$\nabla_a^2 J = 2H^T H. \tag{18}$$

Our goal is to prove Equation (18) to be positive definite by showing that $H^T H > 0$. Initially, matrix $H \in \mathbb{R}^{k \times (n+1)}$ is assumed to have full column rank. Therefore, when solving a linear equation $Hx = 0$ for any nonzero vector $x \in \mathbb{R}^{n+1}$, $x = 0$ is the only solution which does not satisfy the condition for the vector x. Thus $Hx \neq 0$. Taking $Hx \neq 0$ into account, consider the following:

$$x^T H^T Hx = (Hx)^T (Hx) = \|Hx\|^2 > 0.$$

Since $H^T H$ is symmetric and $x^T H^T Hx$ is greater than zero for all nonzero vectors x, we can conclude that $H^T H$ is positive definite by the definition mentioned in 2.1. Therefore, the solution given by Equation (17) is the unique global minimizer of the least-squares cost function.

3.2 Derivation of the Recursive Least Squares Method

3.2.1 Recursive least squares method

In the previous section, we have a general idea on how the method of least squares is being used. However, one can see the computational problem with the least squares method as we have to first collect all the measurements before computing the optimal model for our estimation. Thus, the main goal of this section is to show a general step-by-step on how to derive the recursive least squares which solves the problem of the Least squares method, and is used to derive the Kalman filter later in section 4.

Suppose that we have collected a sequence of k measurements. These collected measurement will be stacked into the vector $Y \in \mathbb{R}^{k\times 1}$. Let $\mathbf{x} \in \mathbb{R}^{n\times 1}$ denote an unknown constant parameter vector to be estimated. The measurement model is assumed to be linear. Therefore, we have the following equation:

$$Y = H\mathbf{x} + v, \tag{19}$$

or in expanded form

$$\begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ \vdots & \ddots & \vdots \\ h_{k1} & \cdots & h_{kn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} v_1 \\ \vdots \\ v_k \end{bmatrix}.$$

where $H \in \mathbb{R}^{k\times n}$ is the observation matrix, and $v \in \mathbb{R}^{k\times 1}$ is the measurement noise vector. The measurement noise is assumed to have zero-mean and mutually uncorrelated. Consequently, its covariance matrix is given by:

$$R = E[vv^T] = \begin{bmatrix} \sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_k^2 \end{bmatrix} \tag{20}$$

Now, let us define the residuals $\varepsilon_y \in \mathbb{R}^k$ as the difference between the actual measurements and our estimation:

$$\varepsilon_y = Y - H\hat{x} \tag{21}$$

Our objective is to minimize the sum of weighted residual squares with respect to $\hat{x}$. Therefore, our cost function can be formulated as follows:

$$J = \frac{\varepsilon_{y_1}^2}{\sigma_1^2} + \cdots + \frac{\varepsilon_{y_k}^2}{\sigma_k^2} = \varepsilon_y^T R^{-1} \varepsilon_y = (Y - H\hat{\mathbf{x}})^T R^{-1} (Y - H\hat{\mathbf{x}}) \tag{22}$$

The measurement model (19), covariance assumptions (20), and the cost function (22) follow the presentation in (Simon, 2006, Chap. 3, pp. 79–84).

Next, let us find the optimizer of the cost function $J(\hat{\mathbf{x}})$:

$$\begin{aligned} J(\hat{\mathbf{x}}) &= (Y - H\hat{\mathbf{x}})^T R^{-1}(Y - H\hat{\mathbf{x}}) \\ &= (Y^T - \hat{\mathbf{x}}^T H^T)R^{-1}(Y - H\hat{\mathbf{x}}) \\ &= Y^T R^{-1} Y - Y^T R^{-1} H\hat{\mathbf{x}} - \hat{\mathbf{x}}^T H^T R^{-1} Y + \hat{\mathbf{x}}^T H^T R^{-1} H\hat{\mathbf{x}} \\ &= Y^T R^{-1} Y - 2\hat{\mathbf{x}}^T H^T R^{-1} Y + \hat{\mathbf{x}}^T H^T R^{-1} H\hat{\mathbf{x}} \end{aligned}$$

where we use the fact that

$$Y^T R^{-1} H\hat{\mathbf{x}} = (Y^T R^{-1} H\hat{\mathbf{x}})^T = \hat{\mathbf{x}}^T H^T R^{-1} Y,$$

since $Y^T R^{-1} H\hat{\mathbf{x}}$ is a scalar and R^{-1} is symmetric.

Taking the gradient of J with respect to the vector $\hat{\mathbf{x}}$, we obtain the following.

$$\nabla_{\hat{\mathbf{x}}} J = -2H^T R^{-1} Y + 2H^T R^{-1} H\hat{\mathbf{x}}$$

Setting the gradient to zero, the optimizer is as follows:

$$\begin{aligned} -2H^T R^{-1} Y + 2H^T R^{-1} H\hat{\mathbf{x}} &= 0 \\ H^T R^{-1} H\hat{\mathbf{x}} &= H^T R^{-1} Y \\ \hat{\mathbf{x}} &= (H^T R^{-1} H)^{-1} H^T R^{-1} Y \end{aligned} \tag{23}$$

To make the system recursive, we partition the matrices in step k:

$$\left.\begin{aligned} H_{k-1} &= \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ \vdots & \ddots & \vdots \\ h_{(k-1)1} & \cdots & h_{(k-1)n} \end{bmatrix} \in \mathbb{R}^{(k-1)\times n}, \\ H_k &= \begin{bmatrix} h_{k1} & \cdots & h_{kn} \end{bmatrix} \in \mathbb{R}^{1\times n} \end{aligned}\right\} \implies H_k = \begin{bmatrix} H_{k-1} \\ H_k \end{bmatrix} \in \mathbb{R}^{k\times n} \tag{24}$$

Similarly, we partition the measurement vector Y_k and the noise covariance matrix $\mathfrak{R}_k$:

$$Y_{k-1} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{k-1} \end{bmatrix} \in \mathbb{R}^{(k-1)\times 1}, \quad Y_k = \begin{bmatrix} Y_{k-1} \\ y_k \end{bmatrix} \in \mathbb{R}^{k\times 1} \tag{25}$$

$$R_{k-1} = \begin{bmatrix} \sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{k-1}^2 \end{bmatrix} \in \mathbb{R}^{(k-1)\times(k-1)}, \quad R_k = \begin{bmatrix} R_{k-1} & 0 \\ 0 & \sigma_k^2 \end{bmatrix} \in \mathbb{R}^{k\times k} \tag{26}$$

By expanding $H_k^T R_k^{-1} Y_k$ and $H_k^T R_k^{-1} H_k$, which are the terms of the optimizer in Equation (23), using the measurement vector Y_k and the noise covariance matrix R_k defined above, we have:

$$H_k^T R_k^{-1} Y_k = \begin{bmatrix} H_{k-1} \\ H_k \end{bmatrix}^T \begin{bmatrix} R_{k-1}^{-1} & 0 \\ 0 & \frac{1}{\sigma_k^2} \end{bmatrix} \begin{bmatrix} Y_{k-1} \\ y_k \end{bmatrix} = H_{k-1}^T R_{k-1}^{-1} Y_{k-1} + {H_k}^T \frac{1}{\sigma_k^2} y_k \tag{27}$$

$$H_k^T R_k^{-1} H_k = \begin{bmatrix} H_{k-1} \\ H_k \end{bmatrix}^T \begin{bmatrix} R_{k-1}^{-1} & 0 \\ 0 & \frac{1}{\sigma_k^2} \end{bmatrix} \begin{bmatrix} H_{k-1} \\ H_k \end{bmatrix} = H_{k-1}^T R_{k-1}^{-1} H_{k-1} + H_k^T \frac{1}{\sigma_k^2} H_k \tag{28}$$

Below is the general derivation of the error covariance matrix, denoted as P. Until now, we know that the optimizer $\hat{\mathbf{x}} = (H^T R^{-1} H)^{-1} H^T R^{-1} Y$. Substitute $Y = H\mathbf{x} + v$ to $\hat{\mathbf{x}}$, we have:

$$\begin{aligned} \hat{\mathbf{x}} &= (H^T R^{-1} H)^{-1} H^T R^{-1} (H\mathbf{x} + v) \\ &= (H^T R^{-1} H)^{-1} H^T R^{-1} H\mathbf{x} + (H^T R^{-1} H)^{-1} H^T R^{-1} v \\ &= \mathbf{x} + (H^T R^{-1} H)^{-1} H^T R^{-1} v \end{aligned}$$

Let $\varepsilon_{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}}$ be estimation error. We obtain the following:

$$\varepsilon_{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}} = -(H^T R^{-1} H)^{-1} H^T R^{-1} v$$

Thus, the covariance of the estimation-error is:

$$\begin{aligned}
E[\varepsilon_{\mathbf{x}}\varepsilon_{\mathbf{x}}^T] &= E[Avv^TA^T] = AE[vv^T]A^T = ARA^T \\
&= (H^TR^{-1}H)^{-1}H^TR^{-1}R[(H^TR^{-1}H)^{-1}H^TR^{-1}]^T \\
&= [(H^TR^{-1}H)^{-1}]^T \\
&= [(H^TR^{-1}H)^T]^{-1} \\
&= (H^TR^{-1}H)^{-1} = P
\end{aligned}$$

in which $A = -(H^TR^{-1}H)^{-1}H^TR^{-1}$, $H^TR^{-1}H$ is symmetric, and we use the assumption for the covariance of the noise v which has zero mean and is mutually uncorrelated $E(vv^T) = R$. Thus, the estimation-error covariance is $P = (H^TR^{-1}H)^{-1}$.

Next, let us denote the estimation-error covariance matrix at step $k-1$ and k as P_{k-1} and P_k, respectively. We obtain the following:

$$P_{k-1} = (H_{k-1}^TR_{k-1}^{-1}H_{k-1})^{-1}, P_k = (H_k^TR_k^{-1}H_k)^{-1} \in \mathbb{R}^{n\times n} \tag{29}$$

The estimator at step $k-1$ is defined as:

$$\hat{\mathbf{x}}_{k-1} = P_{k-1}H_{k-1}^TR_{k-1}^{-1}Y_{k-1} \implies P_{k-1}^{-1}\hat{\mathbf{x}}_{k-1} = H_{k-1}^TR_{k-1}^{-1}Y_{k-1} \tag{30}$$

Using the relations in (24) through (30), we can rewrite the optimizer as:

$$\begin{aligned}
\hat{\mathbf{x}}_k &= (H_k^TR_k^{-1}H_k)^{-1}H_k^TR_k^{-1}Y_k = P_kH_k^TR_k^{-1}Y_k \\
&= P_k(H_{k-1}^TR_{k-1}^{-1}Y_{k-1} + H_k^T\frac{1}{\sigma_k^2}y_k) \\
&= P_k(P_{k-1}^{-1}\hat{\mathbf{x}}_{k-1} + H_k^T\frac{1}{\sigma_k^2}y_k) \\
&= P_k[(P_k^{-1} - H_k^T\frac{1}{\sigma_k^2}H_k)\hat{\mathbf{x}}_{k-1} + H_k^T\frac{1}{\sigma_k^2}y_k] \\
&= P_k(P_k^{-1}\hat{\mathbf{x}}_{k-1} - H_k^T\frac{1}{\sigma_k^2}H_k\hat{\mathbf{x}}_{k-1} + H_k^T\frac{1}{\sigma_k^2}y_k) \\
&= \hat{x}_{k-1} + P_kH_k^T\frac{1}{\sigma_k^2}(y_k - H_k\hat{x}_{k-1})
\end{aligned}$$

From (28), we have the following:

$$P_k^{-1} = P_{k-1}^{-1} + H_k^T\frac{1}{\sigma_k^2}H_k \implies P_k = (P_{k-1}^{-1} + H_k^T\frac{1}{\sigma_k^2}H_k)^{-1}$$

Using the lemma 1, the estimation-error covariance matrix at step k is defined as:

$$P_k = P_{k-1} - P_{k-1}H_k^T(\sigma_k^2 + H_k P_{k-1} H_k^T)^{-1} H_k P_{k-1} \tag{31}$$

Multiplying both side of Equation (31) with $H_k^T \frac{1}{\sigma_k^2}$, we have:

$$\begin{aligned} P_k H_k^T \frac{1}{\sigma_k^2} &= P_{k-1}H_k^T \frac{1}{\sigma_k^2} - P_{k-1}H_k^T(\sigma_k^2 + H_k P_{k-1} H_k^T)^{-1} H_k P_{k-1} H_k^T \frac{1}{\sigma_k^2} \\ &= P_{k-1}H_k^T \frac{1}{\sigma_k^2}[I - (\sigma_k^2 + H_k P_{k-1} H_k^T)^{-1} H_k P_{k-1} H_k^T] \\ &= P_{k-1}H_k^T(\sigma_k^2 + H_k P_{k-1} H_k^T)^{-1} \end{aligned}$$

To make the recursive equations look like in the book (Simon, 2006, pp. 86-87), new notation will be introduced for the covariance of v_k. Instead of using σ_k^2 for the k^{th} variance, R_k will be the alternative notation. This yields the Recursive Least Squares algorithm

$$K_k = P_{k-1}H_k^T(R_k + H_k P_{k-1} H_k^T)^{-1} = P_k H_k^T R_k^{-1} \tag{32}$$

$$P_k = P_{k-1} - K_k H_k P_{k-1} \tag{33}$$

$$\hat{x}_k = \hat{x}_{k-1} + K_k(y_k - H_k \hat{x}_{k-1}) \tag{34}$$

Although the simplified covariance update in Equation (33) preserves positive definiteness in exact mathematical arithmetic, finite-precision computer implementations of the recursion may introduce numerical round-off errors. Over many iterations, these errors can accumulate, causing P_k to lose its symmetric positive-definite properties leading to estimator divergence (Simon, 2006, pp. 86-88).

Naturally, one might wonder why the gain K_k takes the specific form given in Equation (32), and whether Equation (33) can be written in a more numerically robust way. The following proof will show us how to derive the recursive gain K_k and the robust formulation of the estimation-error covariance P_k.

