Information Fusion 53 (2020) 289–304

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full Length Article

DeepMTT: A deep learning maneuvering target-tracking algorithm based on bidirectional LSTM network

Jingxian Liu[a,b], Zulin Wang[a,1], Mai Xu[a,c,2,*]

[a] School of Electronic and Information Engineering, Beihang University, Beijing, China
[b] School of Computer Science and Communication Engineering, Guangxi University of Science and Technology, Liuzhou, China
[c] Hangzhou Innovation Institute (HZII), Beihang University, Hangzhou, China

ARTICLE INFO

Keywords:
Maneuvering target-tracking
Bidirectional long short-term memory network
Multiple models
Trajectory database

ABSTRACT

In the field of radar data processing, traditional maneuvering target-tracking algorithms assume that target movements can be modeled by pre-defined multiple mathematical models. However, the changeable and uncertain maneuvering movements cannot be timely and precisely modeled because it is difficult to obtain sufficient information to pre-define multiple models before tracking. To solve this problem, we propose a deep learning maneuvering target-tracking (DeepMTT) algorithm based on a DeepMTT network, which can quickly track maneuvering targets once it has been well trained by abundant off-line trajectory data from existent maneuvering targets. To this end, we first build a LArge-Scale Trajectory (LAST) database to offer abundant off-line trajectory data for network training. Second, the DeepMTT algorithm is developed to track the maneuvering targets using a DeepMTT network, which consists of three bidirectional long short-term memory layers, a filtering layer, a maxout layer and a linear output layer. The simulation results verify that our DeepMTT algorithm outperforms other state-of-the-art maneuvering target-tracking algorithms.

1. Introduction

Target tracking is a critical issue in the fields of airspace surveillance and air traffic control. A reliable and stable target-tracking algorithm can provide accurate estimations of the target states, thus guaranteeing better and safer airspace managements. In the past several decades, many filtering algorithms such as Kalman filter (KF) [1], extended KF (EKF) [2], unscented KF (UKF) [3] and particle filter (PF) [4] have been proposed to improve the tracking accuracy based on Bayesian tracking theory. In Bayesian tracking theory, pre-defined mathematical tracking models are required to simulate the movements of the target. Hence, the target-tracking performance heavily relies on these pre-defined mathematical models [5,6]. However, in practice, it is intractable to obtain precise mathematical models in advance to track maneuvering targets because the target movement is always uncertain and changing in maneuvering scenarios. This problem causes a serious degradation in the maneuvering-target tracking algorithms with pre-defined mathematical models. Hence, how to improve the performance of maneuvering-target tracking remains an important and challenging issue [7,8].

To solve the maneuvering-target tracking problem, many multiple-model (MM) algorithms [9,10] have been proposed, where more than one model is used with different filters to simultaneously track the target. Thus, the tracking results can be improved by combing all of the tracking results of different models with proper weights. Specifically, on the one hand, Blom and Bar-Shalom [11] proposed the interactive MM (IMM) algorithm to derive the adaptive weights of models according to the changeable observations. Many modified IMM algorithms [12–14] were proposed for specific scenarios of maneuvering-target tracking, such as nonlinear, non-Gaussian and multi-target tracking scenarios. On the other hand, model-set design algorithms were proposed to offer better model approximations in maneuvering-target tracking procedures, such as fixed structure MM (FSMM) [15,16] and variable structure MM (VSMM) [9,17] algorithms. Recently, advanced MM algorithms such as the MIE-BLUE-IMM [18] and hybrid grid MM (HGMM) [19] algorithms have been proposed, where more information, i.e., input estimation [18] and adaptive fine sub-models [19], is offered in the tracking processing. Thus, the maneuvering models can be more precisely estimated, and the tracking performance is further improved.

