

Article

Trajectory Tracking of Reentry Vehicle Based on KalmanNet with Time-Varying Observation Matrix

Xinmiao Liu ¹, Wanchun Chen ^{2,*}, Wengui Lei ² and Zijiao Wang ²¹ National College for Excellent Engineers, Beihang University, Beijing 100191, China² School of Astronautics, Beihang University, Beijing 100191, China

* Correspondence: wanchun_chen@buaa.edu.cn

Abstract

This paper proposes a trajectory-tracking algorithm for reentry vehicles based on KalmanNet with a time-varying observation matrix. First, a nonlinear state evolution model of the reentry vehicle and a radar measurement model are developed in the radar measurement coordinate system. Then, inspired by the computation process of the Kalman gain (KG) in the extended Kalman filter (EKF), the recurrent neural network (RNN) architecture of KalmanNet is improved. The gated recurrent unit (GRU) originally used to track process noise statistics is removed. Instead, the input features are redesigned to directly estimate the prior state covariance. Furthermore, another GRU is introduced to estimate the time-varying observation matrix, considering the nonlinear characteristics of radar measurements. The calculated observation matrix is fed into both the GRU responsible for estimating the covariance of the difference between the predicted observation and the observed value and the fully connected layer that computes the KG. Finally, the proposed method is compared with six representative algorithms, including EKF, particle filter (PF), unscented Kalman filter (UKF), convolutional neural network (CNN), Long Short-Term Memory (LSTM), and the original KalmanNet. Simulation results demonstrate that the proposed method achieves the highest estimation accuracy, while its computational time remains nearly the same as that of the original KalmanNet. Monte Carlo simulations under three model-mismatch conditions are conducted to validate the robustness of the proposed method.

Keywords: KalmanNet; trajectory tracking; reentry vehicle; EKF; time-varying observation matrix

1. Introduction

Trajectory tracking during the reentry phase of flight vehicles is challenging due to the high flight velocity and uncertainties in the dynamic model, which often lead to large tracking errors. Since its introduction, the Kalman filter and its variants have become the mainstream approaches for state estimation [1–3]. To address nonlinear models, the EKF was developed by linearizing nonlinear models through the computation of the Jacobian matrix and subsequently applied to trajectory tracking of reentry vehicles [4].

Since the EKF was developed by linearizing nonlinear models using Jacobian matrices, its estimation accuracy deteriorates when the system exhibits strong nonlinearities. Gordon et al. [5] proposed the PF based on Monte Carlo simulations, which is capable of handling nonlinear systems and non-Gaussian noise [6]. To further improve nonlinear

Academic Editor: Dengqing Cao

Received: 1 June 2026

Revised: 22 June 2026

Accepted: 26 June 2026

Published: 6 July 2026

Copyright: © 2026 by the authors.

Submitted for possible open access

publication under the terms and

conditions of the [Creative Commons](#)[Attribution \(CC BY\)](#) license.

estimation, Julier and Uhlmann [7] introduced the UKF, which addresses nonlinear problems by using unscented transformations, thereby improving the linearization accuracy for nonlinear models. Later, Merwe and Wan [8] improved the selection of sigma points within the unscented transformation.

Arasaratnam et al. [9,10] further proposed the cubature Kalman filter (CKF) algorithm. The core idea of this algorithm is the use of a third-order spherical-radial cubature rule to approximate the posterior mean and covariance of nonlinear functions. The CKF employs a third-order spherical-radial cubature rule to approximate Gaussian-weighted integrals. Compared with the UKF, CKF provides a more systematic numerical integration framework and improved stability for nonlinear filtering problems.

In recent years, deep neural networks have been widely applied in various scenarios. These models are capable of directly learning the functional relationship between input and output features through end-to-end data-driven training [11,12], and have achieved significant progress in image processing, navigation and control, as well as state prediction [13–15]. Several deep recurrent neural network architectures, such as LSTM [16] and GRU [17], have been explored for tracking applications [18–21].

The CNN was originally proposed by LeCun et al. [22,23] in the last century, and gained widespread attention after being further developed by Krizhevsky et al. [24]. In recent years, CNNs have been widely applied in fields such as image processing, face recognition, and audio retrieval. Target tracking algorithms based on regression using CNNs estimate the target position directly from the heat map or probability map generated by the network output, and have attracted increasing attention from researchers [25–27].

For the trajectory-tracking problem of reentry vehicles, Farina et al. [28] applied various nonlinear filtering algorithms to the reentry phase of the vehicles. Zarchan [29] derived a more accurate process-noise model when using the EKF for reentry tracking. Kumar et al. [30] proposed an adaptive extended Kalman filter to track reentry vehicles. Zhao et al. [31] introduced a nonlinear radar measurement model for reentry tracking, making the scenario closer to real operational conditions.

The study of stable tracking and control for nonlinear systems using mathematical optimization methods and intelligent algorithms has also become one of the active research topics in recent years. Jin conducted extensive research on feedback optimization for the stabilization of nonlinear systems [32]. Furthermore, by incorporating filtering-based tracking techniques, Jin et al. investigated the cooperative control of quadrotor swarms under tracking requirements [33]. Chen et al. [34] combined a Convolutional Neural Network with the multiplicative extended Kalman filter (MEKF) to track the position and attitude of spacecraft. Revach et al. [35] proposed KalmanNet, a Kalman-filter-based framework for linear systems in which a recurrent neural network replaces the computation of the KG, significantly accelerating the calculation. Revach et al. [36] further extended KalmanNet to nonlinear systems based on the EKF framework. Choi et al. [37] decomposed the recurrent neural network structure in KalmanNet and recombined it for gain computation, proposing Split-KalmanNet, which improves the robustness of the original model.

However, these studies mainly focus on trajectory tracking under constant-velocity (CV) or constant-acceleration (CA) models and do not explore the use of KalmanNet for highly nonlinear dynamic and radar measurement models such as those encountered in reentry vehicle tracking. Moreover, when a radar measurement model is used, the observation matrix becomes time-varying rather than constant, which places higher demands on the performance of KalmanNet.

Building on existing EKF theory and the KalmanNet methodology, this paper investigates trajectory tracking of reentry vehicles in three-dimensional space using Kal-

manNet, and proposes structural improvements tailored to this application. The main contributions of this work are summarized as follows:

(1) A state-space model of the reentry vehicle is formulated in the radar measurement coordinate system, including both the vehicle's state evolution model and the radar measurement model. The dynamic model contains the position and velocity components in radar measurement coordinates, as well as accelerations induced by aerodynamic lift, drag, and side forces. The measurement model corresponds to a three-dimensional radar observation model, with range, elevation, and azimuth as the measurement quantities.

(2) The EKF is described in detail to lay the foundation for the subsequent explanation of the basic principles of KalmanNet and the design of Ht-KalmanNet. Additionally, we briefly introduce the approaches for trajectory tracking using the PF, UKF, CNN, and LSTM models. Based on the existing literature, we provide an approach for using KalmanNet to perform trajectory tracking of reentry vehicles, along with the input features and the neural network architecture employed.

(3) A trajectory tracking algorithm tailored to the present application is proposed and referred to as Ht-KalmanNet. In KalmanNet, the estimation of the observation matrix is neglected during the fitting of the KG, which is problematic since the observation matrix varies significantly over time in reentry scenarios. To address this, we remove the network branch responsible for process noise estimation and introduce a new neural network module to update the observation matrix. This modification allows the estimated KG to more closely align with the mathematical structure of the EKF and more accurately capture the physical characteristics of reentry vehicles.

(4) Simulation conditions are established, and Monte Carlo simulations are conducted to evaluate the estimation accuracy and computational efficiency of seven filtering methods, including the EKF, PF, UKF, CNN, LSTM, KalmanNet, and Ht-KalmanNet. Furthermore, under three types of model-mismatch conditions—introduction of a sideslip angle to generate lateral maneuvering, bias in range measurement standard deviation (STD), and bias in angle measurement standard deviation—the effectiveness and robustness of Ht-KalmanNet are verified through Monte Carlo simulations.

This paper characterizes the tracking accuracy of each filtering method using the root mean square error (RMSE) of position tracking, and discusses the velocity tracking error and computational time for each method. Compared with the original KalmanNet, the proposed Ht-KalmanNet incorporates the time-varying observation matrix into the Kalman gain estimation process. As a result, the filtering model exhibits an improved fitting capability and achieves more accurate state estimation performance.

The remainder of this paper is organized as follows. Section 2 establishes the state-space model of the reentry vehicle. Section 3 introduces conventional trajectory tracking methods, which serve as benchmark methods for comparison with Ht-KalmanNet. Section 4 provides an in-depth analysis of the structure of the Extended Kalman Filter and proposes the Ht-KalmanNet method based on improvements to KalmanNet. Section 5 presents the simulation settings and reports the tracking errors and computational time of different methods. In addition, parameter bias is introduced to generate model mismatch under various conditions in order to verify the robustness of Ht-KalmanNet.

2. System Model and Problem Formulation

Since this study focuses on reentry vehicles, the problem scenario is formulated in this section, consisting primarily of the state evolution model of the reentry vehicle and the radar measurement model.

2.1. State Evolution Model of the Reentry Vehicle

The reentry vehicle model considered in this study involves a relatively short flight duration. Therefore, the flat-Earth assumption is employed, and the effect of Earth's rotation is neglected.

A radar measurement coordinate system $O-x_i y_i z_i$ is established with its origin located at the radar station. The Ox_i axis is aligned with the local north direction, the Oy_i axis is aligned with the local vertical direction pointing upward and perpendicular to the Ox_i axis, and the Oz_i axis is determined according to the right-hand rule.

A ballistic coordinate system $O-x_v y_v z_v$ is also established at the radar station. The Ox_v axis is aligned with the instantaneous velocity vector of the target; the Oy_v axis lies in the vertical plane, is perpendicular to the Ox_v axis and points upward; and the Oz_v axis is determined according to the right-hand rule.

Since the shape of a vehicle can typically be estimated to some extent [38], an aerodynamic model of a specific vehicle is constructed for tracking purposes while accounting for the effects of drag, lift, and lateral forces [39]. The state vector is defined as Equation (1).

$$\mathbf{x} = [x \ y \ z \ v_x \ v_y \ v_z \ a_d \ a_l \ a_r]^T \quad (1)$$

where x, y, z represent the position information of the vehicle in the radar measurement coordinate system, v_x, v_y, v_z denote the corresponding velocity components, and a_d, a_l, a_r denote the accelerations induced by drag, lift, and side force, in the ballistic coordinate system, respectively. Figure 1 illustrates the coordinate systems and characteristic quantities considered in this study.

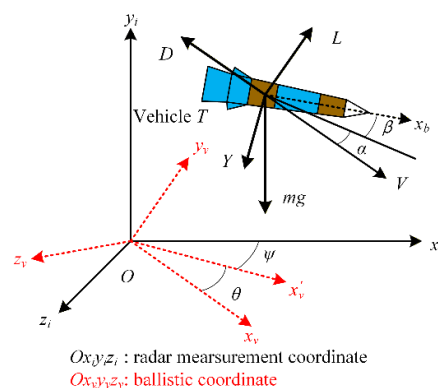


Figure 1. Illustration of the coordinate systems and characteristic quantities.

The flight-path angle θ and heading angle ψ are defined as Equations (2) and (3), respectively.

$$\theta = \arctan \left(\frac{-v_y}{\sqrt{v_x^2 + v_z^2}} \right) \quad (2)$$

$$\psi = \arctan \left(\frac{v_z}{v_x} \right) \quad (3)$$

Let C_v^i denote the transformation matrix from the ballistic coordinate system to the radar measurement coordinate system. Based on the definitions of the radar measurement coordinate system and the velocity coordinate system, Equation (4) can be derived.

$$C_v^i = \begin{bmatrix} \cos \psi \cos \theta & \cos \psi \sin \theta & -\sin \psi \\ -\sin \theta & \cos \theta & 0 \\ \sin \psi \cos \theta & \sin \psi \sin \theta & \cos \psi \end{bmatrix} \quad (4)$$

The accelerations in the three directions—namely, the drag acceleration a_D , lift acceleration a_L , and lateral-force acceleration a_Y —are given as follows:

$$a_D = \frac{-D}{m_0} + w_D \quad (5)$$

$$a_L = \frac{L}{m_0} + w_L \quad (6)$$

$$a_Y = \frac{Y}{m_0} + w_Y \quad (7)$$

where m_0 denotes the vehicle mass, set to 641.7 kg, and w_D, w_L and w_Y represent zero-mean Gaussian white noise with variances $\sigma_D^2, \sigma_L^2, \sigma_Y^2$, respectively. In this study, it is assumed that the values of $\sigma_D^2, \sigma_L^2, \sigma_Y^2$ are all the same and equal to σ_a^2 . The quantities D , L and Y , corresponding to the drag, lift, and lateral forces acting on the vehicle, are expressed as follows.

$$D = \frac{\rho(v_x^2 + v_y^2 + v_z^2)SC_D}{2} \quad (8)$$

$$L = \frac{\rho(v_x^2 + v_y^2 + v_z^2)SC_L}{2} \quad (9)$$

$$Y = \frac{\rho(v_x^2 + v_y^2 + v_z^2)SC_Y}{2} \quad (10)$$

where ρ denotes the atmospheric density and S represents the equivalent reference area of the vehicle, set to 0.7854 m² in this paper. C_D , C_L , and C_Y represent the drag coefficient, lift coefficient, and lateral-force coefficient, respectively. In this study, these coefficients are obtained through interpolation based on the aerodynamic model of the vehicle. The variations in the drag, lift, and lateral-force coefficients with respect to the Mach number, angle of attack α_0 , and sideslip angle β_0 are shown in Figure 2.