Theorem 1. *Let* $\mathbf{x} \in \mathbb{R}^n$ *be an n-dimensional state vector,* $H_k \in \mathbb{R}^{m \times n}$ *be the measurement matrix, which maps the n-dimensional state vector* $\mathbf{x}$ *that needs to be estimated to the measurement values* $y_k \in \mathbb{R}^m$*,* $v_k \in \mathbb{R}^m$ *is the measurement noise, and* $K_k \in \mathbb{R}^{n \times m}$ *is called the estimator gain matrix. The measurement noise* v_k *is assumed to be zero-mean and mutually*

uncorrelated, which means that:

$$E[v_k] = 0$$
$$R_k = E[v_k v_k^T] = \begin{bmatrix} \sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_k^2 \end{bmatrix}$$

The state estimate $\mathbf{x}_k$ *is computed using the state estimate at the previous time index* $k-1$*, the gain of the estimator* K_k*, and the correction term which is defined as the difference between the actual measurement at time index k and the estimated measurement values before seeing the measurement at time index k. The general form of a linear recursive estimator, consisting of a measurement equation and an update rule, is given as:*

$$y_k = H_k \mathbf{x} + v_k \tag{35}$$
$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + K_k(y_k - H_k \hat{\mathbf{x}}_{k-1}) \tag{36}$$

Then, for an arbitrary value of K_k*, the estimation-error covariance* P_k *is:*

$$P_k = (I - K_k H_k) P_{k-1} (I - K_k H_k)^T + K_k R_k K_k^T$$

and the value of the recursive gain K_k*, which minimizes the sum of the variances of the estimation errors is:*

$$K_k = P_{k-1} H_k^T (R_k + H_k P_{k-1} H_k^T)^{-1}$$

Proof. The estimation-error covariance follows by direct substitution:

$$\begin{aligned}
&E[(\mathbf{x} - \hat{\mathbf{x}}_k)(\mathbf{x} - \hat{\mathbf{x}}_k)^T] \\
&= E[\mathbf{x} - \hat{\mathbf{x}}_{\mathbf{k-1}} - K_k(H_k \mathbf{x} + v_k - H_k \hat{\mathbf{x}}_{k-1})][\mathbf{x} - \hat{\mathbf{x}}_{\mathbf{k-1}} - K_k(H_k \mathbf{x} + v_k - H_k \hat{\mathbf{x}}_{k-1})]^T \\
&= E[(I - K_k H_k)\varepsilon_{x,k-1} - K_k v_k][(I - K_k H_k)\varepsilon_{x,k-1} - K_k v_k]^T \\
&= (I - K_k H_k) E(\varepsilon_{x,k-1}, \varepsilon_{x,k-1}^T)(I - K_k H_k)^T + K_k E(v_k, v_k^T) K_k^T \\
&= (I - K_k H_k) P_{k-1} (I - K_k H_k)^T + K_k R_k K_k^T = P_k
\end{aligned} \tag{37}$$

when we utilize the properties that the measurement noise v_k is uncorrelated with the prior estimation error ε_{k-1} (i.e., $E[\varepsilon_{x,k-1}, v_k] = 0$, $E[v_k, \varepsilon_{x,k-1}^T] = 0$, $E[v_k, v_k^T] = R_k$, and $E[\varepsilon_{x,k-1}, \varepsilon_{x,k-1}^T] = P_{k-1}$).

The state estimation error is defined as:

$$\varepsilon_{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}}_k = \begin{bmatrix} \varepsilon_{1,k} \\ \varepsilon_{12k} \\ \vdots \\ \varepsilon_{n,k} \end{bmatrix} = \begin{bmatrix} x_1 - \hat{x}_{1,k} \\ x_2 - \hat{x}_{2,k} \\ \vdots \\ x_n - \hat{x}_{n,k} \end{bmatrix}$$

Thus, the gain K_k that minimizes the sum of the variances of the estimation-error covariance P_k is obtained as follows:

$$\begin{aligned} G_k &= E[(x_1 - \hat{x}_1)^2] + \cdots + E[(x_k - \hat{x}_k)^2] \\ &= E[\varepsilon_{1,k}^2] + E[\varepsilon_{2,k}^2] + \ldots E[\varepsilon_{1,k}^2] \\ &= \mathrm{Tr}(P_k) \\ &= \mathrm{Tr}\left[(I - K_k H_k) P_{k-1} (I - K_k H_k)^T\right] + \mathrm{Tr}\left(K_k R_k K_k^T\right) \\ &= \mathrm{Tr}(P_{k-1}) - \mathrm{Tr}(P_{k-1} H_k^T K_k^T) - \mathrm{Tr}(K_k H_k P_{k-1}) + \mathrm{Tr}(K_k H_k P_{k-1} H_k^T K_k^T) + \mathrm{Tr}(K_k R_k K_k^T) \\ &= \mathrm{Tr}(P_{k-1}) - 2\mathrm{Tr}(K_k H_k P_{k-1}) + \mathrm{Tr}(K_k H_k P_{k-1} H_k^T K_k^T) + \mathrm{Tr}(K_k R_k K_k^T). \end{aligned} \tag{38}$$

Here, since P_{k-1} is symmetric, $Tr(K_k R_k P_{k-1}) = Tr(P_{k-1} H_k^T K_k^T)$, $x_k \in \mathbb{R}^n$ is the actual state value at time k, and $\hat{x}_k \in \mathbb{R}^n$ is the state estimation.

To find the value of K_k which will minimize our cost function G_k, we compute the gradient of G_k with respect to K_k:

$$\begin{aligned} \nabla_{K_k} G &= -2P_{k-1}^T H_k^T + 2K_k H_k P_{k-1} H_k^T + 2K_k R_k \\ &= -2P_{k-1} H_k^T + 2K_k H_k P_{k-1} H_k^T + 2K_k R_k \\ &= 2(I - K_k H_k) P_{k-1} (-H_k^T) + 2K_k R_k \end{aligned}$$

Setting the gradient to zero yields:

$$\begin{aligned} & 2(I - K_k H_k) P_{k-1} (-H_k^T) + 2K_k R_k = 0 \\ \Leftrightarrow & K_k R_k = (I - K_k H_k) P_{k-1} H_k^T = P_{k-1} H_k^T - K_k H_k P_{k-1} H_k^T \\ \Leftrightarrow & K_k R_k + K_k H_k P_{k-1} H_k^T = P_{k-1} H_k^T \\ \Leftrightarrow & K_k (R_k + H_k P_{k-1} H_k^T) = P_{k-1} H_k^T \\ \Leftrightarrow & K_k = P_{k-1} H_k^T (R_k + H_k P_{k-1} H_k^T)^{-1} \end{aligned}$$

which matches the Equation (32) defined the gain of the recursive least squares. □

Remark 3.1. The expression (37), often called the *Joseph form*, is the numerically robust alternative to Equation (33). As mentioned earlier, although P_k preserves symmetric positive definiteness in exact arithmetic, finite-precision implementations of the simplified recursion in Equation (33) may accumulate rounding errors that leave P_k no longer symmetric positive definite. When we take a closer look at the *Joseph form*, because it is the sum of two quadratic forms $(I - K_k H_k)P_{k-1}(I - K_k H_k)^T$ and $K_k R_k K_k^T$, the resulting covariance matrix P_k is mathematically guaranteed to remain symmetric and positive semi-definite, making it the preferred formulation for practical software implementation.

Remark 3.2. The estimation-error covariance P_k is defined as:

$$P_k = E[\varepsilon_{\mathbf{x}} \varepsilon_{\mathbf{x}}^T] = \begin{bmatrix} E[\varepsilon_{1,k}^2] & \dots & E[\varepsilon_{1,k}\varepsilon_{n,k}] \\ \vdots & \ddots & \vdots \\ E[\varepsilon_{n,k}\varepsilon_{1,k}] & \dots & E[\varepsilon_{n,k}^2] \end{bmatrix}$$

Taking the trace of P_k, we have:

$$\mathrm{Tr}(P_k) = E[\varepsilon_{1,k}^2] + \cdots + E[\varepsilon_{n,k}^2]$$

which is equal to the sum of the variance of the estimation error.

3.2.2 Example

To illustrate how the recursive least squares method actually works, we consider a simple problem of estimating the initial position, speed, and acceleration of an ice cream truck which is assumed to be constant. The position of the truck is measured by a noisy sensor, and these noisy measurement data will be sent back to the truck's base center.

Table 1 shows the simulated position measurements of the ice cream truck.

Table 1: Position measurements of the ice cream truck

Time Step (k)	Elapsed time t (s)	Measured position y (m)
1	1	20
2	2	24.0680
3	3	25.3694
4	4	29.5424
5	5	36.1145
⋮	⋮	⋮

The basic kinematic equation used to describe the noisy position of the truck at each time instant k, is given as:

$$y_k = y_0 + v_0 t + \frac{1}{2} a t^2 + w_k$$

where $y_k \in \mathbb{R}$ is the position of the truck at time k, and y_0, v_0, and a are the initial position, initial velocity, and initial acceleration of the truck that we are trying to estimate, respectively, and $w_k \sim \mathcal{N}(0,1)$ is additive noise. The above kinematic equation can be also be rewritten in matrix form:

$$y_k = \begin{bmatrix} 1 & t & \frac{1}{2}t^2 \end{bmatrix} \begin{bmatrix} y_0 \\ v_0 \\ a \end{bmatrix} + w_k$$

In this example, we denote the initial position, speed, and acceleration whose values we are trying to estimate as $\hat{\mathbf{x}}$. The state equation is given as:

$$\mathbf{x}_k = \mathbf{x}_{k-1} = \text{constant}$$

Often, the state equation which describe how the value of the previous state influence the value of the next state is omitted in recursive least squares since the objective is to estimate a constant vector.

Our measurement matrix which is denoted as H_k is given as $\begin{bmatrix} 1 & t & \frac{1}{2}t^2 \end{bmatrix}$. Since the noise w_k has a variance of 1, our measurement noise covariance matrix, which we will denote as R will also be a constant 1. Let the initial state estimate $\hat{\mathbf{x}}_0$ and the error covariance P_0 be defined as follows:

$$\hat{\mathbf{x}}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$$

$$P_0 = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}$$

Notice that the diagonal elements of the error covariance matrix P_0 have a large value which reflects our lack of confidence in our initial guess. Using the recursive least squares method, we compute our estimate in each time step k through three steps: first, calculating the recursive gain K_k that weighs the estimated value against the incoming measurement data from the base station using the measurement matrix H_k and the error covariance matrix P_k, then updating the estimation using the gain calculated in the first step and updating the error covariance matrix P_k using the recursive gain K_k.

At $k = 1$, before the true position measurement for the first second is known, we can already estimate the value of the state, which is often a redundant step in recursive least squares as mentioned earlier, and compute the gain K_1. This gain controls how much we shift our estimate toward the new measurement versus how much we stay with our current kinematic model. In the scalar case, a large K means that we trust the measurement more, while a small K means that we trust our prior estimate more. The gain K_1 in the first time instant is computed as follows:

$$H_1 = \begin{bmatrix} 1 & 1 & \frac{1}{2} \end{bmatrix}, P_0 = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}, R = 1$$

$$\implies K_1 = P_0 H_1^T \left(R + H_1 P_0 H_1^T \right)^{-1} \approx \begin{bmatrix} 0.4425 \\ 0.4425 \\ 0.2212 \end{bmatrix}$$

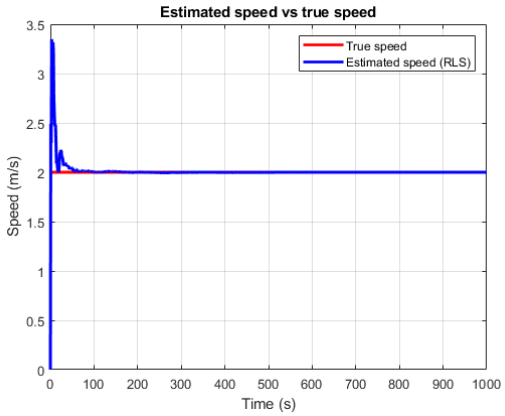
When the measurement $y_1 = 20$ arrives, we update our state estimate and its error covariance using the gain K_1 computed previously:

$$\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_0 + K_1(y_1 - H_1\hat{\mathbf{x}}_0) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.4425 \\ 0.4425 \\ 0.2212 \end{bmatrix} \left(20 - \begin{bmatrix} 1 & 1 & \frac{1}{2} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 8.85 \\ 8.85 \\ 4.424 \end{bmatrix}$$

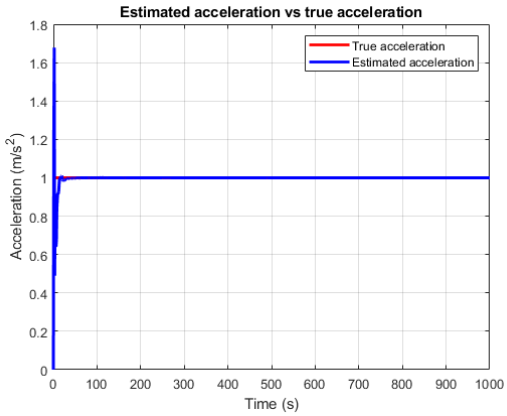
$$P_1 = (I - K_1 H_1) P_0 = \begin{bmatrix} 55.75 & -44.25 & -22.125 \\ -44.25 & 55.75 & -22.125 \\ -22.12 & -22.12 & 88.94 \end{bmatrix}$$

At this step, the first iteration of the algorithm is done. We repeat the same procedure for later iterations. Figure 1 below shows the estimation of the initial position, speed and acceleration of the ice cream truck using the recursive least squares method, along with the true position, speed and acceleration values shown in each sub figure. Initially, the estimates change considerably since the amount of data at the beginning is limited. After getting more measurements data, the estimation values converge to the true value efficiently. However, the eigenvalues of the error covariance matrix P_k are decreasing over time. Then P will become arbitrary small in some directions. Thus, the ability to adapt of these parameters in those directions will become slower and therefore, the estimation will become less efficient (Sastry & Bodson, 1989). The MATLAB code will be shown in the Appendix 1.

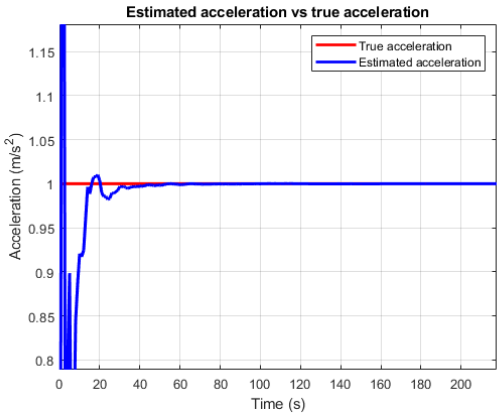
(a) Position estimate.



(b) Speed estimate.



(c) Acceleration estimate.



(d) Acceleration estimate (zoom in).

Figure 1: Position, speed, and acceleration estimates of the truck compared with their true values.