In practice, the movement models are unknown and changing in maneuvering-target tracking scenarios. The traditional MM algorithms must accumulate sufficient observation data to form proper estimations of the movement models, particularly when there are heavy observation

* Corresponding author: School of Electrical and Information Engineering, Beihang University, Beijing, China
E-mail addresses: ljx43031@buaa.edu.cn (J. Liu), wzulin@buaa.edu.cn (Z. Wang), maixu@buaa.edu.cn (M. Xu).
[1] Member, IEEE.
[2] Senior Member, IEEE.

https://doi.org/10.1016/j.inffus.2019.06.012
Received 3 September 2018; Received in revised form 23 May 2019; Accepted 2 June 2019
Available online 5 June 2019

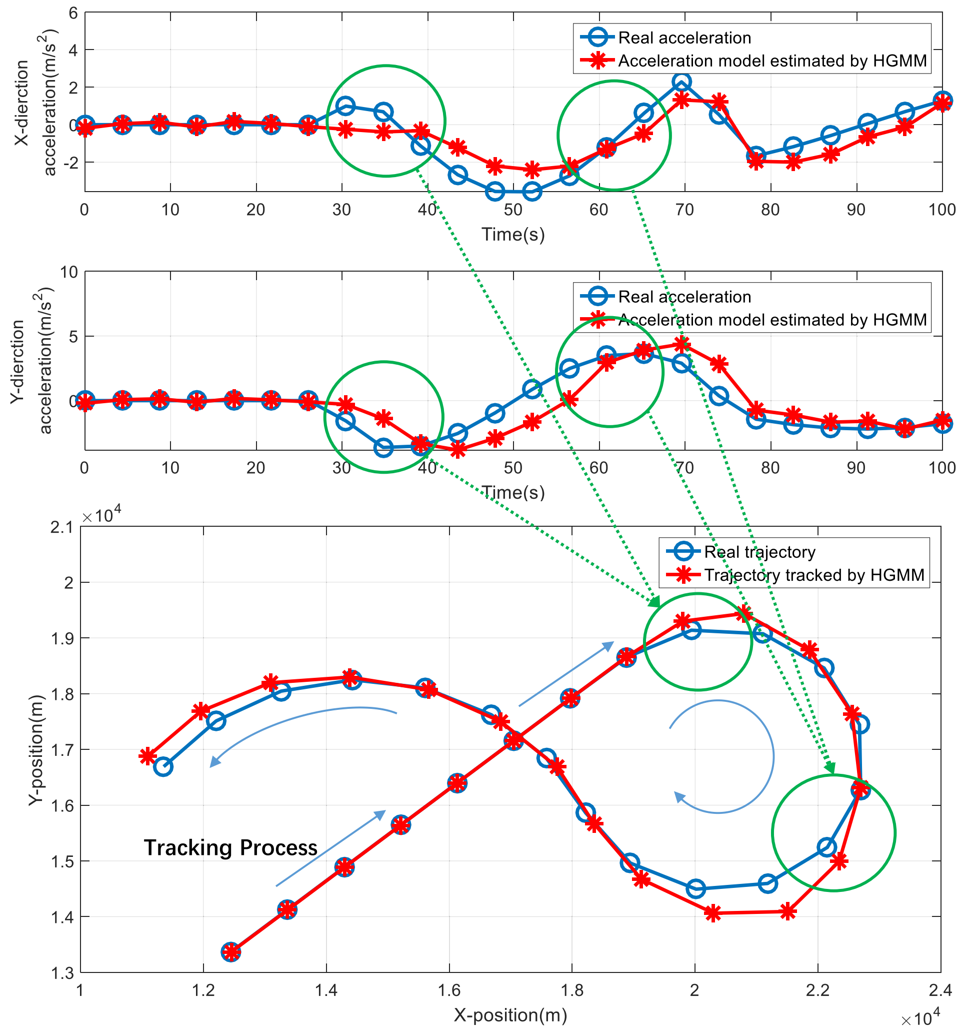


Fig. 1. Model estimation and target-tracking results with HGMM algorithm [19]. The upper two subfigures show the estimations of acceleration model with HGMM algorithm. Green circles indicate that those estimations are inaccurate and delayed in comparison with real accelerations especially when they change. Once the model-estimation-delay happens, the tracking performance degrades, as shown in the bottom subfigure.(For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