Accordingly, the dynamic model of the vehicle in the radar observation coordinate system is given in Equation (11).

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = C_v^i \begin{bmatrix} a_D \\ a_L \\ a_Y \end{bmatrix} + \begin{bmatrix} 0 \\ -g \\ 0 \end{bmatrix} \\ a_D = \frac{\rho(v_x^2 + v_y^2 + v_z^2)SC_D}{-2m_0} + w_D \\ a_L = \frac{\rho(v_x^2 + v_y^2 + v_z^2)SC_L}{2m_0} + w_L \\ a_Y = \frac{\rho(v_x^2 + v_y^2 + v_z^2)SC_Y}{2m_0} + w_Y \end{cases} \quad (11)$$

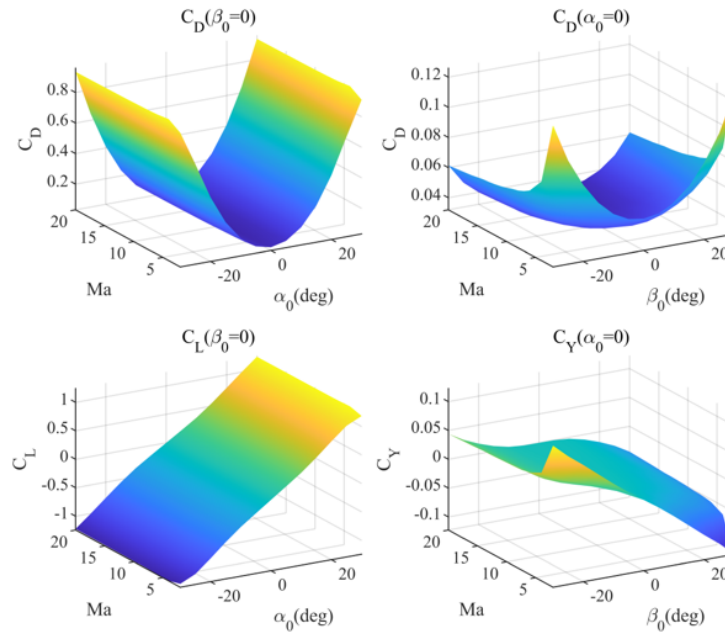


Figure 2. Aerodynamic modeling results.

To establish the state evolution model as expressed in Equation (12), both the dynamic model and the discrete process-noise model must be derived.

$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}) + \mathbf{e}_t, \mathbf{e}_t \sim N(0, \mathbf{Q}_k), \mathbf{x}_t \in R^9 \quad (12)$$

The discrete process-noise matrix $\mathbf{Q}_k$ is derived based on the EKF process-noise formulation presented by Zarchan [29]. The continuous process-noise matrix is given in Equation (13).

$$\mathbf{Q} = E(\mathbf{w}\mathbf{w}^T) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma_a^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_a^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_a^2 \end{bmatrix} \quad (13)$$

Subsequently, the system dynamic matrix is obtained with the Jacobian matrix in Equation (14).

$$\mathbf{F} = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \bigg|_{\mathbf{x}=\hat{\mathbf{x}}} \quad (14)$$

Based on the system dynamic matrix, the state-transition matrix of the system is computed as expressed in Equation (15), where t_s denotes the time interval.

$$\Phi_k = 1 + Ft_s = \begin{bmatrix} 1 & 0 & 0 & t_s & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & t_s & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & t_s & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & C_v^i[1,1]t_s & C_v^i[1,2]t_s & C_v^i[1,3]t_s \\ 0 & 0 & 0 & 0 & 1 & 0 & C_v^i[2,1]t_s & C_v^i[2,2]t_s & C_v^i[2,3]t_s \\ 0 & 0 & 0 & 0 & 0 & 1 & C_v^i[3,1]t_s & C_v^i[3,2]t_s & C_v^i[3,3]t_s \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (15)$$

To compute the discrete process-noise matrix, Equations (13) and (15) are substituted into Equation (16) by replacing t_s with τ .

$$Q_k = \int_0^{t_s} \Phi(\tau) Q \Phi^T(\tau) d\tau \quad (16)$$

Finally, a symmetric ninth-order discrete process-noise matrix is obtained. All non-zero elements on and above the main diagonal are given in Equations (17)–(34).

$$Q_k[4,4] = \frac{\sigma_a^2 t_s^3}{3} \left((C_v^i[1,1])^2 + (C_v^i[1,2])^2 + (C_v^i[1,3])^2 \right) \quad (17)$$

$$Q_k[4,5] = \frac{\sigma_a^2 t_s^3}{3} (C_v^i[1,1]C_v^i[2,1] + C_v^i[1,2]C_v^i[2,2] + C_v^i[1,3]C_v^i[2,3]) \quad (18)$$

$$Q_k[4,6] = \frac{\sigma_a^2 t_s^3}{3} (C_v^i[1,1]C_v^i[3,1] + C_v^i[1,2]C_v^i[3,2] + C_v^i[1,3]C_v^i[3,3]) \quad (19)$$

$$Q_k[4,7] = \frac{t_s^2 C_v^i[1,1] \sigma_a^2}{2} \quad (20)$$

$$Q_k[4,8] = \frac{t_s^2 C_v^i[1,2] \sigma_a^2}{2} \quad (21)$$

$$Q_k[4,9] = \frac{t_s^2 C_v^i[1,3] \sigma_a^2}{2} \quad (22)$$

$$Q_k[5,5] = \frac{\sigma_a^2 t_s^3}{3} \left((C_v^i[2,1])^2 + (C_v^i[2,2])^2 + (C_v^i[2,3])^2 \right) \quad (23)$$

$$Q_k[5,6] = \frac{\sigma_a^2 t_s^3}{3} (C_v^i[2,1]C_v^i[3,1] + C_v^i[2,2]C_v^i[3,2] + C_v^i[2,3]C_v^i[3,3]) \quad (24)$$

$$Q_k[5,7] = \frac{t_s^2 C_v^i[2,1] \sigma_a^2}{2} \quad (25)$$

$$Q_k[5,8] = \frac{t_s^2 C_v^i[2,2] \sigma_a^2}{2} \quad (26)$$

$$Q_k[5,9] = \frac{t_s^2 C_v^i[2,3] \sigma_a^2}{2} \quad (27)$$

$$Q_k[6,6] = \frac{\sigma_a^2 t_s^3}{3} \left((C_v^i[3,1])^2 + (C_v^i[3,2])^2 + (C_v^i[3,3])^2 \right) \quad (28)$$

$$\mathbf{Q}_k[6,7] = \frac{t_s^2 \mathbf{C}_v^i[3,1] \sigma_a^2}{2} \quad (29)$$

$$\mathbf{Q}_k[6,8] = \frac{t_s^2 \mathbf{C}_v^i[3,2] \sigma_a^2}{2} \quad (30)$$

$$\mathbf{Q}_k[6,9] = \frac{t_s^2 \mathbf{C}_v^i[3,3] \sigma_a^2}{2} \quad (31)$$

$$\mathbf{Q}_k[7,7] = t_s \sigma_a^2 \quad (32)$$

$$\mathbf{Q}_k[8,8] = t_s \sigma_a^2 \quad (33)$$

$$\mathbf{Q}_k[9,9] = t_s \sigma_a^2 \quad (34)$$

2.2. Radar Measurement Model

The radar considered in this study is a monostatic ground-based radar. The radar provides measurements of the target range, elevation angle, and azimuth angle of the vehicle in the radar measurement coordinate system. Figure 3 illustrates the geometric relationship between the vehicle and the radar station, where R denotes the distance from the vehicle T to the origin of the radar measurement coordinate system, and T' represents the projection of the vehicle onto the $x_i - z_i$ plane. The variables ε and β denote the elevation angle and azimuth angle, respectively.

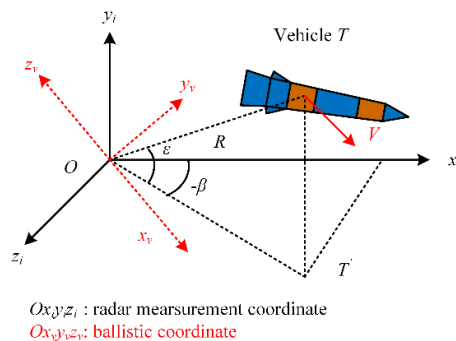


Figure 3. Relationship between the vehicle and the ground station.

In practical measurement processes, radar measurements are often affected by noise interference. Accordingly, the measurement vector is defined as Equation (35):

$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k \quad (35)$$

where $\mathbf{y}_k = [R \ \varepsilon \ \beta]^T$ denotes the radar measurement vector, and $\mathbf{h}(\mathbf{x}_k)$ is the observation function that maps the system state vector $\mathbf{x}_k$ to the measurement space. The mapping relationship between the observation vector and the state vector is given by Equations (36)–(38):

$$R = \sqrt{x^2 + y^2 + z^2} + v_R \quad (36)$$

$$\varepsilon = \arctan\left(\frac{y}{\sqrt{x^2 + z^2}}\right) + v_\varepsilon \quad (37)$$

$$\beta = \arctan\left(-\frac{z}{x}\right) + v_\beta \quad (38)$$

It is assumed that the measurement noise $\mathbf{v}$ represents zero-mean Gaussian white noise with covariance matrix $\mathbf{R}_k$, and that v_R , v_ε and v_β are mutually independent, each following a normal distribution with zero mean and variances σ_R^2 , σ_ε^2 , and σ_β^2 , respectively. Accordingly, the measurement noise covariance matrix $\mathbf{R}_k$ can be expressed as Equation (39):

$$\mathbf{R}_k = \begin{bmatrix} \sigma_R^2 & 0 & 0 \\ 0 & \sigma_\varepsilon^2 & 0 \\ 0 & 0 & \sigma_\beta^2 \end{bmatrix} \quad (39)$$

3. Conventional Approaches

The proposed Ht-KalmanNet is compared with six conventional filtering methods. Given that the EKF forms the model basis for both KalmanNet and Ht-KalmanNet, it is presented with particular emphasis. Furthermore, we briefly outline the fundamental principles and computational steps of the PF and UKF, along with the simulation conditions established in this study. As Ht-KalmanNet incorporates a neural network component, and given that CNN and LSTM are occasionally employed in trajectory tracking, both CNN and LSTM will be discussed in this section. Finally, KalmanNet is described, providing the foundation for the development of the proposed Ht-KalmanNet.

3.1. Extended Kalman Filter

The KalmanNet used in this paper is based on the EKF; thus, it is necessary to first introduce the EKF. While the standard Kalman filter is designed to handle linear state-space models, the EKF was developed to address nonlinear models. The EKF operates in two steps: prediction and update.

Unlike the standard Kalman filter, both the state evolution function and the observation function in the EKF are nonlinear. Therefore, the linearization of these functions is achieved by calculating the Jacobian matrices. In the prediction step, the nonlinear functions and their Jacobian matrices are computed to linearize the system, as shown in Equation (40). First, the Jacobian matrices of the nonlinear functions f and h are computed to achieve linearization, as shown in Equations (40) and (41).

$$\hat{\mathbf{F}}_t = \mathbf{J}_f(\hat{\mathbf{x}}_{t-1|t-1}) \quad (40)$$

$$\hat{\mathbf{H}}_t = \mathbf{J}_h(\hat{\mathbf{x}}_{t|t-1}) \quad (41)$$

In the prediction step, a prior estimate for the current state $\hat{\mathbf{x}}_{t|t-1}$ is computed using the prior estimate $\hat{\mathbf{x}}_{t-1|t-1}$, as shown in Equation (42). Meanwhile, a prior covariance $\Sigma_{t|t-1}$ is obtained based on the previous posterior covariance $\Sigma_{t-1|t-1}$ and the process noise $\mathbf{Q}$, as shown in Equation (43).

$$\hat{\mathbf{x}}_{t|t-1} = \hat{\mathbf{F}}_t \hat{\mathbf{x}}_{t-1|t-1} \quad (42)$$

$$\Sigma_{t|t-1} = \hat{\mathbf{F}}_t \Sigma_{t-1|t-1} \hat{\mathbf{F}}_t^T + \mathbf{Q} \quad (43)$$

A prior estimate for the current observation $\hat{\mathbf{y}}_{t|t-1}$ is computed using the observation matrix $\hat{\mathbf{H}}_t$, as shown in Equation (44). Subsequently, the covariance matrix of the pre-

dicted observation, denoted by $\hat{y}_{t|t-1}$, is computed based on the prior covariance $\Sigma_{t|t-1}$ and the observation noise covariance R , as shown in Equation (45).

$$\hat{y}_{t|t-1} = \hat{H}_t \hat{x}_{t|t-1} \quad (44)$$

$$S_{t|t-1} = \hat{H}_t \Sigma_{t|t-1} \hat{H}_t^T + R \quad (45)$$

Next, the KG is computed during the update step, as shown in Equation (46).

$$\mathcal{K}_t = \Sigma_{t|t-1} \hat{H}_t^T S_{t|t-1}^{-1} \quad (46)$$

Based on the prior estimates at the current time step, the posterior state estimate $\hat{x}_{t|t}$ and the posterior matrix $\Sigma_{t|t}$ are computed, as shown in Equations (47) and (48), respectively.