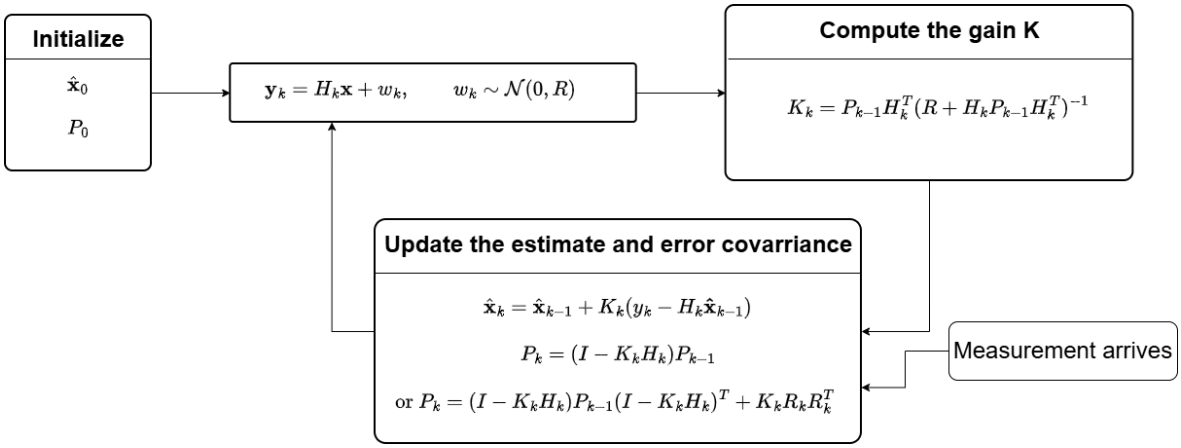


Figure 2: Recursive Least Squares process.

To conclude, the overall process of the recursive least squares method is summarized in Figure2. The method begins with an initialization step, in which an arbitrary starting estimate

is chosen along with an initial covariance P_0 that reflects our uncertainty in that guess. It then computes the gain K_k, the central quantity in recursive least squares, and as we will see, also in the Kalman filter. The gain determines how much weight is placed on the new measurement relative to the current prediction, based on their respective confidence levels. This gain is then used to update both the state estimate and the error covariance, which is carried forward as the starting point for the next iteration. The cycle of computing the gain, updating the estimate and error covariance repeats until a stopping criterion is met.

4 Kalman filtering

In this section, we go through the detailed derivation of the Kalman filter based on the recursive least squares method derived in section 3.2. First, we will introduce the general equations of the discrete-time state space model and a short example to help readers interpret the notation. After that, we will go through two important definitions, which are the priori and posteriori state estimate. The detailed derivation of the Kalman filter is introduced next. And at the end of this section, there is an example in which we extend our tracking problem of the ice cream truck that we have gone through in section 3.2.2.

4.1 Introduction on general equations

The discrete-time state-space model of a dynamical system is given as:

$$\mathbf{x}_k = F_{k-1}\mathbf{x}_{k-1} + G_{k-1}\mathbf{u}_{k-1} + w_{k-1}, \tag{39}$$

$$\mathbf{y}_k = H_k\mathbf{x}_k + v_k. \tag{40}$$

where $k \in \mathbb{N}$ denotes the discrete-time index, $\mathbf{x}_k \in \mathbb{R}^n$ is the state vector, $\mathbf{u}_k \in \mathbb{R}^m$ is the control input, $\mathbf{y}_k \in \mathbb{R}^l$ is the measurement vector, $F_k \in \mathbb{R}^{n\times n}$ is the state transition matrix which describe how the system changes, $G_k \in \mathbb{R}^{n\times m}$ is the control-input matrix, $H_k \in \mathbb{R}^{l\times n}$ is the measurement matrix, $w_k \in \mathbb{R}^n$ is the process noise, and $v_k \in \mathbb{R}^n$ is the measurement noise. We assume that the process noise w_k and v_k has zero mean and the covariance is mutually uncorrelated. In other words,

$$\begin{aligned} E(w_k) = E(v_k) = 0 \\ E(w_k w_j^T) = Q_k \delta_{k-j} \\ E(v_k v_j^T) = R_k \delta_{k-j} \end{aligned}$$

where δ_{k-j} is the Kronecker delta function. $\delta_{k-j} = 0$ if $k \neq j$ and $\delta_{k-j} = 1$ if $k = j$ (Simon, 2006, pp. 124-129), (Chui & Chen, 2017, pp.33). If the system is time-invariant (Constant with respect to time), matrices F_k, G_k, H_k, Q_k, R_k are constant.

Consider an example of tracking the position and speed of an ice cream truck using noisy GPS measurements. The state vector $\mathbf{x}_k$ represents what we are trying to track, which consists of position p_k and speed s_k. Recall the basic kinematic equations,

$$\begin{aligned} p_k &= p_{k-1} + s_{k-1}\Delta t + \frac{1}{2}a_{k-1}\Delta t^2 \\ s_k &= s_{k-1} + a_{k-1}\Delta t \end{aligned}$$

where Δt is the difference between time instant k and $k-1$, and a_k is the acceleration of the truck. Using the kinematic equations mentioned above, we can rewrite it in matrix form as

$$\mathbf{x}_k = \begin{bmatrix} p_k \\ s_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{k-1} \\ s_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} a_{k-1} + w_{k-1} \tag{41}$$

Thus, the state transition matrix is

$$F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

which will describe how the state of the truck evolve from one time instant to the next. The control input matrix in this example is

$$G = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$$

which describes how much acceleration will be added to the movement of the truck. Here, $a_{k-1} \in \mathbb{R}$ is the acceleration of the truck at time instant $k-1$, which will also be the control input vector. The noise w_k can be some effects that we cannot measure. It can be the wind resistance, the inefficiency of the engine, or the slope of the road which may cause certain effects in estimating the speed of the ice cream truck. The noisy GPS measurements can only track the position of the ice cream truck. Thus, the measurement model has the following form:

$$\mathbf{y}_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} p_k \\ s_k \end{bmatrix} + v_k \tag{42}$$

where the measurement matrix is

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

and the measurement noise v_k can be some external effects from the environment or from the GPS sensor itself which can cause some errors in measuring the position of the truck.

4.2 Priori state and posteriori state estimate

Before we look at the actual derivation of the Kalman filtering, let us introduce few more important terminologies. One of the key concepts is the distinction between a priori and a posteriori state estimate.

Definition 4.1. (Simon, 2006, pp.125) A priori state estimate is the estimate of the current

state before incorporating the current measurement. To be more specific, the priori estimation of $\mathbf{x}_k$ is computed by using all the measurements $\mathbf{y}_k$ from time instant 1 to $k-1$, except for the current measurement $\mathbf{y}_k$ which is denoted as $\hat{\mathbf{x}}_k^-$ where the hat notation of $\hat{\mathbf{x}}_k$ indicated as an estimate. The priori state estimate is defined as follows:

$$\hat{\mathbf{x}}_k^- = E[\mathbf{x}_k|\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{k-1}] \tag{43}$$

Definition 4.2. (Simon, 2006, pp.125) A posteriori state estimate is obtained by taking all measurements up to the current measurement y_k which is denoted as $\hat{\mathbf{x}}_k^+$. The hat notation denoted as an estimate and the "$+$" superscript means that our estimation is a posteriori state estimate. The posteriori state estimate is defined as follows;

$$\hat{\mathbf{x}}_k^+ = E[\mathbf{x}_k|\mathbf{y}_1, \mathbf{y_2}, \ldots, \mathbf{y}_k] \tag{44}$$

It is important to note that both priori and posterirori estimates the value of $\mathbf{x}_k$. The difference is that priori does not take the measurement y_k into account while posteriori does not.

4.3 Discrete-time Kalman filtering

4.3.1 Derivation of discrete-time Kalman filtering

In this section, let us move further to the actual derivation of the Kalman filter using all of the above introduced information.

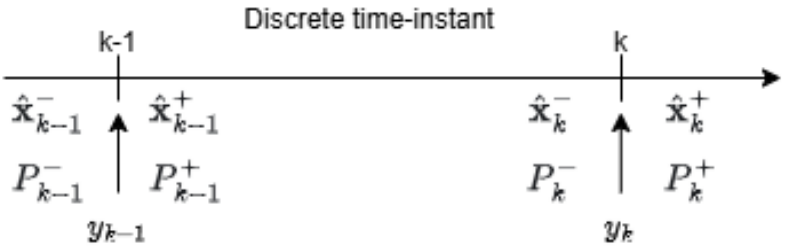


Figure 3: Kalman Filtering updated process.

Figure 3 adapted from (Simon, 2006, pp. 126) shows us how a Kalman filter iteration works in general. At time instant $k-1$ before the measurement data arrives, we calculate the priori state estimation of $\hat{\mathbf{x}}_{k-1}^-$ and its corresponding estimation-error covariance matrix P_{k-1}^-. Once the measurement y_{k-1} arrives, we update our priori estimate to posteriori estimate $\hat{\mathbf{x}}_{k-1}^+$ and the posteriori covariance matrix P_{k-1}^+.

Assume that we choose an initial value for our state estimate denoted as $\hat{\mathbf{x}}_0^+$ which is also our initial posteriori state estimate. The priori state estimate at $k = 1$ is computed using the equation (39). Thus, the priori estimate at the first time instant is:

$$\hat{\mathbf{x}}_1^- = F_0\hat{\mathbf{x}}_0^+ + G_0u_0$$

Note that the process noise does not appear explicitly in the prediction equation. Since the process noise is an unknown random variable with zero mean, its value cannot be predicted. Consequently, only the deterministic part of the system dynamics is used to predict the state. The above equation shows us how to obtain $\hat{\mathbf{x}}_1^-$ from $\hat{\mathbf{x}}_0^+$. Similarly, at time $k = 2$, we obtain the priori state estimate $\hat{\mathbf{x}}_2^-$ using the posteriori calculated previously:

$$\hat{\mathbf{x}}_2^- = F_1\hat{\mathbf{x}}_1^+ + G_1u_1$$

At time instant k^{th}, assume that we have already computed the posteriori state estimate $\hat{\mathbf{x}}_{k-1}^+$, the priori state estimate $\hat{\mathbf{x}}_k^-$ is obtained as follows:

$$\hat{\mathbf{x}}_k^- = F_{k-1}\hat{\mathbf{x}}_{k-1}^+ + G_{k-1}u_{k-1} \tag{45}$$

Next, let us calculate the priori covariance error using the priori estimate of $\hat{\mathbf{x}}_k^-$. By definition, it is calculated as the expected value of the square of the estimation-error. Therefore, we have the following:

$$\begin{aligned}
P_k^- &= E[(\mathbf{x}_k - \hat{\mathbf{x}}_\mathbf{k}^-)(\mathbf{x}_k - \hat{\mathbf{x}}_\mathbf{k}^-)^T] \\
&= E[(F_{k-1}\mathbf{x}_{k-1} + G_{k-1}\mathbf{u}_{k-1} + w_{k-1} - F_{k-1}\hat{\mathbf{x}}_{k-1}^+ - G_{k-1}\mathbf{u}_{k-1})(\dots)^T] \\
&= E[(F_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+) + w_{k-1})(F_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+) + w_{k-1})^T] \\
&= E[F_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T F_{k-1}^T + F_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)w_{k-1}^T \\
&+ w_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T F_{k-1}^T + w_{k-1}w_{k-1}^T] \\
&= F_{k-1}E[(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T]F_{k-1}^T + E[F_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)w_{k-1}^T] \\
&+ E[w_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T F_{k-1}^T] + E[w_{k-1}w_{k-1}^T] \\
&= F_{k-1}P_{k-1}^+F_{k-1}^T + Q_{k-1}
\end{aligned}$$

Given that $E[w_{k-1}w_{k-1}^T] = Q_{k-1}$ is called the process noise covariance matrix and its assumption is mentioned in section 4.1. Thus, at time instant k, the priori covariance error is calculated as:

$$P_k^- = F_{k-1}P_{k-1}^+F_{k-1}^T + Q_{k-1} \tag{46}$$

Equations (45) and (46) show us a general way to calculate the priori state estimate $\hat{\mathbf{x}}^-$ and the priori covariance matrix P^-, also known as the prediction phase in Kalman filter. Once the measurement data arrive at our system, using the recursive least squares method that we have derived previously in section 3.2, we can calculate the posteriori state estimate $\hat{\mathbf{x}}^+$ and the posteriori estimation-error covariance matrix P^+, or the correction phase. Recall the recursive least squares equations:

$$P_k = (I - K_k H_k)P_{k-1}(I - K_k H_k)^T + K_k R_k K_k^T = (I - K_k H_k)P_{k-1} \tag{47}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + K_k(y_k - H_k\hat{\mathbf{x}}_{k-1}) \tag{48}$$

$$K_k = P_{k-1}H_k^T(R_k + H_k P_{k-1}H_k^T)^{-1} \tag{49}$$

Here, $\hat{\mathbf{x}}_{k-1}$ is the estimate of $\mathbf{x}$ before the measurement y_k arrives. Similarly, P_{k-1} is the estimation-error covariance matrix before the measurement y_k arrives. $\hat{\mathbf{x}}_k$ and P_k are the state estimate and the covariance matrix, respectively, after taking y_k into the calculation process. Since $\hat{\mathbf{x}}_{k-1}$ is the estimation of $\hat{\mathbf{x}}$ before the measurement y_k arrives, we can replace $\hat{\mathbf{x}}_{k-1}$ with priori estimate $\hat{\mathbf{x}}_k^-$. Similarly, replace P_{k-1} with the priori covariance estimation error P_k^-, and replace P_k with the posterirori covariance estimation error P_k^+.

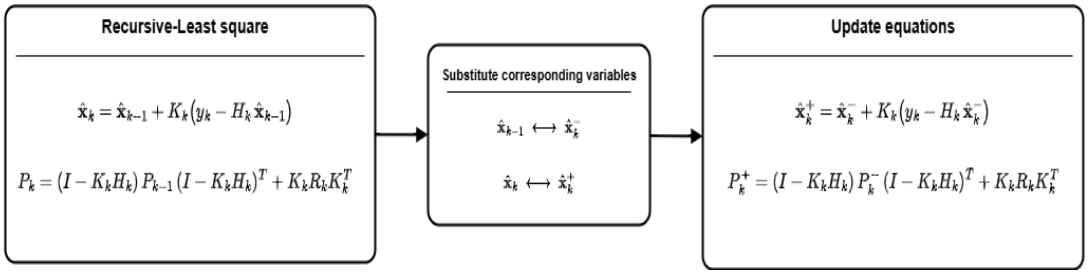


Figure 4: Posteriori estimate equations at time instant k.

The Kalman filter consists of two main phases. The first phase is to make a prediction by using the previous posteriori state estimate and the estimation-error covariance matrix P. For the first time instant, the posteriori state estimate is simply our guess for the initial state, and the estimation-error covariance matrix shows the confidence level corresponding to our initial estimate. In the scalar case, if P is large, it means that we are not confident in our prediction for the initial state. The second phase is to correct our prediction incorporate with the measurement data. Using the equation in Figure 4, we can calculate the posteriori estimation and the posteriori covariance of the error-estimation. The process of making predictions and corrections continues recursively until some stop criteria are met.