noise and nonlinear approximation errors. Thus, in traditional MM algorithms, the proper model estimated by previous observations always lags behind the current target state. This model-estimation-delay issue, mentioned and illustrated in [20,21], causes an obvious degradation of the tracking performance. Specifically, in a two-dimensional maneuvering tracking scenario as shown in Fig. 1, the state-of-the-art MM algorithm, i.e., HGMM, is used to track the maneuvering target. However, Fig. 1 shows that during the entire tracking procedure, the estimation of the acceleration model is delayed for 5 s. As a result, the traditional MM algorithms cannot always accurately track the maneuvering targets.

To solve the model-estimation-delay problem, from the data-driven perspective, we propose a new deep learning maneuvering target-tracking (DeepMTT) algorithm. In our DeepMTT algorithm, we design a “smart” model, which can learn to timely and precisely predict trajectories of maneuvering targets with different observations. A simple method to build this model is based on neural networks [22]. In fact, neural networks have been applied to the field of target tracking since the 1990s [22–24]. However, these networks are too shallow to approximate the complex characteristics of maneuvering trajectories because of the limited parameters in the networks. Thus, they are only complementary methods to provide more information in the tracking process [25–27] and cannot solve the time-delay issue in maneuvering-target tracking.

In contrast to the shallow neural networks of [25–27], our DeepMTT algorithm learns the maneuvering models from observations based on the bidirectional long short-term memory (LSTM) [28–30] structure, which is a deep learning structure. This DeepMTT network can directly estimate the trajectories of the maneuvering targets with no movement model approximation procedure in the classic MM algorithms. Thus, the model-estimation-delay problem can be eliminated to a great extent. To this end, we first build a LArge-Scale Trajectory (LAST) database, which contains 10 million trajectories with the corresponding observations that cover normal maneuvering cases in civil airport surveillance. Using the LAST database, we can design and train a deeper network with a great amount of trainable parameters, which well fit different models of maneuvering movements. Accordingly, our DeepMTT network is built to directly learn the maneuvering trajectories with no movement models. The DeepMTT network contains a filtering layer, three bidirectional LSTM layers, a maxout layer and a linear output layer. Three bidirectional LSTM layers are the main components of the DeepMTT network to learn the temporal information in maneuvering trajectories, meanwhile the filtering layer is designed to decrease the noise. The maxout layer is “good at” fitting the nonlinear function [31], which is used to generate the trajectory data. In addition, the linear output layer reforms the outputs into the target states of trajectories. Finally, we can directly and timely track the maneuvering target with a well-trained DeepMTT network. The simulation results show that our DeepMTT algorithm outperforms other state-of-the-art maneuvering tracking algorithms. The meanings of notation in the paper are listed in Table 1. The main contributions of our work are summarized as follows:

- An LAST database is built to offer sufficient training data for tracking maneuvering targets.
- A DeepMTT network is proposed to track a maneuvering target by learning from the extensive trajectory data of the LAST database.

Table 1
Notation list.