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + \mathcal{K}_t \Delta y_t \quad (47)$$

$$\Sigma_{t|t} = \Sigma_{t|t-1} - \mathcal{K}_t S_{t|t-1} \mathcal{K}_t^T \quad (48)$$

In Equation (47), Δy_t denotes the difference between the predicted observation and the observed value, which is the only term that depends on the observed data, as shown in Equation (49).

$$\Delta y_t = y_t - \hat{y}_{t|t-1} \quad (49)$$

An illustration of the EKF is depicted in Figure 4. The resulting filter admits an efficient linear recursive structure. However, since the EKF replaces the original state transition function f and observation function h with their first-order approximations obtained through the Jacobian matrices, certain model-based approximation errors are introduced. Moreover, when the system exhibits strong nonlinearity, the performance of the EKF further deteriorates. These limitations have motivated the development of KalmanNet, which integrates deep learning techniques to enhance and support the EKF framework.

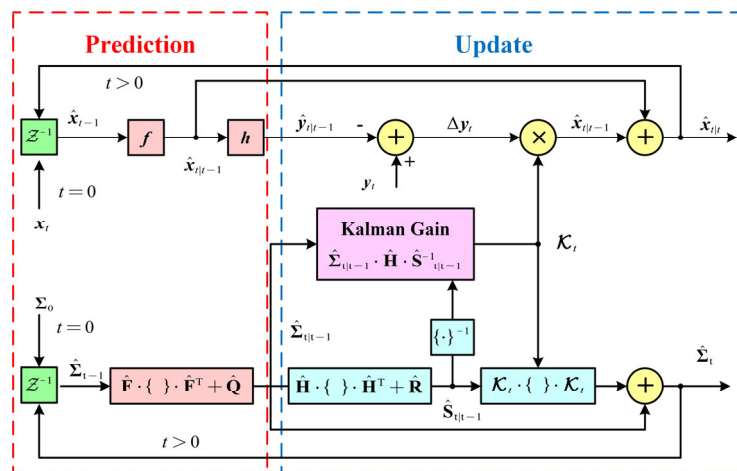


Figure 4. EKF block diagram.

3.2. Particle Filter

The PF is a computational approach that implements the optimal recursive Bayesian filter through Monte Carlo simulation. It avoids the need for analytical solutions or local

linearization for nonlinear or non-Gaussian problems by approximating the posterior probability density function using a large set of weighted particles.

The process of particle filtering is as follows: suppose we have N particles estimated from the posterior density $p(\mathbf{x}_t | \mathbf{y}_t)$ at time t . Let this particle set be denoted as $\Sigma_t = \{\mathbf{x}_t(i), i = 1, 2, \dots, N\}$. Particle filtering recursively propagates and updates the particle set to time $t+1$, resulting in the set $\Sigma_{t+1} = \{\mathbf{x}_{t+1}(i), i = 1, 2, \dots, N\}$, where the random particle set at time $t+1$ is approximately distributed according to the posterior density $p(\mathbf{x}_{t+1} | \mathbf{y}_{t+1})$.

There are many different schemes for performing the propagation of random samples, and thus different particle filtering algorithms exist [40]. In this paper, we use Sequential Importance Resampling (SIR) [5]. Sequential Importance Resampling propagates the random sample set through the following steps [28].

The fundamental idea of the PF is to represent the desired probability density function with a set of random samples, and as the number of particles increases, the approximation becomes more accurate.

3.3. Unscented Kalman Filter

The UKF is based on statistical linearization and constructs sigma points through the unscented transform. It updates the state and covariance within the standard Kalman filtering framework. Unlike the EKF, which linearizes the nonlinear model itself, the UKF approximates the state probability density function. Moreover, the UKF is not restricted by the system structure, does not require Jacobian matrices, and is capable of handling nondifferentiable nonlinear functions. The statistics obtained through the unscented transform can achieve higher accuracy than the EKF. The sampling strategy used in this study adopts the sigma-point method proposed by Merwe and Wan [8].

The procedure for the UKF algorithm is as follows.

- (1) Construct sigma points using UT transformation

$$\begin{aligned} \chi_t^{(0)} &= \mathbf{x}_{t|t} \\ \chi_t^{(i)} &= \mathbf{x}_{t|t} + \left(\sqrt{(m+\lambda) \mathbf{P}_{t|t}} \right)_i, i = 1, 2, \dots, m \\ \chi_t^{(i)} &= \mathbf{x}_{t|t} - \left(\sqrt{(m+\lambda) \mathbf{P}_{t|t}} \right)_i, i = m+1, m+2, \dots, 2m \end{aligned} \quad (50)$$

where $\mathbf{x}_{t|t}$ and $\mathbf{P}_{t|t}$ are the state estimate and covariance matrix at the current time t , respectively, m is the dimension of the state vector, and $\lambda = \varsigma^2(m+t) - m, \varsigma \in [0, 1]$ is used to control the distribution of sigma points, with $t = 3 - m$.

- (2) Propagate sigma points states. The sigma points are propagated through the state equation using the first expression in Equation (51):

$$\chi_{t+1|t}^{(i)} = \mathbf{f}(\chi_t^{(i)}), i = 0, 1, 2, \dots, 2m \quad (51)$$

- (3) Solve for state variables and covariance matrix using inverse unscented transformation. The weighting coefficients for the mean and covariance are defined as follows:

$$\begin{aligned} w_0^{(n)} &= \frac{\lambda}{m+\lambda} \\ w_0^{(c)} &= \frac{\lambda}{m+\lambda} + (1 - \varsigma^2 + \kappa), i = 1, 2, \dots, 2m \\ w_i^{(c)} &= w_i^{(n)} = \frac{1}{2(m+\lambda)} \end{aligned} \quad (52)$$

where $\mathbf{K}$ is a parameter related to the state prior distribution. For Gaussian distributions, the optimal value of $\mathbf{K}$ is typically set to 2. Then, the prior state estimate and its error covariance matrix at time $k+1$ are as follows:

$$\begin{aligned}\mathbf{x}_{t+1|t} &= \sum_{i=0}^{2m} w_i^{(n)} \chi_{t+1|t}^{(i)} \\ \mathbf{P}_{t+1|t} &= \sum_{i=0}^{2n} w_i^{(c)} \left(\chi_{t+1|t}^{(i)} - \mathbf{x}_{t+1|t} \right) \left(\chi_{t+1|t}^{(i)} - \mathbf{x}_{t+1|t} \right)^T + \mathbf{Q}_t\end{aligned}\quad (53)$$

(4) Reconstruct sigma points using UT transformation. Use $\mathbf{x}_{t+1|t}$ and $\mathbf{P}_{t+1|t}$ obtained in Equation (53) to reconstruct sigma points:

$$\begin{aligned}\chi_t^{(0)} &= \mathbf{x}_{t+1|t} \\ \chi_{t+1}^{(i)} &= \mathbf{x}_{t+1|t} + \left(\sqrt{(m+\lambda) \mathbf{P}_{t+1|t}} \right)_i, i=1, 2, \dots, m \\ \chi_{t+1}^{(i)} &= \mathbf{x}_{t+1|t} - \left(\sqrt{(m+\lambda) \mathbf{P}_{t+1|t}} \right)_i, i=m+1, m+2, \dots, 2m\end{aligned}\quad (54)$$

(5) Propagate sigma point observations. Use the Equation (55) to solve for the propagation of sigma points through the observation equation:

$$\mathbf{y}_{t+1}^{(i)} = \mathbf{h} \left(\chi_{t+1}^{(i)} \right) \quad (55)$$

(6) Solve for observation variables and their related covariances using UT inverse transformation.

$$\begin{aligned}\mathbf{y}_{t+1|t} &= \sum_{i=0}^{2m} w_i^{(n)} \mathbf{y}_{t+1}^{(i)} \\ \mathbf{P}_{\mathbf{y}_{t+1}} &= \sum_{i=0}^{2m} w_i^{(c)} \left(\mathbf{y}_{t+1}^{(i)} - \mathbf{y}_{t+1|t} \right) \left(\mathbf{y}_{t+1}^{(i)} - \mathbf{y}_{t+1|t} \right)^T + \mathbf{R}_t \\ \mathbf{P}_{\mathbf{x}_{t+1}, \mathbf{y}_{t+1}} &= \sum_{i=0}^{2m} w_i^{(c)} \left(\chi_{t+1}^{(i)} - \mathbf{x}_{t+1|t} \right) \left(\mathbf{y}_{t+1}^{(i)} - \mathbf{y}_{t+1|t} \right)^T\end{aligned}\quad (56)$$

(7) Update the state and covariance matrix.

$$\begin{aligned}\mathbf{x}_{t+1|t+1} &= \mathbf{x}_{t+1|t} + \mathbf{P}_{\mathbf{x}_{t+1}, \mathbf{y}_{t+1}} \mathbf{P}_{\mathbf{y}_{t+1}}^{-1} \left(\mathbf{y}_{t+1} - \mathbf{y}_{t+1|t} \right) \\ \mathbf{P}_{t+1|t+1} &= \mathbf{P}_{t+1|t} - \mathbf{P}_{\mathbf{x}_{t+1}, \mathbf{y}_{t+1}} \mathbf{P}_{\mathbf{y}_{t+1}}^{-1} \mathbf{P}_{\mathbf{y}_{t+1}}^T\end{aligned}\quad (57)$$

3.4. Convolutional Neural Network

A CNN typically consists of an input layer, convolutional layers, pooling layers, fully connected layers, and an output layer. The input, convolutional, and pooling layers constitute the feature-extraction module, while the fully connected and output layers constitute the classification module.

The convolutional layers serve as feature extractors. Each neuron is connected to a local receptive field in the preceding layer to extract local features, and once these features are obtained, their spatial relationships with other features are implicitly encoded. The primary function of the convolutional layer is to perform convolution operations using convolution kernels; such operations enhance meaningful features while suppressing noise. The pooling layer performs down sampling over local regions of the feature maps, thereby reducing dimensionality and mitigating overfitting. Pooling reduces dimensionality, mitigates overfitting, and provides a degree of robustness to geometric variations. The fully connected layer integrates all extracted features and passes the output to a classifier, thereby producing the final network output.

In regression-based tracking tasks, CNNs can directly map measurement features to target state estimates. To clearly highlight the advantages and limitations of the proposed

method relative to purely numerical approaches, this study employs a three-layer CNN as a comparative baseline.

3.5. Long Short-Term Memory Neural Network

A vanilla RNN is constructed by defining the transition function and the output function, as follows:

$$\begin{aligned} s_k &= f_s(i_k, s_{k-1}) = \phi_s(W^{si}i_k + W^{ss}s_{k-1}) \\ o_k &= f_o(s_k) = \phi_o(W^{os}s_k) \end{aligned} \quad (58)$$

where W^{si} , W^{ss} and W^{os} are the input, transition, and output matrices, respectively, and ϕ_s and ϕ_o are element-wise nonlinear functions. For ϕ_s and ϕ_o , saturating nonlinear functions are commonly used, such as the logistic sigmoid function $\sigma(x)$ or the hyperbolic tangent function $\tanh(x)$, as follows:

$$\begin{aligned} \sigma(x) &= \frac{1}{1 + \exp(-x)} \\ \tanh(x) &= \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \end{aligned} \quad (59)$$

It is worth noting that an RNN is already a type of DNN because multiple nonlinear computational layers exist when it is unfolded in time. Therefore, if the learned sequences are too long, the parameters cannot be learned efficiently due to the vanishing- or exploding-gradient problem during backpropagation [12]. The LSTM is specifically designed to address this issue.

The main difference between different RNN structures lies in the repeating module that defines the state transition function. In a vanilla RNN, this repeating module has only a single-layer structure, as in Equation (58), whereas for LSTM, there are four layers interacting in a very specific way, as in Equation (60). Figure 5 illustrates a typical LSTM module.

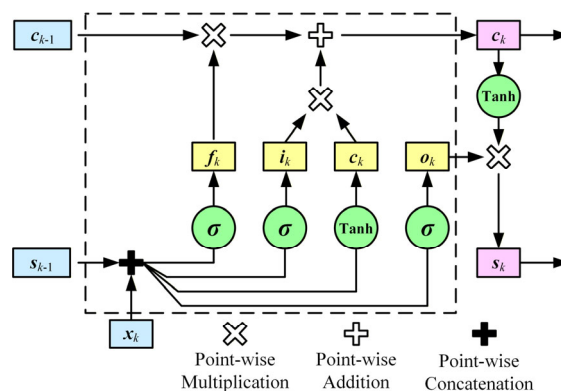


Figure 5. An LSTM module.

The core idea of LSTM lies in the cell state c_k , which is represented by the horizontal line running through the top in Figure 5. The cell state acts as a memory and information carrier. Furthermore, the information added to or removed from the cell state is strictly controlled by gating mechanisms, which are typically composed of sigmoid neural network layers and pointwise multiplication operations. Assuming that the activation functions in the nonlinear neural network layers are all hyperbolic tangent functions, the calculation formulas for the forget gate f_k , input gate i_k , cell state c_k , output gate o_k , and hidden state s_k , at time step k are given as follows.

$$\begin{aligned}
f_k &= \sigma(\mathbf{W}^f \cdot [\mathbf{s}_{k-1}, \mathbf{x}_k] + \mathbf{b}^f) \\
i_k &= \sigma(\mathbf{W}^i \cdot [\mathbf{s}_{k-1}, \mathbf{x}_k] + \mathbf{b}^i) \\
c_k &= f_k \odot c_{k-1} + i_k \odot \tanh(\mathbf{W}^c \cdot [\mathbf{s}_{k-1}, \mathbf{x}_k] + \mathbf{b}^c) \\
o_k &= \sigma(\mathbf{W}^o \cdot [\mathbf{s}_{k-1}, \mathbf{x}_k] + \mathbf{b}^o) \\
s_k &= o_k \odot \tanh(c_k)
\end{aligned} \tag{60}$$

where $\mathbf{W}^\bullet$ and $\mathbf{b}^\bullet$ are learnable parameters, and $\bullet$ denotes the superscripts f , i , and o . The symbol $\odot$ represents the Hadamard product, i.e., element-wise multiplication between vectors. Unlike the vanilla RNN in Equation (58), the LSTM updates both the cell state and the hidden state at each time step. The output of the LSTM module is decoded from the hidden state in a manner similar to that of a vanilla RNN.