One may notice the similarity between the Recursive Least squares method and the Kalman filter. In the Kalman filter, the estimation value of the state vector is recomputed based on

new measurements obtained from external sources. Similarly, the recursive least squares updates its estimation values when new information is obtained. As mentioned earlier in section 3, the recursive least squares, which is the recursive form of the least squares method, is used to derive the Kalman filter. However, it is important to note that the recursive least squares is used to estimate a constant unknown parameter just like the least squares method, whereas the Kalman filter is used to estimate unknown state vector values that vary over time. For instance, if the goal is to estimate a constant unknown variables of a polynomial, either the least squares method or the recursive least squares can be used to fulfill the objective. If we try to estimate a time-varying state vector, the Kalman filter is the choice for that specific problem. The recursive least squares can also be seen as a special case of the Kalman filter where the state transition matrix is the identity matrix and the process noise is zero. With these assumptions, one can easily show that the Kalman filter is equivalence to the recursive least square. Also, there is an interesting property of the discrete-time Kalman filter. We have shown in section 4.1 that the process noise w_k and the measurement noise v_k are zero-mean and their covariance is mutually uncorrelated. The Kalman filter can be seen as the best linear solution without explicitly depending on the Gaussian assumption for the noise process (Simon, 2006, pp. 130).

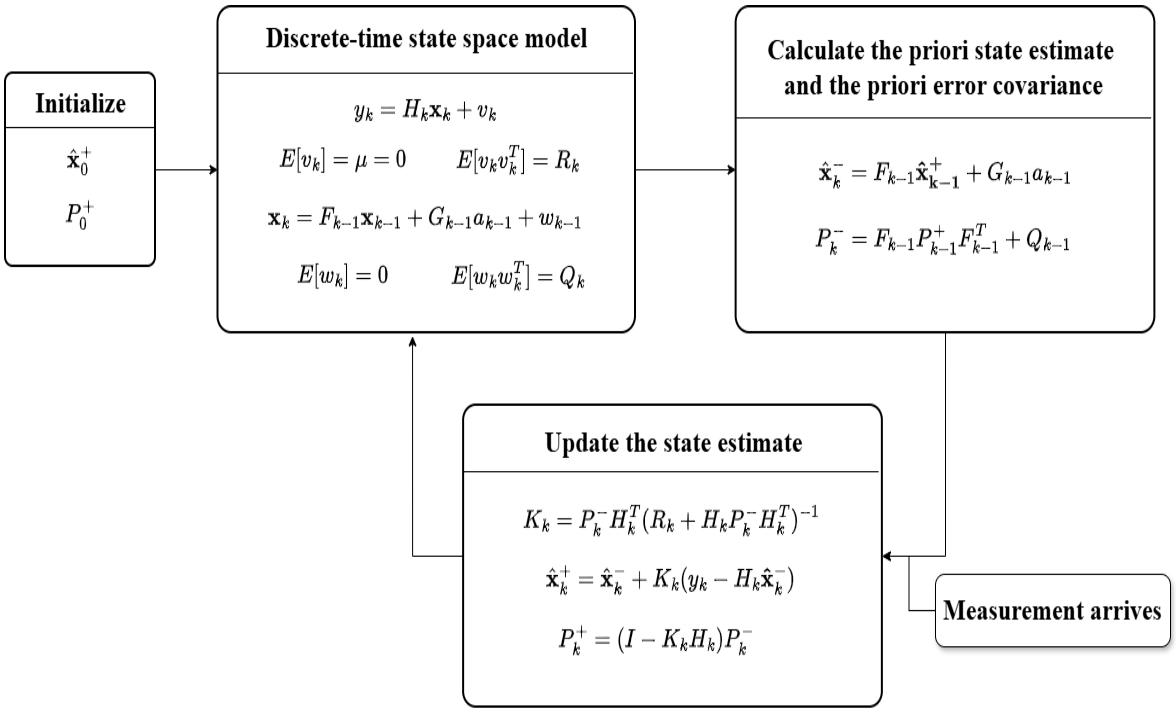


Figure 5: Kalman Filtering process.

To be concluded, Kalman Filtering has two main phases. The first phase is to calculate the priori estimation before the measurement arrives. After our model receives the measurement at time instant k, using the priori estimation that we have calculated in the previous phase, we

can obtain the posteriori estimation. The recursive process of kalman filtering is presented in Figure 5.

4.3.2 Example

Let us extend the example that we have gone through in section 3.2.2. In the previous example, we estimated the initial position and speed of the truck, and its acceleration, which is assumed to be constant. With Kalman filter, we can track the state evolution of the truck from one time instant to the next. Consider the following discrete-time state-space model of the truck:

$$\mathbf{x}_k = \begin{bmatrix} p_k \\ s_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{k-1} \\ s_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} a_{k-1} + w_{k-1}$$

$$E[w_k] = 0, E[w_k w_k^T] = Q_k = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$$

$$\mathbf{y}_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}_k + v_k, \quad v_k \sim \mathcal{N}(0,4)$$

where the state and measurement equations follow the same interpretation as mentioned in section 4.1, and the sampling time is 0.05, which means $\Delta t = 0.05$. Assume that at $k = 0$, $a_0 = 5$, $\hat{\mathbf{x}}_0^+ = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $P_0^+ = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$. It is easy to understand that initially we guess the position and speed of the ice cream truck without knowing anything. Here, the diagonal elements of the error covariance P_0^+ are large compared to diagonal elements of the variance of the state equation. This means we don't have much confidence in our initial prediction. At k = 1, the priori state estimate $\hat{\mathbf{x}}_1^-$ is obtained as follows:

$$\begin{aligned} \hat{\mathbf{x}}_1^- &= F_0 \hat{\mathbf{x}}_0^+ + G_0 a_0 \\ &= \begin{bmatrix} 1 & 0.05 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.00125 \\ 0.05 \end{bmatrix} \times 5 = \begin{bmatrix} 0.00625 \\ 0.25 \end{bmatrix} \end{aligned}$$

The priori estimation at $k = 1$ is computed based on our initial guess and the dynamic model. Next, the priori error covariance P_1^- is calculated as follows:

$$\begin{aligned} P_1^- &= F_0 P_0^+ F_0^T + Q_0 \\ &= \begin{bmatrix} 1 & 0.05 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0.05 & 1 \end{bmatrix} + \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} = \begin{bmatrix} 100.35 & 5 \\ 5 & 100.1 \end{bmatrix} \end{aligned}$$

Notice that every elements of the priori error covariance is large, which means that we are not confident in our prediction of the truck's state before receiving the measurement from the GPS. In the scalar case, the recursive gain is larger if the prediction is not trustable. The

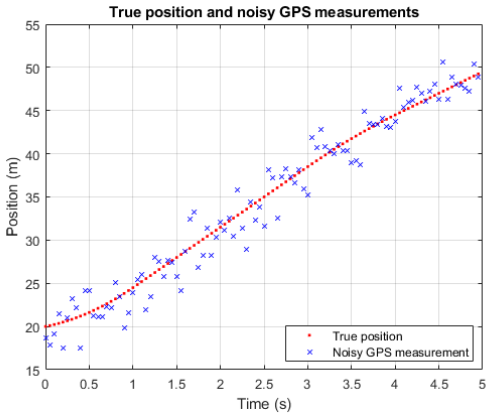
gain K at time instant 1 is:

$$K_1 = P_1^- H_1^T (R_1 + H_1 P_1^- H_1^T)^{-1} = \begin{bmatrix} 0.9617 \\ 0.0479 \end{bmatrix}$$

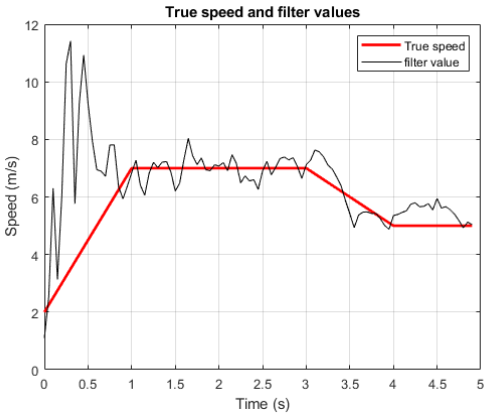
The gain value for the position is close to 1, whereas the gain for the speed is approximately 0.05. This is because the GPS measurement can only track the position, but not the speed of the truck. The first half of the Kalman filter is over. Next, let us compute the posteriori $\hat{\mathbf{x}}_1^+$, P_1^+ after the GPS measurement at time instant $k = 1$ arrives at our system. We assume $\mathbf{y}_1 = 20.5$:

$$\begin{aligned}
&\hat{\mathbf{x}}_1^+ = \hat{\mathbf{x}}_1^- + K_1(y_1 - H_1\hat{\mathbf{x}}_1^-) \\
&= \begin{bmatrix} 0.00625 \\ 0.25 \end{bmatrix} + \begin{bmatrix} 0.9617 \\ 0.0479 \end{bmatrix} (20.5 - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0.00625 \\ 0.25 \end{bmatrix}) = \begin{bmatrix} 19.71505 \\ 1.23165 \end{bmatrix} \\
&P_1^+ = (I - K_1 H_1) P_1^- \\
&= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0.9921 \\ 0.3965 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \right) \begin{bmatrix} 125.1 & 50 \\ 50 & 100.1 \end{bmatrix} = \begin{bmatrix} 3.8467 & 0.1917 \\ 0.1917 & 99.8604 \end{bmatrix}
\end{aligned}$$

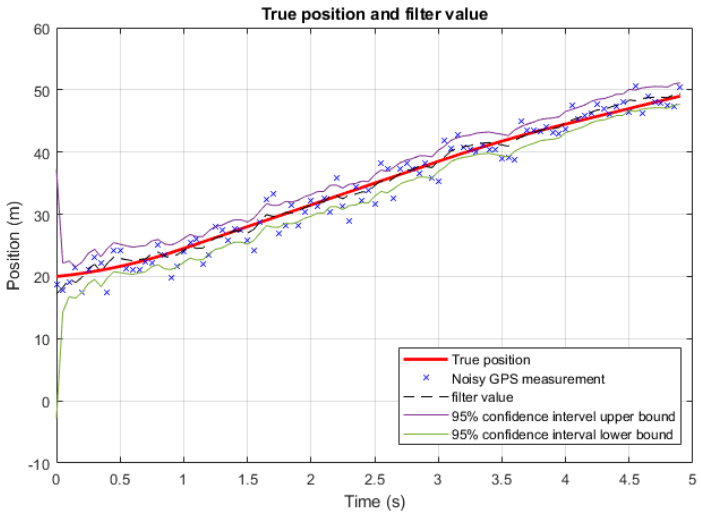
At this stage, the first iteration of the Kalman filter is complete. For the next iterations, we compute the priori state estimate, the priori error covariance, and use the measurement data to update the estimate value. Figure 6 below shows the position and speed estimates using one hundred simulated data. The power of Kalman filter can be observed from figure 6b. At the first 1 second, the speed of the truck is increasing. The filter value is not correct at the very beginning when the amount of measurement data is limited. From 1 to 3 seconds, the speed of the car remains constant, and the filter estimate converges closer to the true value as more data is collected. From 3 to 4 seconds, the speed of the truck is decreasing, and the filter is able to adapt to this change. MATLAB code for the Kalman filter is shown in Appendix 2.



(a) Noisy GPS measurement and true position.



(b) Speed estimate.



(c) Position estimate

Figure 6: Dynamic position and speed estimates of the truck compared with their true values.

4.4 Extended Kalman Filtering

In real-world applications, almost everything is described using non-linear models. For example, when a rocket is launched into space, air drag must be taken into account when formulating the model for its motion. Similarly, when a car is tested on a track, one must also consider the friction from the road, which influences the car's velocity. Another source of nonlinearity comes from the measurements themselves. For instance, consider a radar tracking the position of an object. The distance from the radar to the object can be computed using the Pythagorean theorem, as the square root of the sum of the squared altitude and the squared ground distance from the radar to the object. Because this relationship involves a square root of the state variables rather than a direct linear combination of them, the measurement model itself is nonlinear, even when the underlying motion is linear. So the question

is how do we apply Kalman filtering in a non-linear world? The key is linearization. The Extended Kalman Filter uses a first-order Taylor approximation to linearize the non-linear dynamic system around the current state estimate (Simon, 2006, pp. 396). This transforms the non-linear system into a linear one at each time step, allowing the ordinary Kalman filter to be applied to estimate the state. Because the linearization is recomputed at every new estimate rather than around a single fixed point, the Extended Kalman Filter can adapt to the changing shape of the non-linear system as new information arrives. We will discuss more on linearization in the next section.

4.4.1 Discrete-time extended Kalman filter

In this section, let us derive the Extended Kalman Filter using linearization and the Kalman filter we derived in the previous section. The focus of this chapter is only on the discrete-time version of the EKF, since it is widely used in digital computers. As there might not be enough computational power to implement the continuous version of the EKF, the discrete version is more favorable in real-life applications.

Let us introduce the system model, adapted from (Simon, 2006, pp. 397).

$$x_k = f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1}) \tag{50}$$
$$y_k = h_k(x_k, v_k) \tag{51}$$
$$w_k \sim \mathcal{N}(0, Q_k), E(w_k w_k^T) = Q_k$$
$$v_k \sim \mathcal{N}(0, R_k), E(v_k v_k^T) = R_k$$

where

- k is a discrete-time instant, taking the form $0, \Delta t, 2\Delta t, \ldots, k\Delta t$, and Δt is the sampling period of the system. If $\Delta t = 1$, the discrete-time instants are given by $k = 0, 1, 2, \ldots$;
- $x_k \in \mathbb{R}^n$ is the state vector at discrete-time instant k;
- $u_k \in \mathbb{R}^m$ is the input vector at discrete-time instant k;
- $w_k \in \mathbb{R}^n$ is the noise from the dynamic model. The noise w_k is assumed to be Gaussian with zero mean and covariance matrix Q_k;
- $f_k : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$ is the nonlinear vector function of the dynamic model.
- $y_k \in \mathbb{R}^r$ is the r-dimensional measurement vector;
- $h_k : \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^r$ is the non-linear function that models the relationship between the measurement vector and the state of the system;
- $v_k \in \mathbb{R}^r$ is the measurement noise, which is assumed to be Gaussian with zero mean

and covariance matrix R_k. The dynamic model noise w_k and the measurement noise v_k are assumed to be white noise and uncorrelated.

Before the actual derivation, let us recall the definitions of the a priori and a posteriori estimates, which are defined in more detail in section 4.2. In general, a priori state estimate $\hat{x}_k^-$ is calculated using all measurements up to time $k-1$, while a posteriori state estimate $\hat{x}_k^+$ is calculated by also incorporating the measurement that arrives at time instant k.

The first step in deriving the EKF is to linearize the state equation using a first-order Taylor series expansion around the a posteriori state estimate $\hat{x}_{k-1}^+$ at time instant $k-1$, which is assumed to be known. Below is the Taylor series expansion of the state equation given in (50), with $x_{k-1} = \hat{x}_{k-1}^+$, $u_{k-1} = u_{k-1}^\star$, and $w_{k-1} = 0$.

$$\begin{aligned} x_k &\approx f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}^\star, 0) + A_{k-1}(x_{k-1} - \hat{x}_{k-1}^+) + L_{k-1}w_{k-1} \\ &\approx A_{k-1}x_{k-1} + [f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}^\star, 0) - A_{k-1}\hat{x}_{k-1}^+] + L_{k-1}w_{k-1} \\ &\approx A_{k-1}x_{k-1} + s_{k-1} + \bar{w}_{k-1} \end{aligned} \tag{52}$$

Let us carefully define the notation in Equation (52). Here, $s_{k-1} = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}^\star, 0) - A_{k-1}\hat{x}_{k-1}^+$ which contains all known values like $\hat{x}_{k-1}^+, u_{k-1}^\star, w_{k-1} = 0$, $\bar{w}_{k-1} = L_{k-1}w_{k-1} \sim \mathcal{N}(0, L_{k-1}Q_{k-1}L_{k-1}^T)$, A_{k-1} is the Jacobian matrix of f_{k-1}, and L_{k-1} is the Jacobian matrix of f_{k-1} with respect to w_{k-1}.

$$\begin{aligned} A_{k-1} &= \frac{\partial f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1})}{\partial x_{k-1}}\Big|_{x_{k-1}=\hat{x}_{k-1}^+, u_{k-1}=u_{k-1}^\star, w_{k-1}=0} \\ &= \begin{bmatrix} \frac{\partial f_{1,k-1}}{\partial x_{1,k-1}} & \frac{\partial f_{1,k-1}}{\partial x_{2,k-1}} & \cdots & \frac{\partial f_{1,k-1}}{\partial x_{n,k-1}} \\ \frac{\partial f_{2,k-1}}{\partial x_{1,k-1}} & \frac{\partial f_{2,k-1}}{\partial x_{2,k-1}} & \cdots & \frac{\partial f_{2,k-1}}{\partial x_{n,k-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{n,k-1}}{\partial x_{1,k-1}} & \frac{\partial f_{n,k-1}}{\partial x_{2,k-1}} & \cdots & \frac{\partial f_{n,k-1}}{\partial x_{n,k-1}} \end{bmatrix} \\ L_{k-1} &= \frac{\partial f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1})}{\partial w_{k-1}}\Big|_{x_{k-1}=\hat{x}_{k-1}^+, u_{k-1}=u_{k-1}^\star, w_{k-1}=0} \end{aligned}$$

Similarly, let's linearize the measurement model which is given in Equation (51). We assume that we have known the value of the priori estimate $\hat{x}_k^-$ at time instant k and $v_k = 0$. Therefore,

we have the followings:

$$\begin{aligned} y_k &\approx h_k(\hat{x}_k^-, 0) + C_k(x_k - \hat{x}_k^-) + M_k v_k \\ &= h_k(\hat{x}_k^-, 0) + C_k x_k - C_k \hat{x}_k^- + M_k v_k \\ &= C_k x_k + [h_k(\hat{x}_k^-, 0) - C_k \hat{x}_k^-] + M_k v_k \\ &= C_k x_k + z_k + \bar{v}_k \end{aligned} \tag{53}$$

where C_k is the Jacobian matrix of h_k with respect to x_k, and M_k is the Jacobian matrix of h_k with respect to v_k. Here, $z_k = h_k(\hat{x}_k^-, 0) - C_k \hat{x}_k^-$ is the difference between the measurement function evaluated at the a priori state estimate (with $v_k = 0$) and $C_k \hat{x}_k^-$, and $\bar{v}_k = M_k v_k \sim \mathcal{N}(0, M_k R_k M_k^T)$.

$$\begin{aligned} C_k &= \frac{\partial h_k(x_k, v_k)}{\partial x_k}\Big|_{x_k = \hat{x}_k^-, v_k = 0} \\ &= \begin{bmatrix} \frac{\partial h_{1,k}}{\partial x_{1,k}} & \frac{\partial h_{1,k}}{\partial x_{2,k}} & \cdots & \frac{\partial h_{1,k}}{\partial x_{n,k}} \\ \frac{\partial h_{2,k}}{\partial x_{1,k}} & \frac{\partial h_{2,k}}{\partial x_{2,k}} & \cdots & \frac{\partial h_{2,k}}{\partial x_{n,k}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_{n,k}}{\partial x_{1,k}} & \frac{\partial h_{n,k}}{\partial x_{2,k}} & \cdots & \frac{\partial h_{n,k}}{\partial x_{n,k}} \end{bmatrix} \\ M_k &= \frac{\partial h_k(x_k, v_k)}{\partial v_k}\Big|_{x_k = \hat{x}_k^-, v_k = 0} \end{aligned}$$

To conclude, from Equations (52) and (53), we obtain a system of linearized equations to which we can apply the standard Kalman filter derived in the previous section. Let us summarize the linearized state and measurement equations:

$$x_k = A_{k-1} x_{k-1} + s_{k-1} + \bar{w}_{k-1} \tag{54}$$

$$y_k = C_k x_k + z_k + \bar{v}_k \tag{55}$$

$$\begin{aligned} E(\bar{w}_k) = E(\bar{v}_k) &= 0 \\ E(\bar{w}_k \bar{w}_k^T) &= L_k Q_k L_k^T \\ E(\bar{v}_k \bar{v}_k^T) &= M_k R_k M_k^T \end{aligned} \tag{56}$$

Initially, we choose an arbitrary value for our a posteriori state estimate $\hat{x}_{k-1}^+$ and the error covariance estimate P_{k-1}^+ associated with it. If P_{k-1}^+ is large, this indicates that the chosen estimate is not reliable, and vice versa, a small P_{k-1}^+ indicates high confidence in the estimate. By using Equation (50), we can obtain the priori estimate of the state at time instant k. Notice

that we use the full nonlinear function f_{k-1}, rather than its linear approximation computed in Equation (52), to compute the a priori estimate. This keeps the point estimate as accurate as possible, since the linear approximation is used only in calculating the priori error covariance estimate P_k^-, not the estimate of the state itself.

$$\hat{x}_k^- = f_{k-1}(x_{k-1} = \hat{x}_{k-1}^+, u_{k-1}, w_{k-1} = 0) \tag{57}$$

Next, let's calculate the priori error covariance estimate at time instant k by using Equation (46) woth the help of the linearized state estimate from Equation (52). As mentioned in the previous section, the priori error covariance estimate is defined as the expected value of the squared error. Here, the error is defined as the difference between the true state x_k and the priori state estimate x_k^-.

$$\begin{aligned}
P_k^- &= E[(\mathbf{x}_k - \hat{\mathbf{x}}_\mathbf{k}^-)(\mathbf{x}_k - \hat{\mathbf{x}}_\mathbf{k}^-)^T] \\
&= E[(f_{k-1}(x_{k-1}, u_{k-1}, 0) - f_{k-1}(\hat{\mathbf{x}}_{k-1}^+, u_{k-1}, 0))(\dots)^T] \\
&\approx E[(A_{k-1}x_{k-1} + s_{k-1} + \bar{w}_{k-1} - A_{k-1}\hat{\mathbf{x}}_{k-1}^+ - s_{k-1})(\dots)^T] \\
&= E[(A_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+) + \bar{w}_{k-1})(A_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+) + \bar{w}_{k-1})^T] \\
&= E[A_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T A_{k-1}^T + A_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)\bar{w}_{k-1}^T \\
&+ \bar{w}_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T A_{k-1}^T + \bar{w}_{k-1}\bar{w}_{k-1}^T] \\
&= A_{k-1}E[(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T]A_{k-1}^T + E[A_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)\bar{w}_{k-1}^T] \\
&+ E[\bar{w}_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T A_{k-1}^T] + E(\bar{w}_{k-1}\bar{w}_{k-1}^T) \\
&= A_{k-1}P_{k-1}^+ A_{k-1}^T + L_{k-1}Q_{k-1}L_{k-1}^T
\end{aligned} \tag{58}$$

Just a little justification, since the noise and the error estimation $(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)$ is independent with each other, $E[A_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)w_{k-1}^T] = E[w_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}^+)^T A_{k-1}^T] = 0$.

Now, we have all the priori estimates at time instant k. The next step is to calculate the gain K_k using Equation (49).

$$K_k = P_k^- C_k^T (C_k P_k^- C_k^T + M_k R_k M_k^T)^{-1} \tag{59}$$

One may notice that the difference between Equations (49) and (59) lies in the covariance matrix. In the ordinary Kalman filter, the covariance of the measurement noise v_k in Equation (40) is simply R_k. After linearizing the nonlinear measurement model, the noise covariance is no longer simply R_k, but instead takes the more complex form $M_k R_k M_k^T$. The value of K_k or so called the Kalman gain of the Extended Kalman filter has the same meaning as in the ordinary Kalman filter which is defined in the previous section.

Following the prediction step is the correction step, which uses the gain K_k to update our a priori estimate to the a posteriori estimate at time instant k. Substituting the definition of $z_k = h_k(\hat{x}_k^-, 0) - C_k\hat{x}_k^-$ from Equation (53), and simplifying, we obtain the updated equations presented below:

$$\begin{aligned}
\hat{x}_k^+ &= \hat{x}_k^- + K_k(y_k - C_k\hat{x}_k^- + C_k\hat{x}_k^- - h_k(\hat{x}_k^-, 0)) \\
&= \hat{x}_k^- + K_k[y_k - h_k(\hat{x}_k^-, 0)] \\
P_k^+ &= (I - K_kC_k)P_k^- \\
P_k^+ &= (I - K_kC_k)P_k^-(I - K_kC_k)^T + K_kM_kR_kM_k^TK_k^T
\end{aligned} \tag{60}$$

The term $y_k - h_k(\hat{x}_k^-, 0)$ is called the measurement residual, which measures the gap between what we actually observed, y_k, and what we estimated using the a priori state estimate $\hat{x}_k^-$. There are also two ways to calculate the a posteriori error covariance P_k^+. The first expression is the standard, simplified covariance update, which we defined in Section 3.2. The second, known as the *Joseph form*, is mathematically equivalent when K_k is computed exactly. For a more detailed explanation of why there are two forms of the error covariance, see Section 3.2. At this stage, we have finished the first iteration of the Extended Kalman Filter. Using the same logic, we can estimate the state even when the state equation, the measurement equation, or both are nonlinear. The only difference between the Extended Kalman Filter and the ordinary Kalman filter is the linearization step, which transforms the nonlinear equations into linear ones at the point of interest.

5 EM algorithm

This section presents a brief introduction and a short derivation of the smoother equations, which is also known as the estimation step or the E-step of the Expectation–Maximization (EM) algorithm. Our objective is to introduce to the reader an algorithm that determines the covariance noise matrix based on observed data rather than by experience or assumptions of individuals.

5.1 Motivation

In the Kalman filter, the covariance noise matrix is determined purely from experience or by assumptions without any mathematical derivation. With poor tuning for the value of the covariance matrix in our initial assumption, the estimation value will not converge to the actual value. Fine tuning the covariance matrix is not an easy task as there can be many effects that influence the value of the covariance matrix. The sensor can be degraded over time, which causes the measurements to drift away from the mean. Or even the state space model does not take into account other factors that can alter the reading of the sensor. So, how do we overcome this problem? One method of approximating the value for the process and measurement noise covariance matrix is to use the EM algorithm. Work in fields such as economics, environmental science, and statistics relies heavily on data to make predictions, draw conclusions, or filter out noisy values. However, complete data are not always available or ready to be used immediately. For example, we want to estimate the average lifetime of a light bulb. To do so, we collect 1000 samples of data. Some light bulbs only survive for 3 years. Others can survive 5 years, and some can last longer than that. If the testing period is limited - say, it ends after 5 years - how do we estimate the average lifetime of the light bulb? An idea is to estimate the missing data and apply maximum likelihood estimation to determine the mean lifetime. This combination of estimation and maximization is, in fact, exactly what the EM algorithm does.

5.2 Prediction, filter, and smoother values

Before introducing the EM algorithm, let us briefly discuss the difference between predicting, filtering, and smoothing. To make this section easy to interpret, we consider an example in which we want to estimate an unknown value at time instant t given data up to time s. If t is larger than s, the process of estimating unknown variables is called prediction. If s is equal to t, this is known as filtering. Just like in the Kalman filter, we use the latest measurement data to correct our priori estimation. If s is larger than t, this is called smoothing. Smoothing means we use all the available data to compute the previous estimate. In this section, we denote $\hat{x}_t^n$ as the smoother value at time t and P_t^n as the smoother covariance matrix at time t.

5.3 EM algorithm

5.3.1 How does the EM algorithm work?

Let us denote $\Theta = \{\hat{x}_0, P_0, F, H, Q, R\}$ as the parameter vector consisting of the initial state estimate $\hat{x}_0$, the initial error-estimation covariance matrix P_0, the state transition matrix F, the measurement matrix H, the process noise covariance matrix Q and the measurement noise covariance matrix R. Under the Gaussian assumption, what we are trying to do is find the value of Θ, which will maximize the likelihood of observing the complete data. Here, the complete data are the combination of the measurement value y_t and the unobserved state value x_t. The complete data likelihood presented in (Shumway & Stoffer, 2017, pp. 340) is written as:

$$\begin{aligned} -2\ln_{X,Y}(\Theta) = \ln|P_0| + (x_0 - \hat{x}_0)^T P_0^{-1}(x_0 - \hat{x}_0) \\ + n\ln|Q| + \sum_{t=1}^{n}(x_t - Fx_{t-1})^T Q^{-1}(x_t - Fx_{t-1}) \\ + n\ln|R| + \sum_{t=1}^{n}(y_t - Hx_t)^T R^{-1}(y_t - Hx_t) \end{aligned}$$

where $X = \{x_1, \ldots, x_n\}$ are the unobserved state values, and $Y = \{y_1, \ldots, y_n\}$ is a collection of known measurement values.

The problem arises because of the absence of complete data. However, we can compute the conditional distribution of the unobserved states given the current value of the parameter vector Θ_{old} and the measurement data Y. What it means is that we will approximate the mean and covariance of the unobserved data X given the current parameter vector Θ and the measurement values Y. This idea will be shown clearly in section 5.3.2. By doing this, we have a general picture of how the unobserved data are spreading over the space. This is what the estimation step or the E-step does. Once the conditional distribution of the unobserved data is known, we can obtain the MLEs easily under Gaussian assumptions.

5.3.2 EM algorithm in Kalman filter

We now first briefly introduce the equations for the EM algorithm used in the Kalman filter, and go through the derivation of the E-step in the EM algorithm. The maximization step is lengthy to derive. For a more detailed proof and derivation of the EM algorithm, we recommend to curious readers the proof and derivation mentioned in (Shumway & Stoffer, 1982). This paper shows the general idea of EM algorithms, the needs of this algorithm and builds the foundation for the derivation of such an algorithm. A newer version of the original paper (Shumway & Stoffer, 2017), and (Bishop, 2016, pp.635 643) also contain

similar proof. The detailed derivation of Kalman smoother or the Rauch-Tung-Striebel (RTS) smoother which is also a crucial part of the E-step in the EM algorithm is presented in (Murphy, 2012, pp.645).