The forget gate plays a crucial role in enabling backpropagation through time (BPTT), allowing gradient information to propagate effectively across long temporal sequences during training [16]. As a result, long-term sequential dependencies can be captured more effectively. To evaluate the tracking performance of recurrent neural networks in reentry trajectory-tracking problems, a four-layer LSTM architecture was employed in this study.

3.6. KalmanNet

The KalmanNet adopted in this study is an interpretable hybrid data-driven model capable of real-time state estimation in nonlinear systems with partially known dynamics. KalmanNet is developed based on the EKF framework and similarly consists of prediction and update stages. The state evolution and observation functions are still approximated through Jacobian-based linearization. Unlike the EKF, KalmanNet provides only the posterior estimate and does not output the posterior covariance.

When computing the KG, KalmanNet does not explicitly compute the covariance matrices $\Sigma_{t|t-1}$ and $S_{t|t-1}$, nor does it require the matrix inversion involved in Equation (46). Instead, it employs a neural network to learn the KG directly, thereby significantly reducing the computational cost. Moreover, KalmanNet is not tied to predefined process noise $\mathbf{Q}$ or measurement noise $\mathbf{R}$; rather, it implicitly infers the relevant information from data, thus decreasing reliance on the original model.

The GRUs used in KalmanNet extract a set of features to estimate the KG. Following the input-feature design proposed in [36], the observation difference $\Delta \tilde{\mathbf{y}}_t$, the forward evolution difference $\Delta \tilde{\mathbf{x}}_t$, and the forward update difference $\Delta \hat{\mathbf{x}}_t$ are adopted as input features, and the corresponding RNN architecture is constructed accordingly, as illustrated in Figure 6.

The observation difference $\Delta \tilde{\mathbf{y}}_t$, the forward evolution difference $\Delta \tilde{\mathbf{x}}_t$, and the forward update difference $\Delta \hat{\mathbf{x}}_t$ are computed as follows:

$$\Delta \tilde{\mathbf{y}}_t = \mathbf{y}_t - \mathbf{y}_{t-1} \tag{61}$$

$$\Delta \tilde{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t} - \hat{\mathbf{x}}_{t-1|t-1} \tag{62}$$

$$\Delta \hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t} - \hat{\mathbf{x}}_{t|t-1} \tag{63}$$

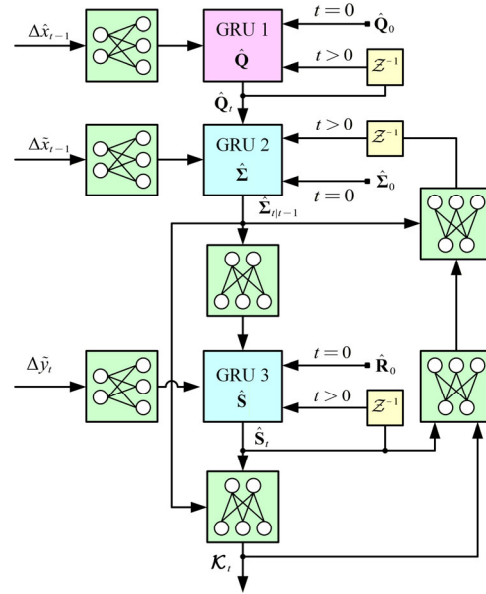


Figure 6. RNN block diagram of KalmanNet.

4. Ht-KalmanNet

When KalmanNet is applied to trajectory tracking, the time-varying characteristics of the observation matrix are not explicitly incorporated into the KG estimation process. This mismatch may lead to suboptimal Kalman gain estimation due to insufficient modeling of the time-varying observation matrix. In this section, the RNN architecture of KalmanNet is revisited, and improvements are introduced by integrating the algorithmic procedures of the EKF, leading to the development of the proposed Ht-KalmanNet framework.

4.1. RNN Architecture of Ht-KalmanNet

To properly design the neural network for estimating the KG, we first revisit the computation of the KG $\mathcal{K}_t$ in the EKF, as shown in Equation (64).

$$\mathcal{K}_t = \Sigma_{t|t-1} \mathbf{H}_t^T \mathbf{S}_{t|t-1}^{-1} \quad (64)$$

According to Equation (64), the computation of the KG requires the prior covariance $\Sigma_{t|t-1}$, the transpose of observation matrix $\mathbf{H}_t^T$, and the inverse of the prior covariance for the current observation $\mathbf{S}_{t|t-1}$. Clearly, $\Sigma_{t|t-1}$, $\mathbf{H}_t^T$, and $\mathbf{S}_{t|t-1}$ are all time-varying quantities in the state-space model of the reentry model. As illustrated in Figure 6, the original RNN architecture of KalmanNet neglects the modeling of the time-varying observation matrix $\mathbf{H}_t^T$, resulting in inconsistency with the mathematical formulation of the KG in the EKF framework. Consequently, the effectiveness of KalmanNet is mainly limited to scenarios with linear observation models, where the observation matrix remains constant over time. To overcome this limitation, three dedicated GRU modules are introduced to estimate the corresponding second-order statistical moments. The overall architecture of the proposed RNN is shown in Figure 7.

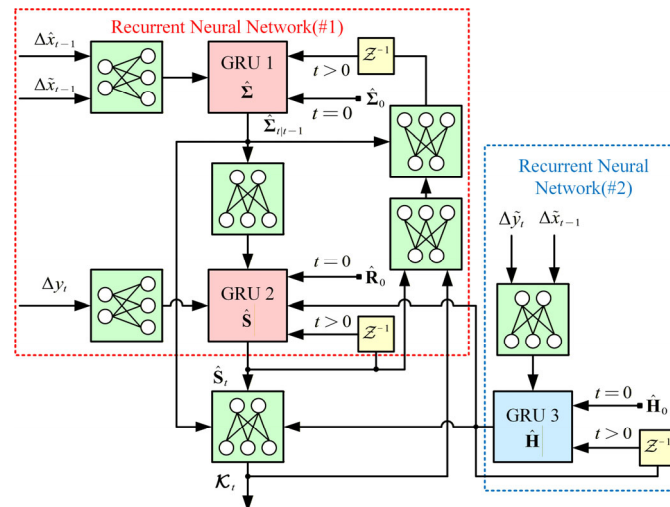


Figure 7. RNN block diagram of Ht-KalmanNet.

First, based on the analysis of Figure 6, GRU1 associated with the process noise $\hat{\mathbf{Q}}$ only serves as an auxiliary input to GRU2 for estimating $\hat{\Sigma}_{t|t-1}$ and does not directly contribute to the estimation of other statistical quantities. Moreover, the estimation of the process noise $\hat{\mathbf{Q}}$ is not directly coupled with other RNN components. According to the analysis of the EKF process in Section 3.1, the process noise influences only the computation of $\hat{\Sigma}_{t|t-1}$ and has no direct impact on other statistical moments. Since the number of neurons in a GRU directly determines the dimensionality of the hidden state [41], GRU1 with 81 hidden state variables introduces considerable computational and memory overhead in terms of memory usage and computational cost. Therefore, GRU1 is removed from the original RNN architecture, and its corresponding input feature $\Delta\tilde{\mathbf{x}}_t$ is used directly as an input for estimating $\hat{\Sigma}_{t|t-1}$, effectively merging the functionalities of GRU1 and GRU2. In this case, the GRU layer to estimate $\hat{\Sigma}_{t|t-1}$ named as GRU1 in Figure 7 takes the forward evolution difference $\Delta\tilde{\mathbf{x}}_{t-1}$ and forward update difference $\Delta\hat{\mathbf{x}}_{t-1}$ as input features to estimate the prior covariance matrix $\hat{\Sigma}_{t|t-1}$, which contains 81 state variables.

Next, we discuss GRU3, which is designed to track the time-varying observation matrix $\hat{\mathbf{H}}_t$. In this case, the observation difference $\Delta\tilde{\mathbf{y}}_t$ and forward evolution difference $\Delta\tilde{\mathbf{x}}_{t-1}$ are used as input features to independently estimate $\hat{\mathbf{H}}_t$. The resulting estimated observation matrix carries clear physical interpretability in the nonlinear radar measurement considered in this study, where the measured range varies over time, the elevation angle spans from 0° to 90° , and the azimuth angle ranges from -90° to 90° . Since the state-space model adopted in this work contains nine state variables and three measurement variables, the observation matrix consists of 27 elements. Accordingly, the hidden layer of GRU3 is configured with 27 hidden state variables, matching the dimensionality of the elements in $\hat{\mathbf{H}}_t$.

The GRU module is used to estimate the prior observation covariance matrix for the current observation $\mathbf{S}_{t|t-1}$ of the observation variables. In the process of the EKF, the $\mathbf{S}_{t|t-1}$ is given by Equation (65).

$$\mathbf{S}_{t|t-1} = \hat{\mathbf{H}}_t \hat{\Sigma}_{t|t-1} \hat{\mathbf{H}}_t^T + \mathbf{R} \quad (65)$$

As indicated by Equation (65), the computation of $\mathbf{S}_{t|t-1}$ depends on both $\mathbf{\Sigma}_{t|t-1}$ and $\hat{\mathbf{H}}_t$. Therefore, the inputs to GRU2 consist of two feature groups: one is the estimated $\hat{\mathbf{\Sigma}}_{t|t-1}$ produced from the output of GRU1, and the other is the estimated $\hat{\mathbf{H}}_t$ generated by GRU3. These two components are concatenated and fed into GRU2. Since $\mathbf{S}_{t|t-1}$ is a 3×3 matrix, GRU2 is configured with nine hidden state variables.

Finally, following the computation of the KG in Equation (64), a fully connected layer is employed to take the estimated matrices $\hat{\mathbf{\Sigma}}_{t|t-1}$, $\mathbf{S}_{t|t-1}$, and $\hat{\mathbf{H}}_t$ as inputs, which are produced by the three GRU layers, and its output serves as the fitted KG $\mathcal{K}_t$. Considering the characteristics of the proposed model, the overall architecture of the RNN is designed as shown in Figure 8. In this structure, the RNN(#1) module consists of GRU1, GRU2, and their corresponding fully connected layers, which are used to estimate $\hat{\mathbf{\Sigma}}_{t|t-1}$ and $\hat{\mathbf{S}}_{t|t-1}$; the RNN(#2) module includes GRU3 and its associated fully connected layer, which is used to estimate $\hat{\mathbf{H}}_t$. The outputs from RNN(#1) and RNN(#2) are subsequently integrated through a fully connected layer to produce the final estimation of the KG $\mathcal{K}_t$.

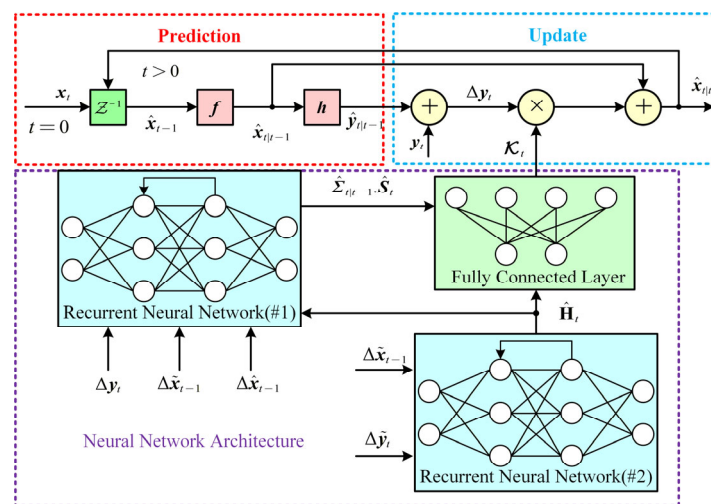


Figure 8. Ht-KalmanNet block diagram.

With this design, the number of GRUs and fully connected layers in Ht-KalmanNet remains consistent with those in the original KalmanNet architecture proposed in [36]. As a result, the model enhances the physical consistency of the tracking process without increasing the architectural complexity, thereby maintaining computational speed while improving estimation accuracy.

It should be noted that the learned observation matrix in this work can be interpreted as an approximation of the analytical Jacobian matrix; however, it is not identical to it. The analytical Jacobian matrix used in model linearization inevitably introduces approximation errors for nonlinear systems. Therefore, the neural network design for estimating the time-varying observation matrix is guided by the computation principle of the analytical Jacobian matrix.

Specifically, since the analytical Jacobian matrix corresponds to the partial derivatives of the observation function with respect to the state variables, we construct the network inputs in a physically consistent manner. For the observation term, we use observation differences as input features, since they approximate the temporal variation in the

measurements. For the state term, we use forward evolution difference, which represents the differences between successive posterior state estimates. This design ensures that the network inputs closely follow the computation mechanism of the analytical Jacobian matrix.

4.2. Architecture of Ht-KalmanNet

Based on the improved RNN architecture shown in Figure 7, a filtering algorithm distinct from the original KalmanNet is developed and named Ht-KalmanNet. Its overall framework is illustrated in Figure 8. RNN(#1) takes the forward evolution difference $\Delta\tilde{\mathbf{x}}_{t-1}$, forward update difference $\Delta\hat{\mathbf{x}}_{t-1}$, and innovation difference $\Delta\mathbf{y}_t$ as input features, while RNN(#2) uses the observation difference $\Delta\tilde{\mathbf{y}}_t$ and forward evolution difference $\Delta\tilde{\mathbf{x}}_{t-1}$ as its input features. Note that RNN(#1) and RNN(#2) are not independent. As described earlier, the output of RNN(#2), namely the estimated observation matrix $\hat{\mathbf{H}}_t$, also contributes to the computation of $\hat{\mathbf{S}}_{t|t-1}$, and is thus fed into RNN(#1). Finally, the outputs of RNN(#1) and RNN(#2) are fused through a fully connected layer to produce the estimated KG $\mathcal{K}_t$, which is then used in the update step to refine the posterior estimate $\hat{\mathbf{x}}_{t|t}$ according to Equation (66).

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathcal{K}_t \Delta\mathbf{y}_t \quad (66)$$

This study adopts the observation difference $\Delta\tilde{\mathbf{y}}_t$, the innovation difference $\Delta\mathbf{y}_t$, the forward evolution difference $\Delta\tilde{\mathbf{x}}_t$, and the forward update difference $\Delta\hat{\mathbf{x}}_t$ adopted as the input features of the RNN within the Ht-KalmanNet, which are respectively expressed in Equations (67)–(70).

$$\Delta\tilde{\mathbf{y}}_t = \mathbf{y}_t - \mathbf{y}_{t-1} \quad (67)$$

$$\Delta\mathbf{y}_t = \mathbf{y}_t - \hat{\mathbf{y}}_{t|t-1} \quad (68)$$

$$\Delta\tilde{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t} - \hat{\mathbf{x}}_{t-1|t-1} \quad (69)$$

$$\Delta\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t} - \hat{\mathbf{x}}_{t|t-1} \quad (70)$$

4.3. Training Method

The Ht-KalmanNet is trained in a supervised manner using simulated trajectory datasets. Although the RNN within Ht-KalmanNet primarily learns to approximate the KG $\mathcal{K}_t$, to compensate for the limitations caused by model linearization, an end-to-end training strategy is adopted. Specifically, KalmanNet is trained using the true state $\mathbf{x}_t$ and the estimated state $\hat{\mathbf{x}}_{t|t}$ to compute the loss function, which is defined as the mean squared error (MSE), as shown in Equation (71):

$$\mathcal{L} = \|\mathbf{x}_t - \hat{\mathbf{x}}_{t|t}\|^2 \quad (71)$$

This loss function is equivalent to the evaluation metric used in traditional Kalman filtering. Moreover, the loss can be backpropagated through the network to update the parameters responsible for computing the KG $\mathcal{K}_t$. The corresponding loss gradient, derived from the KG $\mathcal{K}_t$ output of Ht-KalmanNet, is expressed in Equation (72). In addi-

tion, such an end-to-end training approach eliminates the need to explicitly provide the true KG $\mathcal{K}_i$ during training, thereby simplifying the overall training process.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathcal{K}_i} &= \frac{\|\mathcal{K}_i \Delta \mathbf{y}_i - \Delta \mathbf{x}_i\|^2}{\partial \mathcal{K}_i} \\ &= 2(\mathcal{K}_i \Delta \mathbf{y}_i - \Delta \mathbf{x}_i) \Delta \mathbf{y}_i^T\end{aligned}\quad (72)$$

For the training dataset of Ht-KalmanNet, let T_i denote the length of the i -th trajectory. The dataset can then be expressed as $\mathcal{D} = \{(\mathbf{Y}_i, \mathbf{X}_i)\}_1^N$ where

$$\mathbf{Y}_i = [\mathbf{y}_1^{(i)}, \dots, \mathbf{y}_{T_i}^{(i)}], \mathbf{X}_i = [\mathbf{x}_0^{(i)}, \mathbf{x}_1^{(i)}, \dots, \mathbf{x}_{T_i}^{(i)}] \quad (73)$$

Let $\boldsymbol{\theta}$ denote the trainable parameters of the recurrent neural network, and let γ be a regularization coefficient. Based on this, a ℓ_2 -regularized MSE loss function is established as Equation (74).

$$\ell_i(\boldsymbol{\theta}) = \frac{1}{T_i} \sum_{t=1}^{T_i} \|\hat{\mathbf{x}}_t(\mathbf{y}_t^{(i)}; \boldsymbol{\theta}) - \mathbf{x}_t^{(i)}\|^2 + \gamma \cdot \|\boldsymbol{\theta}\|^2 \quad (74)$$

To optimize $\boldsymbol{\theta}$, a mini-batch stochastic gradient descent (SGD) variant is employed. Specifically, for the k -th batch, $M < N$ trajectories are randomly selected, denoted as $i_1^k, \dots, i_M^k$. The mini-batch loss is then computed as Equation (75).

$$\mathcal{L}_k(\boldsymbol{\theta}) = \frac{1}{M} \sum_{j=1}^{M_i} \ell_{i_j^k}(\boldsymbol{\theta}) \quad (75)$$

Because Ht-KalmanNet adopts a recursive architecture with both external recurrence and an internal recurrent neural network, we train the model using backpropagation through time (BPTT) [42]. Given the long trajectory durations in our case, training on full-length sequences may lead to the gradient vanishing or exploding. Therefore, we employ the truncated BPTT algorithm [43], in which long trajectories are segmented into shorter sequences of fixed length, randomly shuffled, and then used for training. The Adam optimizer is adopted during the training process [44].

4.4. Implementation Steps

The overall flow chart of the proposed algorithm is illustrated in Figure 9. By integrating the offline training and online inference procedures of Ht-KalmanNet, the overall implementation procedure can be summarized as follows:

Step 1: Perform offline training of Ht-KalmanNet using truncated BPTT and the Adam optimizer. In this step, we need to obtain the vehicle state data and the radar measurement data in the radar measurement coordinate system for neural network training.

Step 2: Carry out the prediction step of online trajectory tracking after offline training, to compute the prior state estimate $\hat{\mathbf{x}}_{t|t-1}$ with Equations (40) and (42) and a prior estimate for the current observation $\hat{\mathbf{y}}_{t|t-1}$ with Equations (41) and (44).

Step 3: Construct the input feature vector $\Delta \tilde{\mathbf{y}}_t$, the innovation difference $\Delta \mathbf{y}_t$, the forward evolution difference $\Delta \tilde{\mathbf{x}}_t$, and the forward update difference $\Delta \hat{\mathbf{x}}_t$ with Equations (67)–(70). These quantities are used as input features for the neural network in Ht-KalmanNet, which outputs the learned KG.

Step 4: Compute a posterior estimate $\hat{\mathbf{x}}_{t|t}$ according to Equation (66) with the prior estimate $\hat{\mathbf{x}}_{t|t-1}$, the innovation difference $\Delta \mathbf{y}_t$, and the KG.

Step 5: Repeat the above recursive estimation procedures 2–4 until the tracking of the entire trajectory is completed.

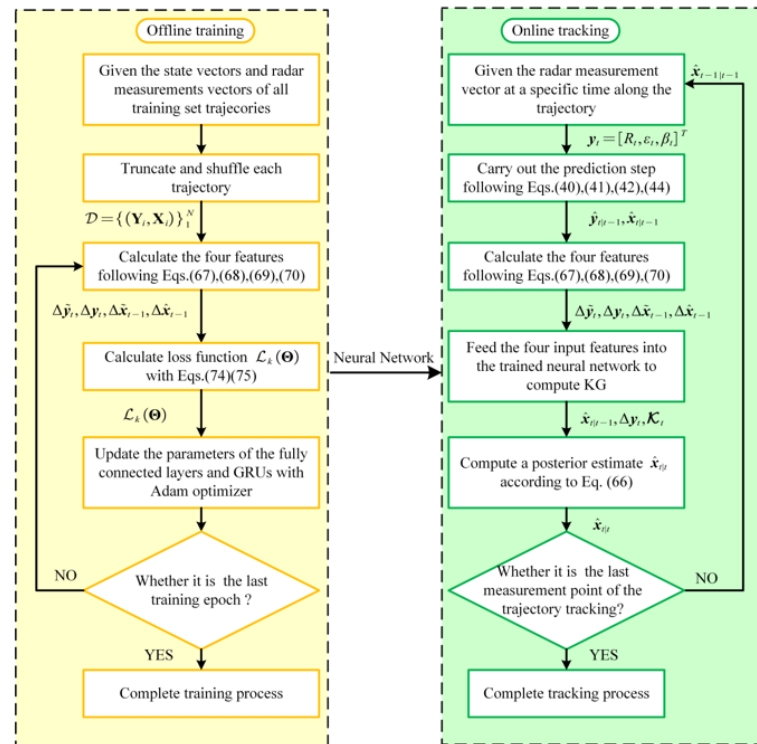


Figure 9. Flowchart of proposed method.

5. Simulation Results and Discussion

In this section, simulation scenarios are designed to evaluate and compare the performance of different filtering methods. Monte Carlo simulations are conducted to assess both the tracking accuracy and computational cost of seven filtering methods, including the EKF, PF, UKF, CNN, LSTM, KalmanNet, and Ht-KalmanNet. Furthermore, under four types of model mismatch conditions—bias in atmospheric density and aerodynamic coefficients, the introduction of a sideslip angle to generate lateral maneuvering, bias in range measurement STD, and bias in angle measurement STD—Monte Carlo simulations are subsequently conducted to evaluate the robustness and effectiveness of the proposed Ht-KalmanNet framework.

5.1. Simulation Parameters

The initial sampled state vector $\mathbf{x}$ is defined as follows:

$$\mathbf{x}_0 = [x_0 \quad y_0 \quad z_0 \quad v_{x0} \quad v_{y0} \quad v_{z0} \quad a_{D0} \quad a_{Y0} \quad a_{L0}]^T \quad (76)$$

where the corresponding sampling parameters are listed in Table 1. Here, μ_0 represents the sampling mean, σ_0 denotes the sampling STD, and all state variables are assumed to be mutually independent.

Table 1. Initial state sampling parameters.

	x_0 (m)	y_0 (m)	z_0 (m)	v_{x0} (m/s)	v_{y0} (m/s)	v_{z0} (m/s)	a_{D0} (m/s ²)	a_{Y0} (m/s ²)	a_{L0} (m/s ²)
μ_0	−2000	70,000	10,000	1000	−1120	800	0	0	0
σ_0	100	100	100	10	10	10	0.01	0.01	0.01

The angle of attack α_0 and sideslip angle β_0 in the dynamic model are both 0° . The total number of sampling points for each trajectory is $T = 500$. In the radar observa-

tion model, the STD of the range measurement σ_r is 100 m, the STD of the elevation angle measurement σ_ϵ is 1 mrad, and the STD of the azimuth angle measurement σ_β is 1 mrad. The STD of aerodynamic acceleration σ_a is set to 0.1 m/s². Based on these settings, an initial trajectory dataset is generated using the EKF with a time interval of 0.01 s. During the filtering computation, the sampling period for each trajectory is set to 0.1 s, and the total number of sampling points for each trajectory is set to 500. A total of 100 trajectories are generated for testing. The original test trajectories are shown in Figure 10.

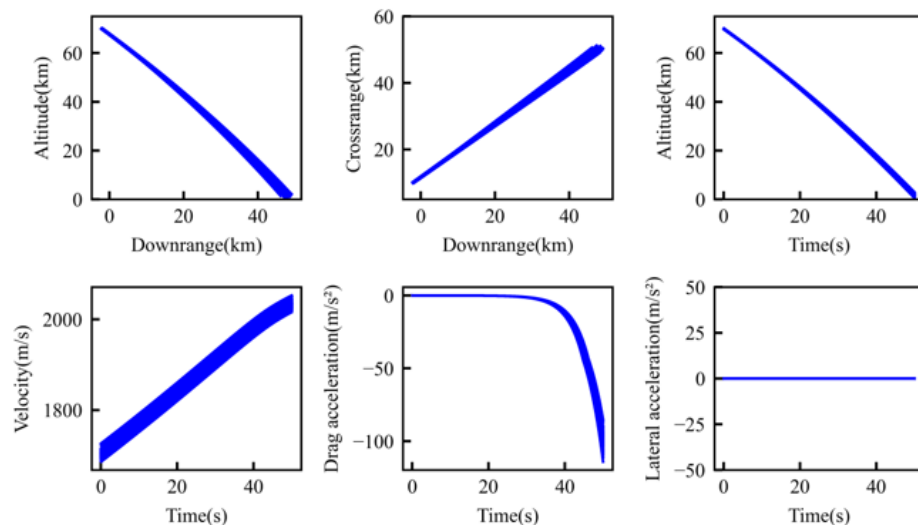


Figure 10. Trajectory plot of nominal simulation.

Figure 10 illustrates the trajectories in the training dataset generated using the initial conditions listed in Table 1, in which both the angle of attack and the sideslip angle are set to zero. As the flight time increases, the drag acceleration gradually increases, while the lateral acceleration remains approximately zero due to the absence of a sideslip motion.

The CNN, LSTM, KalmanNet, and Ht-KalmanNet models employed in this study are all trained using raw datasets generated under the simulation conditions described in Section 5.1. The relevant hyperparameter settings are shown in Table 2. In Section 5.2, the test dataset is also generated using the same computational settings specified in Section 5.1. The subsequent testing trajectories are produced under model-mismatch conditions to verify the effectiveness and robustness of the proposed model.

Table 2. Hyperparameter settings for different methods.