Recall the state space model that we have defined in (39) and (40), given $t = \{1,2,\ldots,n-1,n\}$ with an initial value for the state vector $\hat{x}_0^+$ and the posteriori covariance P_0^+, the Kalman filter is defined as:

$$\begin{aligned}
\hat{x}_t^- &= F_{t-1}\hat{x}_{t-1}^+ + G_{t-1}u_{t-1} \\
P_t^- &= F_{t-1}P_{t-1}^+F_{t-1}^T + Q_{t-1} \\
K_t &= P_t^-H_t^T(R_t + H_tP_t^-H_t^T)^{-1} \\
\hat{x}_t^+ &= \hat{x}_t^- + K_t(y_t - H_t\hat{x}_t^-) \\
P_t^+ &= (I - K_tH_t)P_t^-(I - K_tH_t)^T + K_tR_tK_t^T
\end{aligned}$$

For $t = \{n, n-1,\ldots,1\}$, the smoother values for the Kalman filter or the E-step of the EM algorithm are:

$$\hat{x}_{t-1}^n = \hat{x}_{t-1}^+ + J_{t-1}(\hat{x}_t^n - \hat{x}_t^-) \tag{61}$$

$$P_{t-1}^n = P_{t-1}^+ + J_{t-1}(P_t^n - P_t^-)J_{t-1}^T \tag{62}$$

$$J_{t-1} = P_{t-1}^+F_t^T(P_t^-)^{-1} \tag{63}$$

and the M-step of the EM algorithm is given as:

$$\begin{aligned}
Q_{new} &= \frac{1}{n}\sum_{t=1}^{n}(E[x_tx_t^T] - E[x_tx_{t-1}^T]F^T - FE[x_{t-1}x_t^+] + FE[x_{t-1}x_{t-1}^T]F^T) \\
&= \frac{1}{n}\sum_{i=1}^{n}P_t^n - P_{t,t-1}^nF^T - FP_{t,t-1}^n + FP_{t-1}^nF^T + (\hat{x}_t^n - F\hat{x}_{t-1}^n)(\hat{x}_t^n - F\hat{x}_{t-1}^n)^T
\end{aligned} \tag{64}$$

$$R_{new} = \frac{1}{n}\sum_{t=1}^{n}[(y_t - Hx_t)(y_t - Hx_t)^T + HP_t^nH^T] \tag{65}$$

where the state transition matrix F, and the measurement matrix H is assumed to be constant. For $t = \{n, n-1,\ldots,2\}$, the covariance between state estimate at index t, and $t-1$ is given as:

$$P_{t-1,t-2}^n = P_{t-1}^+J_{t-2}^T + J_{t-1}(P_{t,t-1}^n - FP_{t-1}^+)J_{t-2}^T \tag{66}$$

$$P_{n,n-1}^n = (I - K_nH)FP_{n-1}^+ \tag{67}$$

The following is a detailed derivation for the Kalman smoother values. The general idea of the proof is similar to that mentioned in (Murphy, 2012, pp.645)

Proof. A simple state space model has the following form:

$$x_{t+1} = F_t x_t + w_t, \quad w_t \sim \mathcal{N}(0, Q), \quad Q = E[w_t w_t^T]$$
$$y_t = H_t x_t + v_t, \quad v_t \sim \mathcal{N}(0, R), \quad R = E[v_t v_t^T]$$

where $F_t \in \mathbb{R}^{n \times n}$ is the state transition matrix, $x_t \in \mathbb{R}^n$ is the state vector, $w_t \in \mathbb{R}^n$ is the process noise, $H_t \in \mathbb{R}^{m \times n}$ is the measurement matrix, $v_t \in \mathbb{R}^m$ is the measurement noise, and $y_t \in \mathbb{R}^m$ is the measurement vector.

The joint conditional distribution of x_t and x_{t+1} given $y_{1:t}$ is:

$$\begin{bmatrix} x_t \\ x_{t+1} \end{bmatrix} \middle| y_{1:t} \sim \mathcal{N}\left(\begin{bmatrix} E(x_t \mid y_{1:t}) \\ E(x_{t+1} \mid y_{1:t}) \end{bmatrix}, \begin{bmatrix} \mathrm{Var}(x_t \mid y_{1:t}) & \mathrm{Cov}(x_t, x_{t+1} \mid y_{1:t}) \\ \mathrm{Cov}(x_{t+1}, x_t \mid y_{1:t}) & \mathrm{Var}(x_{t+1} \mid y_{1:t}) \end{bmatrix} \right)$$

Here, the priori, and posteriori estimation error is defined as:

$$e_{t+1}^- = x_{t+1} - \hat{x}_{t+1}^-, \; e_t^+ = x_t - \hat{x}_t^+$$

which then implies that:

$$\begin{aligned} x_{t+1} - \hat{x}_{t-1}^- &= F_t(x_t - \hat{x}_t^+) + w_t \\ \implies e_{t+1}^- &= F_t e_t^+ + w_t \end{aligned} \tag{68}$$

The conditional variance and covariance is interpreted as follows:

$$Var(x_t|y_{1:t}) = E[(x_t - E[x_t|y_{1:t}])(x_t - E[x_t|y_{1:t}])^T] = E[(x_t - \hat{x}_t^+)(x_t - \hat{x}_t^+)^T] = P_t^+ \tag{69}$$

$$\begin{aligned} &Var(x_{t+1}|y_{1:t}) = E[(x_{t+1} - E[x_{t+1}|y_{1:t}])(x_{t+1} - E[x_{t+1}|y_{1:t}])^T] \\ &= E[(x_{t+1} - \hat{x}_{t+1}^-)(x_{t+1} - \hat{x}_{t-1}^-)] = P_{t+1}^- \end{aligned} \tag{70}$$

$$\begin{aligned} &Cov(x_t, x_{t+1}|y_{1:t}) = E[(x_t - E[x_t|y_{1:t}])(x_{t+1} - E[x_{t+1}|y_{1:t}])^T] \\ &= E[(x_t - \hat{x}_t^+)(x_{t+1} - \hat{x}_{t+1})^-] = E[e_t^+ e_{t+1}^-] = E[e_t^+ (F_t e_t^+ + w_t)^T] \\ &= E[e_t^+ (e_t^+)^T] F_t^T + E[e_t^+ (w_t)^T] = P_t^+ F_t^T \end{aligned} \tag{71}$$

$$Cov(x_{t+1}, x_t|y_{1:t}) = F_t P_t^+ \tag{72}$$

Using the definition of priori and posteriori state estimate in section 4.2, and equations (69), (70), (71), (72), the joint conditional distribution of x_t, and x_{t+1} given the measurement values up to time t can be written as:

$$\begin{bmatrix} x_t \\ x_{t+1} \end{bmatrix} \middle| y_{1:t} \sim \mathcal{N}\left(\begin{bmatrix} \hat{x}_t^+ \\ \hat{x}_{t+1}^- \end{bmatrix}, \begin{bmatrix} P_t^+ & P_t^+ F_t^T \\ F_t P_t^+ & P_{t+1}^- \end{bmatrix} \right)$$

where the conditional mean and covariance are:

$$E[x_t|x_{t+1}, y_{1:t}] = \hat{x}_t^+ + P_t^+ F_t^T (P_{t+1}^-)^{-1}(x_{t+1} - \hat{x}_{t+1}^-) \tag{73}$$
$$Cov(x_t|x_{t+1}, y_{1:t}) = P_t^+ - P_t^+ F_t^T (P_{t+1}^-)^{-1} F_t P_t^+ \tag{74}$$

Using the Markov property, we obtain the following:

$$E[x_t|x_{t+1}, y_{1:t}] = E[x_t|x_{t+1}, y_{1:n}] = \hat{x}_t^+ + P_t^+ F_t^T (P_{t+1}^-)^{-1}(x_{t+1} - \hat{x}_{t+1}^-). \tag{75}$$

Therefore, the Kalman smoother state estimate is interpreted as follows:

$$\begin{aligned} E[x_t|y_{1:n}] &= E[E(x_t|x_{t+1}, y_{1:t})|y_{1:n}] \text{ (by the law of Iterated Expectation)} \\ &= E[\hat{x}_t^+|y_{1:n}] + P_t^+ F_t^T (P_{t+1}^-)^{-1}(E(x_{t+1}|y_{1:n}) - E(\hat{x}_{t+1}^-|y_{1:n})) \\ &= \hat{x}_t^+ + P_t^+ F_t^T (P_{t+1}^-)^{-1}(\hat{x}_{t+1}^n - \hat{x}_{t+1}^-) = \hat{x}_t^n \end{aligned} \tag{76}$$

The smoother covariance is derived using the law of total covariance, which is given as follows:

$$\begin{aligned} &Cov(x_t|y_{1:n}) = Cov(E[x_t|x_{t+1}, y_{1:n}]|y_{1:n}) + E[cov(x_t|x_{t+1}, y_{1:n})|y_{1:n}] \\ &= Cov(E[x_t|x_{t+1}, y_{1:t}]|y_{1:n}) + E[cov(x_t|x_{t+1}, y_{1:t})|y_{1:n}] \\ &= Cov(\hat{x}_t^+ + P_t^+ F_t^T (P_{t+1}^-)^{-1}(x_{t+1} - \hat{x}_{t+1}^-)|y_{1:n}) + E[P_t^+ - P_t^+ F_t^T (P_{t+1}^-)^{-1} F_t P_t^+|y_{1:n}] \end{aligned}$$

Let us denote $J_t = P_t^+ F_t^T (P_{t+1}^-)^{-1}$, we have:

$$\begin{aligned} &Cov(x_t|y_{1:n}) = Cov(\hat{x}_t^+ + J_t(x_{t+1} - \hat{x}_{t+1}^-)|y_{1:n}) + E[P_t^+ - J_t P_{t+1}^- J_t^T|y_{1:n}] \\ &= J_t Cov(x_{t+1}|y_{1:n}) J_t^T + P_t^+ - J_t P_{t+1}^- J_t^T \\ &= J_t Var(x_{t+1}|y_{1:n}) J_t^T + P_t^+ - J_t P_{t+1}^- J_t^T \\ &= J_t P_{t+1}^n J_t^T + P_t^+ - J_t P_{t+1}^- J_t^T \\ &= P_t^+ + J_t(P_{t+1}^n - P_{t+1}^-) J_t^T = P_t^n \end{aligned} \tag{77}$$

□

In summary, the smoother value for the state estimate and the covariance in Kalman filter is given as follows:

$$\begin{aligned} \hat{x}_t^n &= \hat{x}_t^+ + J_t(\hat{x}_{t+1}^n - \hat{x}_{t+1}^-) \\ P_t^n &= P_t^+ + J_t(P_{t+1}^n - P_{t+1}^-) J_t^T \\ J_t &= P_t^+ F_t^T (P_{t+1}^-)^{-1} \end{aligned}$$

which now completes the E-step of the EM algorithm.

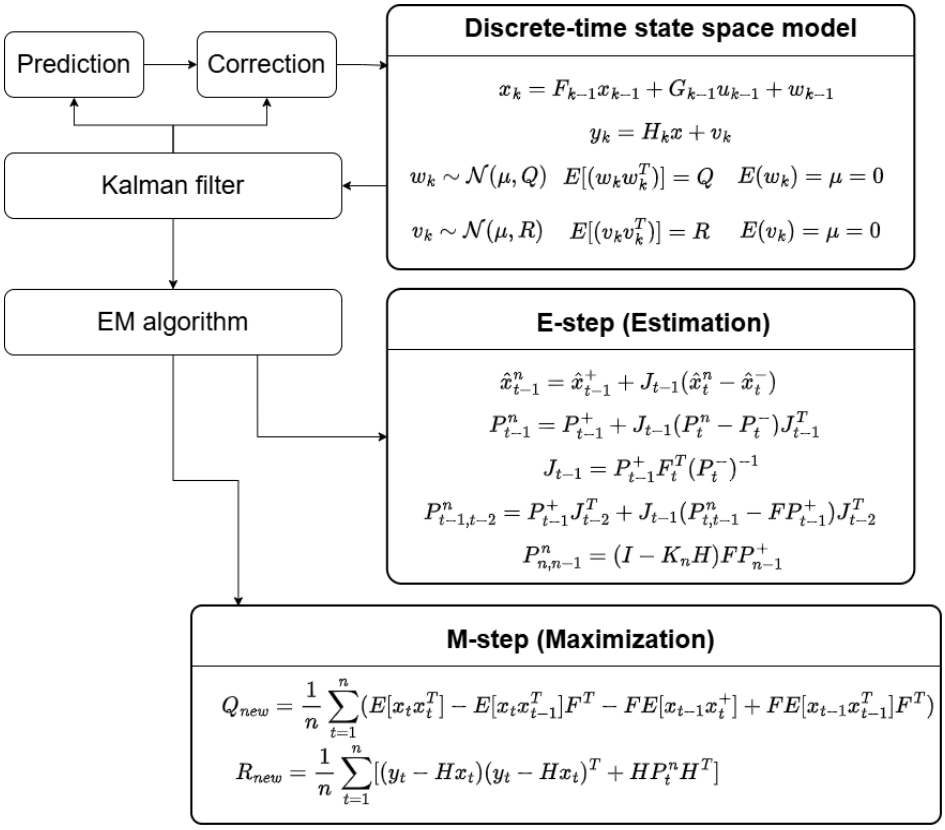


Figure 7: EM algorithm.

6 Kalman filter application

So far, we have derived the Kalman filter using the recursive least squares method. In addition, we have developed a general understanding of how the EM algorithm is implemented. Let us now apply these methods to tackle the tracking problem using non-simulated data. Our task is to filter out the noisy GPS measurement to approximate the true distance and speed of a city bus, given noisy GPS distance and speed measurement. Furthermore, the acceleration of the city bus is measured using another sensor at the same time. For more details on how the algorithms are implemented, curious reader can refer to Appendix 3

6.1 Data collection

The experiment data is collected using *Phyphox*, also known as the physical phone experiment. This tool allows us to collect real time noisy measurement data using the GPS sensors from our phones. For our tracking experiment, we used two different sensors. One GPS chip is the Intel PMB9955. This chip (or sensor) is used to measure the acceleration of the city bus. The other sensor is embedded inside the processor of Motorola g24 phone, called as MediaTekHelio G85 which is used to collect the GPS data of the city bus. These sensors were activated and stopped at the same time. The raw data that we obtained consist of a few attributes that may not be useful for the simplicity of this experiment. A few values were not collected, and as a result these values were set to *NaN* (short for not a number). We then replaced these values with the current collected value to achieve a clean data set. In addition, 100 different acceleration values are collected during 1 second. We average these collected values to attain the acceleration value for that second, which will become useful as our distance measurement is collected once for every 1 second. The dataset is published publicly in Zenodo (Nguyen, 2026). Curious reader can use this dataset to reimplement the practical experiment in this section.

6.2 Implementation and discussion

To begin with our tracking application, recall the simple state-space model mentioned in section 4.1:

$$\mathbf{x}_k = \begin{bmatrix} p_k \\ s_k \end{bmatrix} = F\mathbf{x}_{k-1} + Ga_{k-1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{k-1} \\ s_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} a_{k-1} + w_{k-1},$$

$$w_k \sim \mathcal{N}(0, Q_k),\ Q_k = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{78}$$

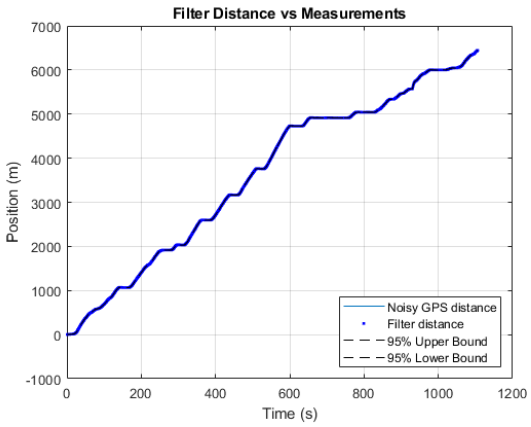
$$\mathbf{y}_k = H\mathbf{x}_k + v_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}_k + v_k,\ v_k \sim \mathcal{N}(0, R_k),\ R_k = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \tag{79}$$

where our state transition matrix $F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$, the control-input matrix G is $\begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$, and the process noise is assumed to be Gaussian, zero-mean with the covariance noise matrix Q_k. The measurement matrix H in our task is $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Our measurement noise is assumed to be Gaussian, zero-mean, with the measurement covariance noise matrix R_k.

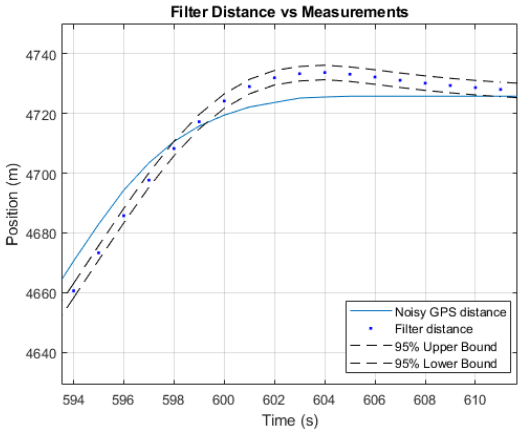
Up to this point, the data and the state-space model are ready. Let us choose an arbitrary value for the initial state vector and the estimation-error covariance matrix to start the Kalman filter.

$$\hat{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, P_0 = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$$

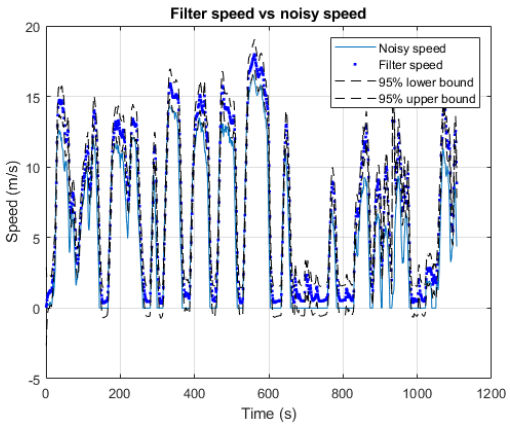
The filtered value for the distance and speed of the city bus is shown in the figure 8 below.



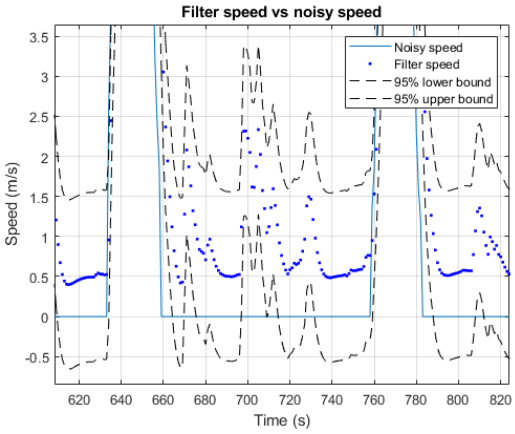
(a) Noisy distance measurement and filter distance.



(b) Filter distance (Zoom in).



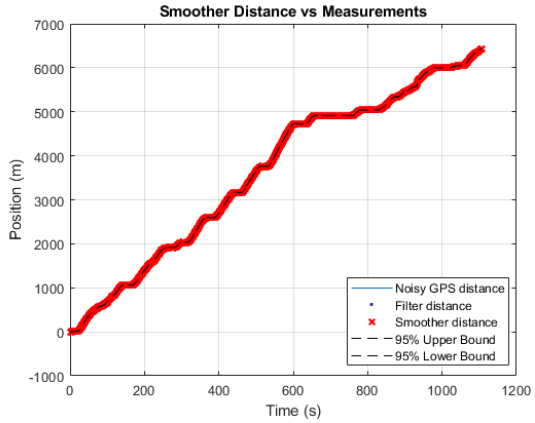
(c) Noisy speed measurement and filter speed.



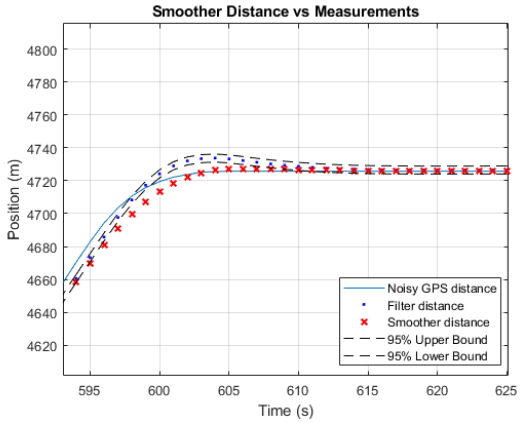
(d) Filter speed (Zoom in).

Figure 8: Distance and speed estimates of the truck compared with their respective filter values.

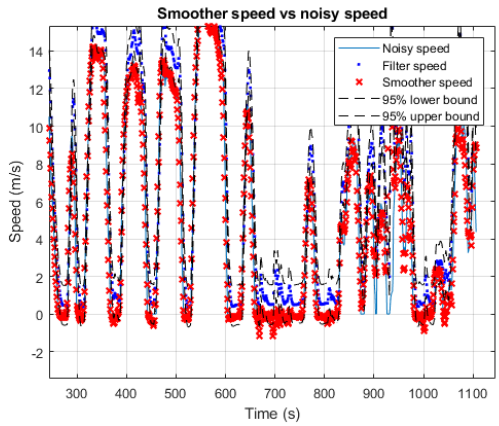
We know exactly that the bus does not move in some time intervals, as it has to stop to let passengers get on and off at each bus stop. For example, in figure 8d, the speed of the bus is around 0 m/s in the range of 660 to around 750 seconds. The filter speed value fluctuates from around 0.5 m/s to 2.5 m/s, which does not accurately reflect the state of the bus. The question is whether the smoother speed value can precisely approximate the state of the bus during this time interval? Let us examine the result given in figure 9 below.



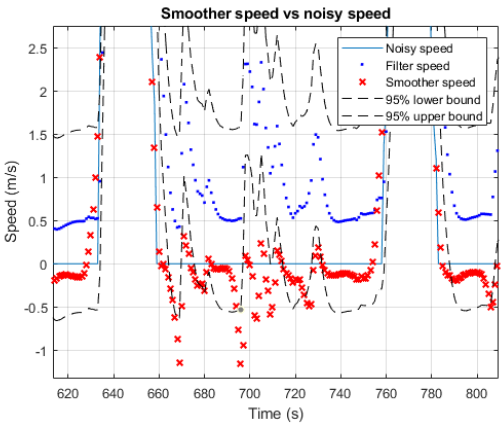
(a) Noisy distance measurement and smoother distance.



(b) Smoother distance (Zoom in).



(c) Noisy speed measurement and smoother speed.



(d) Smoother speed (Zoom in).

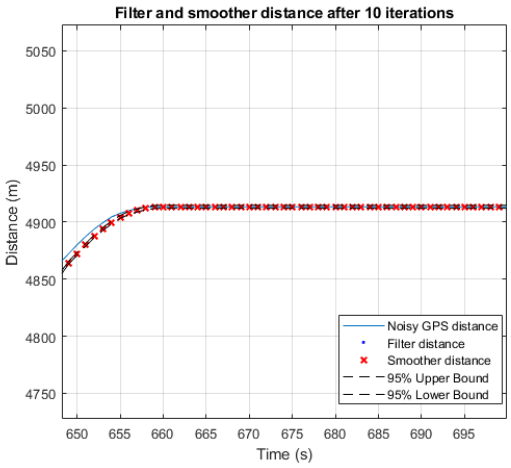
Figure 9: Noisy distance and speed measurements of the truck compared with their respective filter values and smoother values.

Clearly, we can observe a better speed approximation from figure 9d. For example, starting from around 660 to around 760 seconds, the smoother speed value now fluctuates around 0, whereas the filter speed fluctuates around 0.5. When applying the EM algorithm, we obtain not only the smoother state estimate but also the value for the covariance noise matrix Q and R. Using the new value for the covariance noise matrix and running the Kalman filter

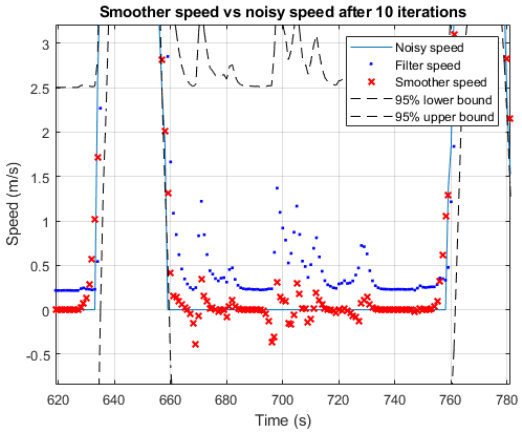
again, the filter value will eventually get better, and converge close to the true value. Let us illustrate this idea by running 25 iterations of Kalman filter with EM algorithm. The result is shown in figure 10 below.

In figure 10e, the approximated mean of the filter speed during the time interval from around 660 to 760 seconds after 25 iterations is now fluctuating around the true speed value (0 $\mathrm{m/s}$), whereas the filter speed value after 10 iterations of Kalman filter and EM algorithm vary from 0.2 $\mathrm{m/s}$ to around 1.5 $\mathrm{m/s}$. If the bus is not moving during this time interval, we also expect that the distance will be constant as well. And, as predicted earlier, the filter distance shown in figure 10c now converges to a constant.

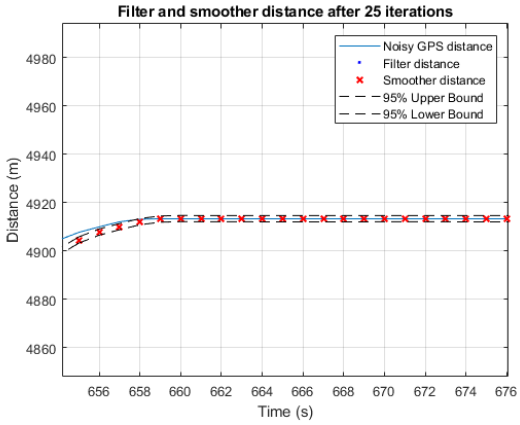
What makes the Kalman filter so powerful in our tracking problem is its recursive form. The Kalman filter uses the current obtained data to approximate the value of the next state. Therefore, the Kalman filter is fast to compute and easy to implement, whereas the batch processing method will process all the data at once to predict the value for the next state, which requires a lot of computational power. We can also see a significant improvement in the estimation of the filter value using the EM algorithm and the Kalman filter. By not just doing one Kalman iteration, we can perform the filtering process multiple times with the parameter estimation value obtained from the EM algorithm. This will avoid the process of manually setting the value for the covariance noise matrix, which will play an essential role in affecting the filter correctness.



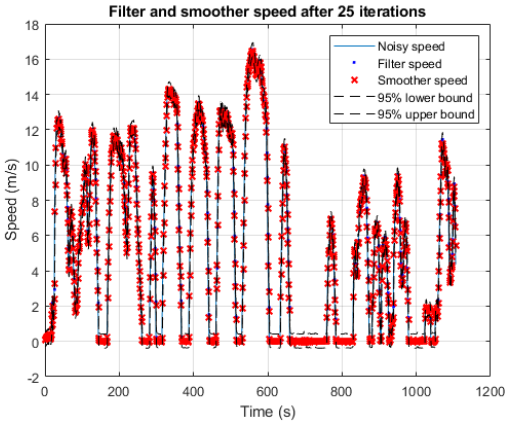
(a) Smoother distance after 10 iterations (Zoom in).



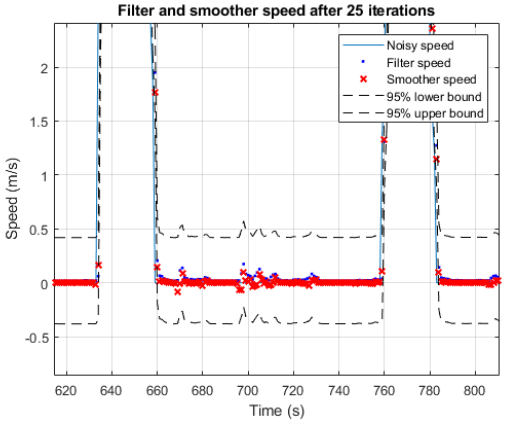
(b) Smoother speed after 10 iterations (Zoom in).



(c) Smoother distance after 25 iterations (Zoom in).



(d) Smoother speed after 25 iterations



(e) Smoother speed after 25 iterations (Zoom in)

Figure 10: Distance and speed estimates of the truck compared with their respective new filter values and smoother values.

7 Conclusion

In conclusion, our aim in this research is to derive the Kalman filter using the recursive least squares method and the extended Kalman filter using the previously obtained result, and to investigate how the Kalman filter and the EM algorithm could be used in the state estimation of a dynamical system. From one experiment using simulated data and the other using actual data, we agree that the Kalman filter was able to estimate the hidden state values of a linear state-space model using the noisy measurement data, although the estimation is not very accurate. With the help of the EM algorithm, the performance of the Kalman filter was getting better, and the filter values converge to the actual true value after a number of iterations. The EM algorithm also helps to estimate the noise covariance matrix, avoids the assumption of individuals, which will make the filter results close to the actual value. As mentioned earlier, fine tuning the covariance noise matrix is a difficult task and also an ongoing research topic. With a bad covariance noise matrix, the filter results would be imprecise and diverge from the actual state values. Also, we can claim that the Kalman filter is the best linear solution without using the Gaussian assumption for the process and measurement noise. With all of these results, we can now understand the importance of the Kalman filter and the reason why it is becoming the backbone algorithm in various practical applications. The experiment that we are working on is a linear state-space model, with Gaussian noise assumptions. For a more general nonlinear problem with different distributions for the noise assumption, the same result would not be obtained. We could also study in more detail for other form of Kalman filter which can be applied to the non-linear world, and applying others algorithm, or even using machine learning algorithm to estimate the noise covariance matrix in these scenarios in our future work as well.