	CNN	LSTM	KalmanNet	Ht-KalmanNet
Learning Rate	10^{-3}	0.5	10^{-4}	10^{-4}
Batch Size	256	256	256	256
Epochs	10^5	10^4	10^4	10^4
Weight Decay	10^{-4}	10^{-4}	10^{-4}	10^{-4}

The training, validation, and initial test datasets were generated under identical conditions. Specifically, the training and validation datasets contained 10,000 and 1000 trajectories, respectively, whereas the initial test dataset consisted of 100 trajectories. In this study, the number of particles is set to 10,000. All simulations were performed in Python 3.12.8 on a laptop equipped with an Intel(R) Core(TM) i9-14900HX processor.

5.2. Simulation Results of the Original Test Set

The RMSE of position and velocity tracking was calculated for all sampling points of each trajectory. The RMSE calculation result curves for position-tracking and velocity-tracking errors of 100 trajectories under different filtering methods are shown in Figure 11. The calculation results of different filtering methods are presented in Table 3.

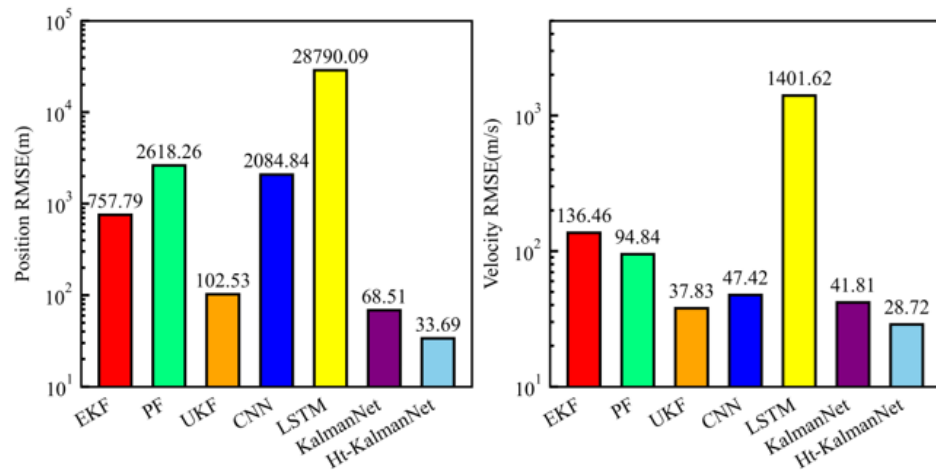


Figure 11. RMSE for position and velocity tracking under nominal conditions.

Table 3. Simulation results under nominal conditions.

		EKF	PF	UKF	CNN	LSTM	KalmanNet	Ht-KalmanNet
Position	RMSE (m)	758.11	2618.26	102.53	2084.84	28,790.09	68.51	33.69
	STD (m)	5.10	2122.33	34.17	123.72	167.96	35.36	7.40
Velocity	RMSE (m/s)	136.46	94.84	37.83	47.42	1401.62	41.81	28.72
	STD (m/s)	1.34	72.59	10.98	4.69	8.99	12.13	10.82
Run-time (s)		517.04	954.16	850.81	0.043	0.77	1.72	1.63

As shown in Table 3 and Figure 11, among the traditional filtering algorithms, the EKF and PF exhibit relatively low tracking accuracy for strongly nonlinear models such as those of reentry vehicles. In contrast, the UKF achieves significantly higher accuracy because the unscented transform provides a higher-order approximation of nonlinear transformations. Among the neural-network-based approaches, the CNN and LSTM exhibit relatively poor tracking performance because they rely primarily on data-driven learning without explicitly incorporating physical model information.

In addition, the original KalmanNet already achieves higher estimation accuracy than the UKF while maintaining relatively fast computational speed. In the proposed Ht-KalmanNet, a dedicated GRU module is designed to estimate the linearized observation matrix of the observation function, thereby further improving the estimation accuracy and outperforming the original KalmanNet. Although the recurrent neural network structures of Ht-KalmanNet and KalmanNet differ in architecture, the numbers of GRU units and fully connected layers remain unchanged. Consequently, the proposed method significantly improves the estimation accuracy while maintaining nearly the same computational efficiency as the original KalmanNet.

In addition, the time histories of the position tracking error and velocity tracking error for each filtering method are shown in Figure 12 and Figure 13, respectively. Although both KalmanNet and Ht-KalmanNet achieve relatively low position tracking errors, the standard deviation of the velocity tracking error in KalmanNet increases significantly. In

contrast, the corresponding standard deviation of Ht-KalmanNet remains at a relatively low level, indicating more stable tracking performance.

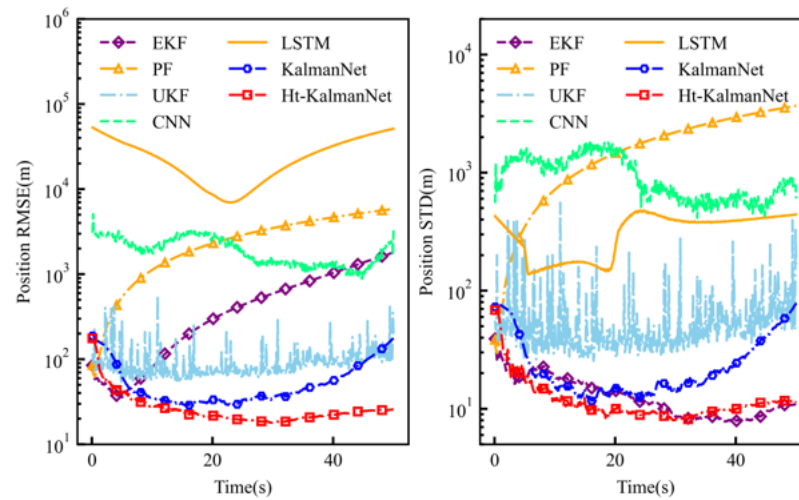


Figure 12. Position-tracking error curves under nominal conditions.

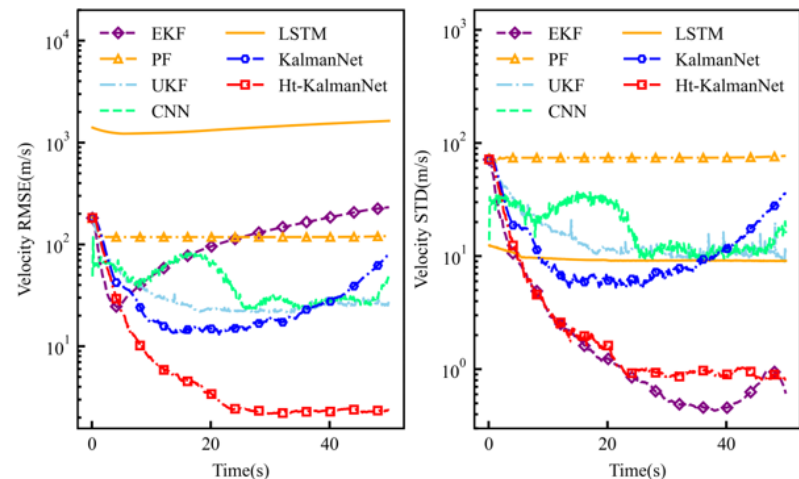


Figure 13. Velocity-tracking error curves under nominal conditions.

5.3. Lateral Maneuvering

To evaluate the robustness of different filtering methods under maneuvering conditions, the models trained using the nominal dataset are applied to track trajectories with lateral maneuvering. Specifically, a constant sideslip angle is introduced to generate lateral maneuvering. The sideslip angle is assumed to follow a Gaussian distribution with a mean of 15° and a standard deviation of 2° . Based on this setting, 100 test trajectories are generated, and the corresponding trajectories are shown in Figure 14.

As shown in Figure 14, the introduction of nonzero sideslip angles generates noticeable lateral acceleration components. Moreover, the lateral acceleration gradually increases with flight time and atmospheric density. The trajectory tracking under lateral maneuvering is then performed using the original model, and the corresponding results are presented in Figure 15 and Table 4.

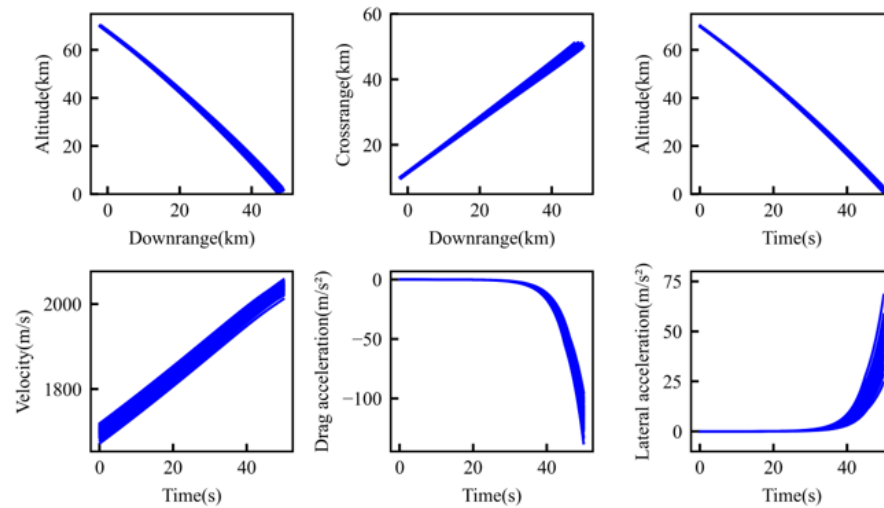


Figure 14. Trajectory plot under lateral maneuvering.

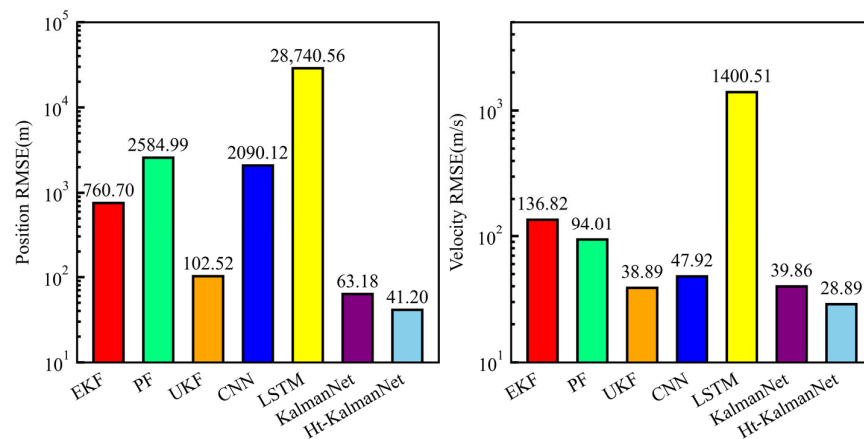


Figure 15. RMSE of position and velocity during lateral maneuvering.

Table 4. Simulation results for lateral maneuvering.

		EKF	PF	UKF	CNN	LSTM	KalmanNet	Ht-KalmanNet
Position	RMSE (m)	760.70	2584.99	102.52	2090.12	28,740.56	63.18	41.20
	STD (m)	4.63	1624.34	27.40	114.66	169.58	17.05	7.56
Velocity	RMSE (m/s)	136.82	94.01	38.89	47.92	1400.51	39.86	28.89
	STD (m/s)	1.22	55.04	10.06	4.80	9.79	11.05	10.20
Run-time (s)		582.43	1129.52	922.48	0.049	0.64	1.85	1.75

As shown in Figure 15, when the flight vehicle performs lateral maneuvering, Ht-KalmanNet still maintains relatively small tracking errors. The time histories of the position tracking error and velocity tracking error for each filtering method are shown in Figure 16 and Figure 17, respectively.

A combined analysis of Figures 16 and 17 indicates that as the flight time increases and the lateral acceleration becomes larger, the position- and velocity-tracking errors of Ht-KalmanNet gradually increase. However, even at the end of the trajectories, corresponding to the final stage of the flight, the position- and velocity-tracking errors of Ht-KalmanNet remain consistently lower than those of the competing filtering methods, in-

cluding the UKF and KalmanNet, demonstrating its effectiveness in handling maneuvering conditions.

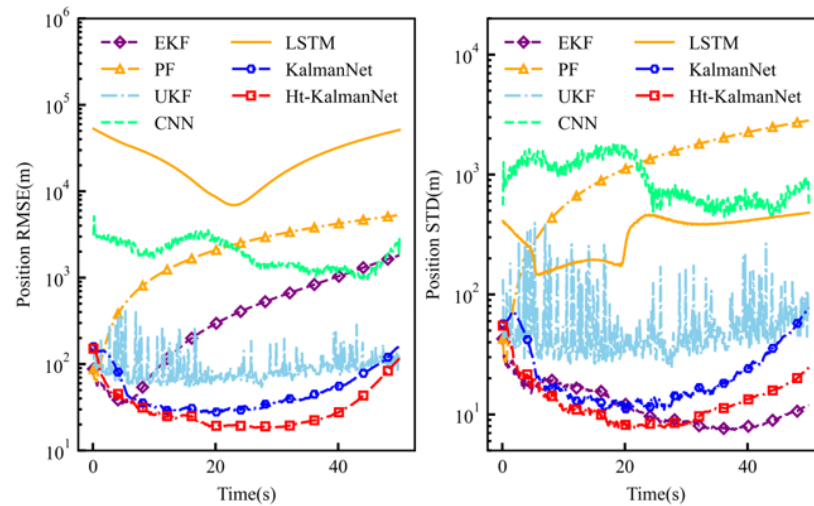


Figure 16. Position-tracking error curves under lateral maneuvering.

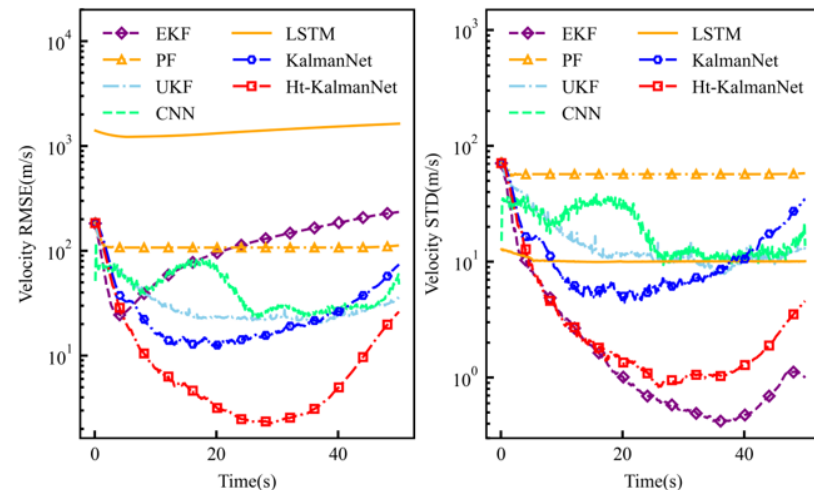


Figure 17. Velocity-tracking error curves under lateral maneuvering.

5.4. Bias in Range Measurement Standard Deviation

In practical scenarios, the actual range measurement STD may deviate from the value assumed in the filtering model. In this subsection, the tracking performance of different filtering methods is investigated under the condition in which the actual range measurement STD is larger than that used in the filtering model.

The test trajectory model used in this subsection is the same as that used for the original dataset, except that a bias is introduced in the observations. The actual STD of the range measurement σ_R' used to generate the observation data is increased to 200 m, whereas the STD σ_R assumed in the filtering model remains 100 m. Under this condition, the corresponding tracking results are shown in Figure 18 and Table 5.

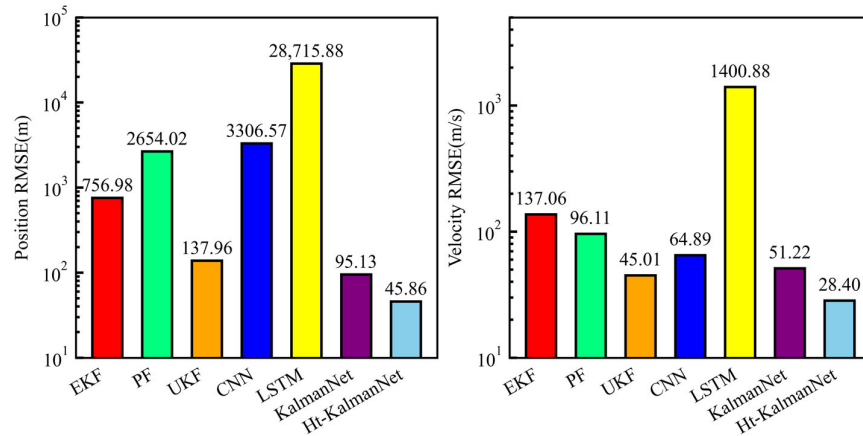


Figure 18. RMSE of position and velocity with bias in range measurement STD.

Table 5. Results with bias in range measurement STD.

		EKF	PF	UKF	CNN	LSTM	KalmanNet	Ht-KalmanNet
Position	RMSE (m)	756.98	2654.02	137.96	3306.57	28,715.88	95.13	45.86
	STD (m)	7.84	1385.44	25.96	132.36	180.80	25.35	9.98
Velocity	RMSE (m/s)	137.06	96.11	45.01	64.89	1400.88	51.22	28.40
	STD (m/s)	1.78	47.01	10.14	3.37	10.27	12.08	9.34
Run-time (s)		585.84	1129.73	930.69	0.063	0.68	1.91	1.83

As shown in Figures 11 and 18, when the standard deviation (STD) of the range measurement increases, the tracking errors of UKF, KalmanNet, and Ht-KalmanNet increase moderately. Nevertheless, Ht-KalmanNet still achieves the highest estimation accuracy among all considered filtering methods. The time histories of the position- and velocity-tracking errors for each method are presented in Figure 19 and Figure 20, respectively. Throughout the entire tracking process, Ht-KalmanNet maintains relatively low position- and velocity-tracking errors, demonstrating stable tracking performance.

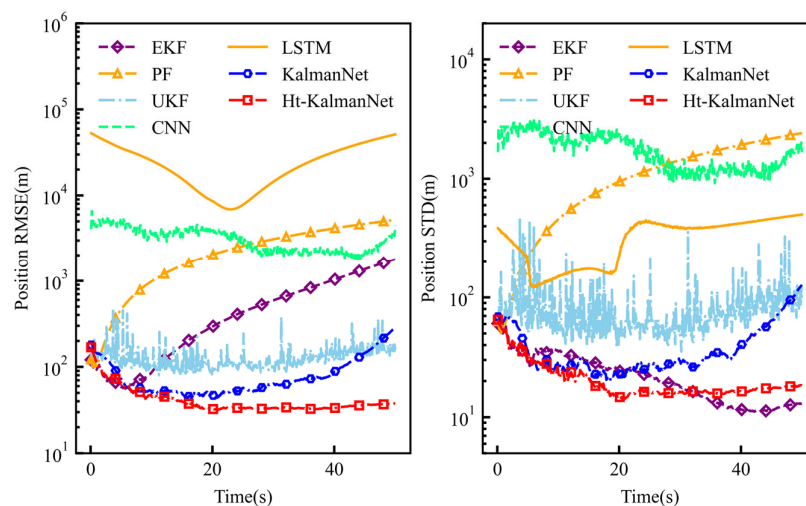


Figure 19. Position-tracking error curves with bias in range measurement STD.

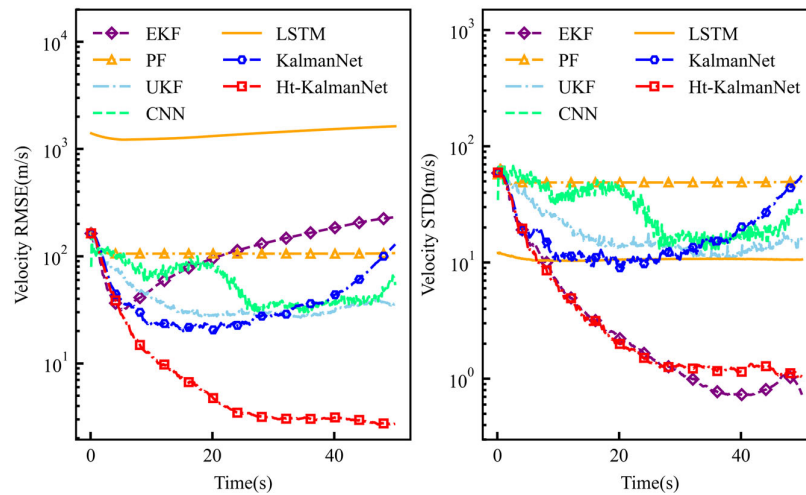


Figure 20. Velocity-tracking error curves with bias in range measurement STD.

5.5. Bias in Angle Measurement Standard Deviation

In practical situations, the actual STDs of angular measurements may deviate from the values assumed in the filtering model. In this subsection, the case in which the actual angle measurement STD is larger than that used in the filtering model is investigated, and the tracking results of different filtering methods under this condition are analyzed.

Similar to the scenario considered in Section 5.5, the test trajectories used in this subsection are generated using the nominal model, while additional observation noise bias is introduced into the angular measurements. In this subsection, the actual STD of the elevation angle measurement σ'_e and that of the azimuth angle measurement σ'_β used to generate the observation data are set to 2 mrad, while the angle measurement STDs assumed in the filtering model remain 1 mrad. The corresponding tracking results under this condition are shown in Figure 21 and Table 6.

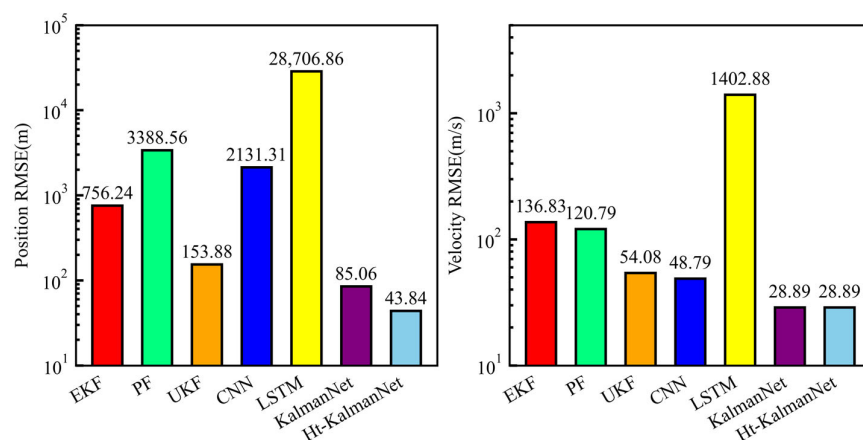


Figure 21. RMSE of position and velocity with bias in angle measurement STD.

Table 6. Simulation results with bias in angle measurement STD.

		EKF	PF	UKF	CNN	LSTM	KalmanNet	Ht-KalmanNet
Position	RMSE (m)	756.24	3388.56	153.88	2131.31	28,706.86	85.06	43.84
	STD (m)	6.95	1893.28	39.69	134.56	172.65	19.82	9.26

Velocity	RMSE (m/s)	136.83	120.79	54.08	48.79	1402.88	28.89	28.89
	STD (m/s)	1.19	64.49	12.84	6.18	9.85	10.40	9.48
	Run-time (s)	576.28	1127.05	916.27	0.052	0.63	1.83	1.81

As illustrated in Figures 12 and 21, model mismatch caused by bias in the angular measurement STD results in noticeable degradation in the estimation accuracy of both the UKF and KalmanNet. The time histories of the position- and velocity-tracking errors for different filtering methods are shown in Figure 22 and Figure 23, respectively.

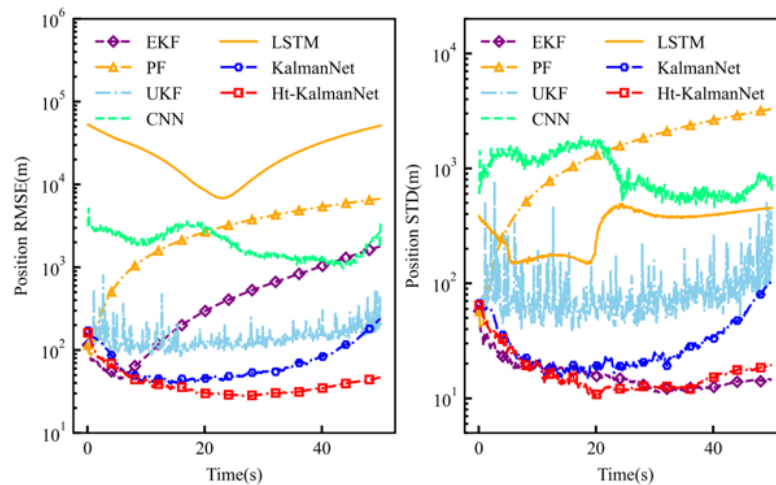


Figure 22. Position-tracking error curves with bias in angle measurement STD.

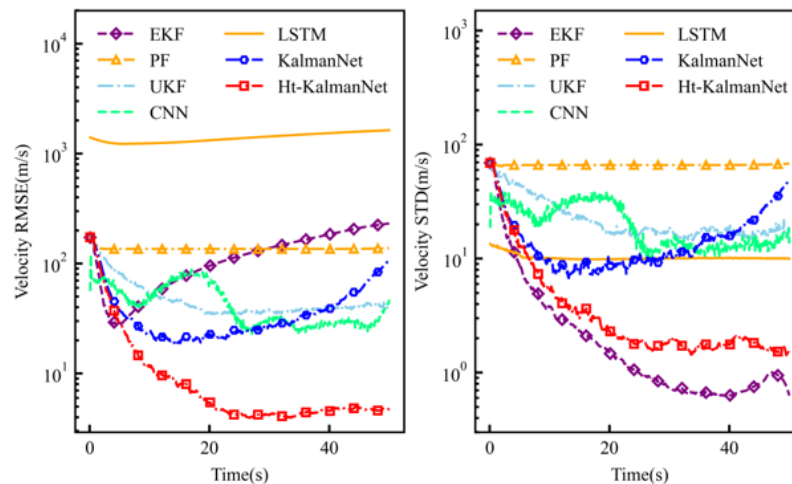


Figure 23. Velocity-tracking error curves with bias in angle measurement STD.

Although the position- and velocity-tracking errors of Ht-KalmanNet also increase under biased angular measurement conditions, the overall estimation errors remain consistently low throughout the entire tracking process, demonstrating strong robustness to observation-model mismatch and measurement uncertainty.

5.6. Discussion

In this section, under nominal simulation conditions, three conventional tracking methods, namely the EKF, PF, and UKF, are implemented, while four neural-network-based tracking models, including CNN, LSTM, KalmanNet, and the proposed Ht-Kal-

manNet, are trained using the nominal trajectory dataset. The estimation accuracy and computational cost of these seven filtering methods are systematically evaluated. Furthermore, the tracking performance of the considered methods is investigated under three model-mismatch scenarios, including perturbations in atmospheric density and aerodynamic coefficients, lateral maneuvers induced by nonzero sideslip angles, range-measurement noise deviation, and angular-measurement noise deviation.