References

Battin, R. H. 1982. Space guidance evolution—A personal narrative. *Journal of Guidance, Control, and Dynamics* 5(2), pp. 97–110. DOI: 10.2514/3.19761.

Billingsley, P. 2012. *Probability and Measure*. Wiley Series in Probability and Statistics. Wiley. ISBN: 9781118341919. URL: https://books.google.fi/books?id=a3gavZbxyJcC.

Bishop, C. 2016. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York. ISBN: 9781493938438. URL: https://books.google.fi/books?id=kOXDtAEACAAJ.

Chui, C. & Chen, G. 2017. *Kalman Filtering: with Real-Time Applications*. Physics and Astronomy. Springer International Publishing. ISBN: 9783319476124. URL: https://books.google.com.vn/books?id=8o50DgAAQBAJ.

Dumortier, F. 1985. Theory and practice of recursive identification: Lennart Ljung and Torsten Söderström. *Automatica* 21(4), pp. 499–501. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(85)90088-3. URL: https://www.sciencedirect.com/science/article/pii/0005109885900883.

Grewal, M., Weill, L. & Andrews, A. 2007. *Global Positioning Systems, Inertial Navigation, and Integration*. Wiley. ISBN: 9780470099711. URL: https://books.google.fi/books?id=799oi-elP0sC.

Grewal, M. & Andrews, A. Jan. 2001. Kalman filtering: theory and practice using MATLAB. *New York: John Wiley and Sons* 14. DOI: 10.1002/9780470377819.

Grewal, M. & Andrews, A. July 2010. Applications of Kalman Filtering in Aerospace 1960 to the Present [Historical Perspectives. *Control Systems, IEEE* 30, pp. 69–78. DOI: 10.1109/MCS.2010.936465.

Haykin, S. S. 2014. *Adaptive Filter Theory*. 5th. Upper Saddle River, NJ: Pearson. ISBN: 978-0133356724.

Humpherys, J., Redd, P. & West, J. Jan. 2012. A Fresh Look at the Kalman Filter. en. *SIAM Review* 54(4), pp. 801–823. DOI: 10.1137/100799666. URL: http://dx.doi.org/10.1137/100799666.

Islam, S. & Bernstein, D. June 2019. Recursive Least Squares for Real-Time Implementation [Lecture Notes]. *IEEE Control Systems* 39, pp. 82–85. DOI: 10.1109/MCS.2019.2900788.

Kalman, R. E. 1960. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering* 82(1), pp. 35–45.

Kiencke, U. & Nielsen, L. 2000. *Automotive Control Systems*. Society of Automotive Engineers. Springer. ISBN: 9780768005059. URL: https://books.google.fi/books?id=QcuXQgAACAAJ.

Lai, B. & Bernstein, D. S. 2024. *Generalized Forgetting Recursive Least Squares: Stability and Robustness Guarantees*. arXiv: 2308.04259 [eess.SY]. URL: https://arxiv.org/abs/2308.04259.

McGee, L. A. & Schmidt, S. F. 1985. *Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry*. NASA Technical Memorandum NASA-TM-86847. Moffett Field, California: NASA Ames Research Center.

Murphy, K. 2012. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series. MIT Press. ISBN: 9780262304320. URL: https://books.google.fi/books?id=RC43AgAAQBAJ.

Nguyen, L. D. A. Oct. 2026. *GPS and acceleration dataset of a bus driving in the city center*. Zenodo. DOI: 10.5281/zenodo.23284348. URL: https://doi.org/10.5281/zenodo.23284348.

Pearson, J. B. & Stear, E. B. 1974. Kalman Filter Applications in Airborne Radar Tracking. *IEEE Transactions on Aerospace and Electronic Systems* AES-10(3), pp. 319–329. DOI: 10.1109/TAES.1974.307824.

Sastry, S. & Bodson, M. 1989. *Adaptive Control: Stability, Convergence, and Robustness*. Englewood Cliffs, NJ: Prentice-Hall. ISBN: 0-13-004326-5.

Serio, G., Manara, A. & Sicoli, P. Jan. 2002. Giuseppe Piazzi and the Discovery of Ceres. *Asteroids III*.

Shumway, R. H. & Stoffer, D. S. 1982. AN APPROACH TO TIME SERIES SMOOTHING AND FORECASTING USING THE EM ALGORITHM. *Journal of Time Series Analysis* 3(4), pp. 253–264. DOI: https://doi.org/10.1111/j.1467-9892.1982.tb00349.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9892.1982.tb00349.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9892.1982.tb00349.x.

Shumway, R. H. & Stoffer, D. S. 2017. *Time Series Analysis and Its Applications: With R Examples*. 4th. Springer. DOI: 10.1007/978-3-319-52452-8.

Simon, D. 2006. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley. ISBN: 9780470045336. URL: https://books.google.com.vn/books?id=UiMVoP_7TZkC.

Stigler, S. M. 1981. Gauss and the invention of least squares. *the Annals of Statistics*, pp. 465–474.

Strang, G. 2006. *Linear Algebra and Its Applications*. Thomson, Brooks/Cole. ISBN: 9780534422004. URL: https://books.google.com.vn/books?id=q9CaAAAACAAJ.

Zarchan, P., Musoff, H., Aeronautics, A. I. of & Astronautics. 2009. *Fundamentals of Kalman Filtering: A Practical Approach*. Progress in astronautics and aeronautics. American Institute of Aeronautics and Astronautics. ISBN: 9781600867187. URL: https://books.google.com.vn/books?id=pUvTAQAACAAJ.

Appendix 1 MATLAB code for the recursive least squares method

```
%% The recursive least squares method
% Duc Nguyen, 2026

clc; close all; clearvars;

initial_position = 20;
initial_vel = 2;
initial_acceleration = 1;

n = 1000;
R = 1; % Measurement noise variance

position = zeros(1,n);
noisy_position = zeros(1,n);

time = linspace(0,1000,n);

% Generate true and noisy position

for i = 1:n
    % True position
    position(i) = initial_position ...
        + initial_vel*time(i) ...
        + 0.5*initial_acceleration*time(i)^2;
    % Noisy measurement
    noisy_position(i) = position(i) + sqrt(R)*randn;
end

x_0_hat = [0; 0; 0];

P_0 = [100 0 0;
       0 100 0;
       0 0 100];

filter_pos = zeros(1,n);
filter_vel = zeros(1,n);
filter_acceleration = zeros(1,n);
est_val = x_0_hat;
est_cov = P_0;

for i = 1:n
    % Measurement matrix
    C = [1 time(i) 0.5*time(i)^2];
```

```
    % RLS gain
    K = est_cov*C' / (C*est_cov*C' + R);

    % State estimate
    est_val = est_val + K*(noisy_position(i) - C*est_val);

    % Covariance update
    est_cov = (eye(3) - K*C)*est_cov;

    % Store estimated position
    filter_vel(i) = est_val(2);
    filter_pos(i) = est_val(1);
    filter_acceleration(i) = est_val(3);
end
```

Appendix 2 MATLAB code for the Kalman filter

```
%% Kalman example: simulated data only
% Duc Nguyen, 2026

clc; close all; clearvars;

rng(1);   % Reproducible random data
initial_position = 20;
initial_vel = 2;

dt = 0.05;
n = 100;
time = (0:n-1)*dt;

F = [1, dt;
     0, 1];

G = [0.5*dt^2;
     dt];

H = [1, 0];

sigma_a = 0.005;

% Process-noise covariance
Q_k = [0.1 0;0 0.1];

% GPS measurement-noise variance
R_k = 4;

acceleration_arr = zeros(1,n-1);
acceleration_arr(1:20) = 1.4;
acceleration_arr(21:60) = 0;
acceleration_arr(61:80) = 2;
acceleration_arr(81:end) = 2;

true_state_arr = zeros(2,n);
noisy_GPS = zeros(1,n);

true_state_arr(:,1) = [initial_position;
                       initial_vel];

noisy_GPS(1) = H*true_state_arr(:,1) ...
             + sqrt(R_k)*randn;
```

```
for i = 2:n
    % Process noise
    process_noise = sqrt(0)*randn(2,1);

    % True state
    true_state_arr(:,i) = F*true_state_arr(:,i-1) ...
                        + G*acceleration_arr(i-1) ...
                        + process_noise;

    % Noisy GPS position measurement
    noisy_GPS(i) = H*true_state_arr(:,i) ...
                 + sqrt(R_k)*randn;
end

% Extract true position and speed
true_position = true_state_arr(1,:);
true_speed = true_state_arr(2,:);

%% kalman

N = n - 1; % Number of filtered steps
filter_state = zeros(2, N);
priori_state = zeros(2, N);
P_minus_forward = zeros(2, 2, N);
P_plus_forward = zeros(2, 2, N);
P11 = zeros(1, N);
P22 = zeros(1, N);

x_0_hat = [0; 0];
P_0 = [100 0; 0 100];
est_state = x_0_hat;
est_P = P_0;

for i = 2:n
    idx = i - 1; % Align storage index to (1 to n-1)

    % Prediction
    x_k_minus = F * est_state + G * acceleration_arr(idx);
    P_minus = F * est_P * F' + Q_k;

    P_minus_forward(:, :, idx) = P_minus;
    priori_state(:, idx) = x_k_minus;

    % Correction
    K = P_minus * H' / (R_k + H * P_minus * H');
    est_state = x_k_minus + K * (noisy_GPS(i) - H * x_k_minus);
```

```
    est_P = (eye(2) - K * H) * P_minus * (eye(2) - K * H)' + K * R_k * K
        ';

    P_plus_forward(:, :, idx) = est_P;
    filter_state(:, idx) = est_state;
    P11(idx) = est_P(1, 1);
    P22(idx) = est_P(2, 2);
end

upper_pos = filter_state(1, :) + 2 * sqrt(P11);
lower_pos = filter_state(1, :) - 2 * sqrt(P11);
```

Appendix 3 Kalman filter and EM algorithm practical example

```
%% Practical experiment with Kalman filter and EM algorithm
% Duc Nguyen, 2026

clc; close all; clearvars;

% Load Excel files and set up all the needed variables.

%% Kalman filter function
function [N,filter_state, priori_state, P_minus_forward, P_plus_forward,
    ...
    P11, P22, upper_pos, lower_pos] = kalmanFilter(n, x_0_hat ...
    , P_0, noisy_gps, F, G, H, Q_k, R_k, acc_arr)
    N = n - 1; % Number of filtered steps
    filter_state = zeros(2, N);
    priori_state = zeros(2, N);
    P_minus_forward = zeros(2, 2, N);
    P_plus_forward = zeros(2, 2, N);
    P11 = zeros(1, N);
    P22 = zeros(1, N);
    est_state = x_0_hat;
    est_P = P_0;

    for i = 2:n
        idx = i - 1; % Align storage index to (1 to n-1)

        % Prediction
        x_k_minus = F * est_state + G * acc_arr(idx);
        P_minus = F * est_P * F' + Q_k;

        P_minus_forward(:, :, idx) = P_minus;
        priori_state(:, idx) = x_k_minus;

        % Correction
        K = P_minus * H' / (R_k + H * P_minus * H');
        est_state = x_k_minus + K * (noisy_gps(:,idx) - H * x_k_minus);
        est_P = (eye(2) - K * H) * P_minus * (eye(2) - K * H)' + K ...
            * R_k * K';

        P_plus_forward(:, :, idx) = est_P;
        filter_state(:, idx) = est_state;
        P11(idx) = est_P(1, 1);
        P22(idx) = est_P(2, 2);
```

```
    end

    % Confidence bounds
    upper_pos = filter_state(1, :) + 2 * sqrt(P11);
    lower_pos = filter_state(1, :) - 2 * sqrt(P11);
end

%% EM algorithm function
function [smooth_val, smooth_cov, Q_new, R_new] = EMalg(N, filter_state, P
    _plus_forward,P_minus_forward,priori_state,F,H,R_k, noisy_gps)
    % RTS Smoother (EM E-Step)
    smooth_val  = zeros(2, N);
    smooth_cov  = zeros(2, 2, N);
    J_gains     = zeros(2, 2, N); % Store J_t cleanly at index t

    % Boundary condition at t = N
    smooth_val(:, N)    = filter_state(:, N);
    smooth_cov(:, :, N) = P_plus_forward(:, :, N);

    % 1. Standard RTS Backward Pass
    for t = N-1:-1:1
        % Smoother gain J_t = P_{t|t} * F' / P_{t+1|t}
        J = P_plus_forward(:, :, t) * F' / P_minus_forward(:, :, t + 1);
        J_gains(:, :, t) = J; % J_t stored at index t

        % Mean and covariance updates
        smooth_val(:, t)   = filter_state(:, t) + J * (smooth_val(:, t +
            1) - priori_state(:, t + 1));
        smooth_cov(:, :, t) = P_plus_forward(:, :, t) + J * (smooth_cov
            (:, :, t + 1) - P_minus_forward(:, :, t + 1)) * J';
    end

    % Lag-1 Cross-Covariance Recursion (Shumway & Stoffer A12 & A11)
    P_lag = zeros(2, 2, N);
    K_N = P_minus_forward(:, :, N) * H' / (R_k + H * P_minus_forward(:
        ...
        , :, N) * H');
    % Compute the lag covariance at the time N and N-1
    P_lag(:, :, N) = (eye(2) - K_N * H) * F * P_plus_forward(:, :, N - 1)
        ;

    % (A11) Backward recursion
    % Computes P_cov_val(:,:,t-1) = P_{t-1, t-2|N} with t = N-1
    for t = N-1:-1:2
        J_t1 = J_gains(:, :, t);
        J_t2 = J_gains(:, :, t - 1);
```

```
        % Fixed: Subtracted P_plus_forward(:,:,t-1) instead of t
        P_lag(:, :, t) = P_plus_forward(:, :, t) * J_t2' + ...
                                 J_t1 * (P_lag(:, :, t+1) - F * ...
                                 P_plus_forward(:, :, t)) * J_t2';
    end

    % M-step
    s1 = zeros(2, 2);
    s2 = zeros(2, 2);
    s3 = zeros(2,2);
    for i = 2:N
        % Uncertainty contribution
        s1 = s1 + smooth_cov(:, :, i) ...
                - P_lag(:, :, i) * F' ...
                - F * P_lag(:, :, i)' ...
                + F * smooth_cov(:, :, i - 1) * F';

        % Mean residual contribution
        res = smooth_val(:, i) - F * smooth_val(:, i - 1);
        s2 = s2 + (res * res');
    end
    % Divide by number of transition intervals (N - 1 = 98)
    Q_new = (s1 + s2) / (N - 1);
    % Enforce numerical symmetry to guarantee real positive eigenvalues
    Q_new = (Q_new + Q_new') / 2;
    sR = zeros(2,2);

    for t = 1:N

        res_y = noisy_gps(:,t) ...
              - H*smooth_val(:,t);

        sR = sR ...
            + res_y*res_y' ...
            + H*smooth_cov(:,:,t)*H';
    end
    R_new = sR/N;
end
```