The simulation results demonstrate that, among the three conventional filtering methods, the UKF achieves the highest estimation accuracy, owing to its superior capability in handling nonlinear transformations. The CNN and LSTM exhibit relatively large tracking errors because they rely primarily on data-driven learning without explicitly incorporating physical model information. In contrast, KalmanNet achieves higher tracking accuracy than the UKF while maintaining low computational cost through the integration of recursive filtering and neural-network-based gain estimation. Among all seven methods, the proposed Ht-KalmanNet achieves the highest tracking accuracy together with high computational efficiency.

Under different model-mismatch conditions, perturbations in atmospheric density and aerodynamic parameters exert relatively limited influence on the tracking performance because the nominal model considered in this study corresponds to a non-maneuvering re-entry vehicle. In contrast, lateral maneuvers and deviations in range and angular measurement noise significantly degrade the tracking performance of all methods. Nevertheless, Ht-KalmanNet consistently demonstrates superior estimation accuracy and robustness under all testing scenarios, thereby highlighting its effectiveness in nonlinear trajectory tracking under time-varying observation conditions.

6. Conclusions

The proposed Ht-KalmanNet demonstrates significant effectiveness in trajectory tracking for reentry vehicles. The framework is developed based on the algorithmic structure of the extended Kalman filter (EKF), in which the recurrent neural network architecture employed in KalmanNet is further improved to enable adaptive estimation of the time-varying observation matrix, thereby leading to the proposed Ht-KalmanNet framework.

The simulation results indicate that conventional filtering methods, including the EKF, PF, and UKF, generally require relatively high computational costs, among which the UKF achieves the highest estimation accuracy because of its superior capability in handling nonlinear transformations. In contrast, neural-network-based approaches such as CNN and LSTM provide fast computational speed but relatively limited tracking accuracy due to the lack of explicit physical-model information.

KalmanNet achieves estimation accuracy comparable to that of the UKF while maintaining significantly higher computational efficiency through the integration of recursive filtering and a neural network. The proposed Ht-KalmanNet not only preserves nearly the same computational efficiency as KalmanNet, but also further improves the tracking accuracy through adaptive estimation of the nonlinear time-varying observation matrix.

Furthermore, Monte Carlo simulations under three model-mismatch conditions demonstrate that Ht-KalmanNet maintains high estimation accuracy, thereby exhibiting strong robustness and generalization capability under uncertain observation conditions. These results demonstrate the effectiveness of the proposed framework for trajectory tracking during the reentry phase of flight vehicles.

The models considered in this study mainly focus on non-maneuvering reentry vehicles. Future research may incorporate more complex maneuvering vehicle models to further improve estimation accuracy and computational efficiency under highly nonlinear flight conditions. In addition, further attention can be devoted to exploring deeper inte-

gration between artificial intelligence techniques and traditional nonlinear filtering algorithms for aerospace tracking and guidance applications.

Author Contributions: Conceptualization, X.L. and W.C.; methodology, W.C.; software, X.L.; validation, W.C.; formal analysis, X.L. and W.L.; investigation, X.L. and W.C.; resources, W.C.; data curation, W.C.; writing—original draft preparation, X.L. and W.L.; writing—review and editing, X.L. and Z.W.; visualization, X.L. and W.L.; supervision, W.C.; project administration, W.C. and W.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All data used during the study appear in the submitted article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

EKF	Extended Kalman filter
PF	Particle filter
UKF	Unscented Kalman filter
CKF	Cubature Kalman filter
CNN	Convolutional neural network
RNN	Recurrent neural network
LSTM	Long Short-Term Memory
GRU	Gated recurrent unit
KG	Kalman gain
RMSE	Root mean square error
STD	Standard deviation
MSE	Mean squared error
SGD	Stochastic Gradient Descent
BPTT	Backpropagation through time
CV	Constant-velocity
CA	Constant-acceleration
MEKF	Multiplicative extended Kalman filter
MMSE	Minimum mean square error
UT	Unscented transform

References

1. Kalman, R.E. A new approach to linear filtering and prediction problems. *J. Basic Eng.* **1960**, *82D*, 35–45. <https://doi.org/10.1115/1.3662552>.
2. Luo, R.; Yuan, J.; He, Y.; Bao, S.; Du, L.; Hu, Z. Sensorless payload estimation of serial robots using an improved disturbance Kalman filter with a variable-parameter noise model. *Actuators* **2025**, *14*, 568. <https://doi.org/10.3390/act14120568>.
3. Wu, Z.; Zhu, J.; Xu, J.; Sun, X.; Han, Y. High-speed sensorless control strategy for dual three-phase linear induction motors based on nonlinear Kalman filter. *Actuators* **2026**, *15*, 78. <https://doi.org/10.3390/act15020078>.
4. Larson, R.E.; Dressler, R.M.; Ratner, R.S. *Application of the Extended Kalman Filter to Ballistic Trajectory Estimation*; Technical Report; Stanford Research Institute: Monlo Park, CA, USA, 1967.
5. Gordon, N.J.; Salmond, D.J.; Smith, A.F.M. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proc. F Radar Signal Process.* **1993**, *140*, 107–113. <https://doi.org/10.1049/ip-f-2.1993.0013>.
6. Liu, J.S.; Chen, R. Sequential Monte Carlo methods for dynamic systems. *J. Am. Stat. Assoc.* **1998**, *93*, 1032–1044. <https://doi.org/10.1080/01621459.1998.10473765>.
7. Julier, S.J.; Uhlmann, J.K. A new extension of the Kalman filter to nonlinear systems. In Proceedings of the SPIE Signal Processing, Sensor Fusion, and Target Recognition VI, Orlando, FL, USA, 21–25 April 1997; pp. 182–193. <https://doi.org/10.1117/12.280797>.

8. Van der Merwe, R.; Wan, E.A. Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models. Ph.D. Thesis, Oregon Health & Science University, Portland, OR, USA, 2004.
9. Arasaratnam, I.; Haykin, S. Cubature Kalman filters. *IEEE Trans. Autom. Control* **2009**, *54*, 1254–1269. <https://doi.org/10.1109/TAC.2009.2019800>.
10. Arasaratnam, I.; Haykin, S.; Hurd, T.R. Cubature Kalman filtering for continuous-discrete systems: Theory and simulations. *IEEE Trans. Signal Process.* **2010**, *58*, 4977–4993. <https://doi.org/10.1109/tsp.2010.2056923>.
11. Bengio, Y. Learning deep architectures for AI. *Found. Trends Mach. Learn.* **2009**, *2*, 1–127. <https://doi.org/10.1561/2200000006>.
12. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. <https://doi.org/10.1038/nature14539>.
13. Xu, J.; Guo, C.; Wang, Y.; Xiao, Y.; Hu, X. Fault-tolerant controller design for reusable launch vehicle. *Actuators* **2025**, *14*, 565. <https://doi.org/10.3390/act14110565>.
14. Chen, S.; Zhao, X.; Jin, X.; Wang, H. Adaptive fixed-time tracking control of cart–pendulum robotic systems with bias actuator dynamics. *Actuators* **2025**, *14*, 245. <https://doi.org/10.3390/act14050245>.
15. Lu, Y.; Sheng, B.; Li, Y.; Fu, G.; Jiang, S.; Jiang, Z. Adaptive prediction of compressor cylinder pressure dynamics using a physics-guided VAE-CNN state space model. *Actuators* **2025**, *14*, 535. <https://doi.org/10.3390/act14110535>.
16. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>.
17. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. In Proceedings of the NIPS 2014 Workshop on Deep Learning and Representation Learning, Montreal, QC, Canada, 8–13 December 2014.
18. Gao, X.; Ji, Y.; Zhang, S.; Van, K.; Li, H. Long short-term memory-based deep recurrent neural networks for target tracking. *Inf. Sci.* **2019**, *502*, 279–296. <https://doi.org/10.1016/j.ins.2019.06.039>.
19. Milan, A.; Rezaeifighi, S.H.; Dick, A.; Reid, I.; Schindler, K. Online multi-target tracking using recurrent neural networks. *Proc. AAAI Conf. Artif. Intell.* **2017**, *31*, 4225–4232. <https://doi.org/10.1609/aaai.v31i1.11194>.
20. Ondruska, P.; Posner, I. Deep tracking: Seeing beyond seeing using recurrent neural networks. *Proc. AAAI Conf. Artif. Intell.* **2016**, *30*, 3361–3367. <https://doi.org/10.1609/aaai.v30i1.10413>.
21. Yang, T.; Chan, A.B. Recurrent filter learning for visual tracking. In Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW), Venice, Italy, 22–29 October 2017; pp. 2010–2019. <https://doi.org/10.1109/iccvw.2017.235>.
22. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. <https://doi.org/10.1109/5.726791>.
23. LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* **1989**, *1*, 541–551. <https://doi.org/10.1162/neco.1989.1.4.541>.
24. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. <https://doi.org/10.1145/3065386>.
25. Danelljan, M.; Bhat, G.; Khan, F.S.; Felsberg, M. ECO: Efficient convolution operators for tracking. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 6931–6939. <https://doi.org/10.1109/cvpr.2017.733>.
26. Ma, C.; Huang, J.B.; Yang, X.; Yang, M.H. Hierarchical convolutional features for visual tracking. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 3074–3082. <https://doi.org/10.1109/iccv.2015.352>.
27. Wang, L.; Ouyang, W.; Wang, X.; Lu, H. Visual tracking with fully convolutional networks. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 3119–3127. <https://doi.org/10.1109/iccv.2015.357>.
28. Farina, A.; Ristic, B.; Benvenuti, D. Tracking a ballistic target: Comparison of several nonlinear filters. *IEEE Trans. Aerosp. Electron. Syst.* **2002**, *38*, 854–867. <https://doi.org/10.1109/taes.2002.1039404>.
29. Zarchan, P. *Tactical and Strategic Missile Guidance*, 6th ed.; American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2012.
30. Kumar, G.; Prasad, D.; Singh, R.P. Adaptive extended Kalman filter for ballistic missile tracking. *Int. J. Electr. Comput. Eng.* **2017**, *7*, 475–480. <https://doi.org/10.5281/zenodo.1131203>.
31. Zhao, Z.; Chen, H.; Chen, G.; Kwan, C.; Li, X.R. Comparison of several ballistic target tracking filters. In Proceedings of the American Control Conference, Minneapolis, MN, USA, 14–16 June 2006; pp. 2905–2910. <https://doi.org/10.1109/ACC.2006.1656545>.

32. Jin, Z. Global asymptotic stability analysis for autonomous optimization. *IEEE Trans. Autom. Control* **2025**, *70*, 6953–6960. <https://doi.org/10.1109/TAC.2025.3567509>.
33. Jin, Z.; Li, H.; Qin, Z.; Wang, Z. Gradient-free cooperative source-seeking of quadrotor under disturbances and communication constraints. *IEEE Trans. Ind. Electron.* **2025**, *72*, 1969–1979. <https://doi.org/10.1109/TIE.2024.3423337>.
34. Chen, Z.; Gui, H.; Zhong, R. Neural network-based navigation filter for monocular pose and motion tracking of noncooperative spacecraft. *Adv. Space Res.* **2025**, *75*, 2908–2928. <https://doi.org/10.1016/j.asr.2024.11.006>.
35. Revach, G.; Shlezinger, N.; van Sloun, R.J.G.; Eldar, Y.C. KalmanNet: Data-driven Kalman filtering. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 6–11 June 2021; pp. 3905–3909. <https://doi.org/10.1109/ICASSP39728.2021.9413750>.
36. Revach, G.; Shlezinger, N.; Ni, X.; Escoriza, A.L.; van Sloun, R.J.G.; Eldar, Y.C. KalmanNet: Neural network aided Kalman filtering for partially known dynamics. *IEEE Trans. Signal Process.* **2022**, *70*, 1532–1547. <https://doi.org/10.1109/tsp.2022.3158588>.
37. Choi, G.; Park, J.; Shlezinger, N.; Eldar, Y.C.; Lee, N. Split-KalmanNet: A robust model-based deep learning approach for SLAM. *IEEE Trans. Veh. Technol.* **2023**, *72*, 12326–12331. <https://doi.org/10.1109/TVT.2023.3270353>.
38. Maurice, A. Aerodynamic Performance Predictions of a SA-2 Missile Using Missile DATCOM. Master's Thesis, Naval Post-graduate School, Monterey, CA, USA, 2009.
39. Li, X.R.; Jilkov, V.P. Survey of maneuvering target tracking. Part II: Motion models of ballistic and space targets. *IEEE Trans. Aerosp. Electron. Syst.* **2010**, *46*, 96–119. <https://doi.org/10.1109/TAES.2010.5417150>.
40. Doucet, A.; de Freitas, N.; Gordon, N. (Eds.) *Sequential Monte Carlo Methods in Practice*; Springer: New York, NY, USA, 2001.
41. Dey, R.; Salem, F.M. Gate-variants of gated recurrent unit (GRU) neural networks. In Proceedings of the IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, USA, 6–9 August 2017; pp. 1597–1600. <https://doi.org/10.1109/MWSCAS.2017.8053243>.
42. Werbos, P.J. Backpropagation through time: What it does and how to do it. *Proc. IEEE* **1990**, *78*, 1550–1560. <https://doi.org/10.1109/5.58337>.
43. Sutskever, I. Training Recurrent Neural Networks. Ph.D. Thesis, University of Toronto, Toronto, ON, Canada, 2013.
44. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980. <https://doi.org/10.48550/arXiv.1412.6980>.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.