Notation	Meaning of the notation
k	Discrete time step for tracking, $k = 1, 2, 3, \dots, K$.
$\boldsymbol{x}_k$	Target state vector at time step k.
$\boldsymbol{F}$	State transition matrix.
$\boldsymbol{n}$	Transition noise.
$\boldsymbol{z}_k$	Observation vector at time step k.
$\boldsymbol{h}(\cdot)$	Nonlinear observation function.
$\boldsymbol{m}$	Observation noise.
$d_{x,k}$, $d_{y,k}$	Distances along x and y directions at time step k.
$v_{x,\,k}$, $v_{y,\,k}$	Velocities along x and y directions at time step k.
n_d, n_v	Transition noises of distance and velocity.
s_τ	Sampling interval.
α	Turn rate of maneuvering target.
θ_k, r_k	Azimuth and distance observed at time step k.
m_θ, m_r	Observation noises of azimuth and distance.
t_{segment}	Length of trajectory segment.
$V_{\max}$	Maximum velocity of aircraft.
D_{random}	Distance uniformly sampled from a given range.
V_{random}	Velocity uniformly sampled from a given range.
θ_{distance}	Direction of distance sampled from $[-180°, 180°]$.
θ_{velocity}	Direction of velocity sampled from $[-180°, 180°]$.
σ_d, σ_v, σ_a, σ_θ, σ_r	Standard deviations of distance, velocity, acceleration, azimuth and distance, respectively.
$\hat{\boldsymbol{x}}_k$	State vector estimated by filtering algorithm at time step k.
$\hat{\boldsymbol{x}}_k^N$	Normalized state vector at time step k.
$C_{\max}$	Maximum value in sequence $\hat{\boldsymbol{x}}_{1:K}$.
$\boldsymbol{r}_{1:K}$	Residual sequence of the estimated trajectory.
$\tilde{\boldsymbol{x}}_{1:K}$	Modified estimated trajectory.
$\boldsymbol{A}$	Matrix of sliding window for filtering.
$\hat{\boldsymbol{x}}_k^F$	Output vector of filtering layer at time step k.
$\phi_n(\cdot)$	Noisy activation function.
$\phi_p(\cdot)$	Hard-tanh activation function.
$\sigma(\cdot)$	Scaling function.
$\boldsymbol{h}_{1:K}^L$	Output sequence of bidirectional LSTM layers.
$\boldsymbol{h}_{1:K}^M$	Output sequence of maxout layer.

2. Large-Scale trajectory database of maneuvering target

In this paper, the LAST database is established for maneuvering-target tracking. The samples in our database contain the observations and ground-truth of trajectories, which constitute the input-output pairs for the DeepMTT network. In fact, it is difficult to obtain sufficient ground-truth data of maneuvering-target trajectory in civil airport surveillance. Instead, based on the state space model (SSM) [7], we design a trajectory generator that can simulate the segments of different maneuvering target trajectories, which are viewed as samples in our LAST database. To guarantee that the trajectories in our LAST database can match the real tracking scenario, all the parameters used to generate the trajectories are set according to the real scenarios. Here, 10 million samples are generated by the trajectory generator to establish the LAST database, which cover all common cases of maneuvering targets in the air traffic control (ATC) system [7,12].

Specifically, the SSM of the trajectory generator is defined as follows:

$$\textbf{Transition equation}: \ \boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_{k-1} + \boldsymbol{n}, \tag{1a}$$

$$\textbf{Observation equation}: \ \boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{m}, \tag{1b}$$

where $\boldsymbol{x}_k$ and $\boldsymbol{z}_k$ are the target state and the corresponding observation at time step k, respectively. In transition equation (1a), $\boldsymbol{F}$ is the transition matrix, and $\boldsymbol{n}$ is the transition noise. In the observation equation of (1b), $h(\cdot)$ is the nonlinear observation, and $\boldsymbol{w}$ is the observation noise. According to the SSM, the ground-truth $\boldsymbol{x}_{1:K} \triangleq \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_K\}$ of trajectories is generated by (1a), and the observations $\boldsymbol{z}_{1:K} \triangleq \{\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_K\}$ are generated by (1b). To produce proper data for our LAST database, the SSM is analyzed in detail as follows.

In this paper, we only consider an X-Y plane coordinate of maneuvering-target tracking. Thus, $\boldsymbol{x}_k$ is defined as $[d_{x,\,k}, d_{y,\,k}, v_{x,\,k}, v_{y,\,k}]^{\mathrm{T}}$, where $[d_{x,k}, d_{y,k}]^{\mathrm{T}}$ is the two-dimensional (2-D) position, and $[v_{x,\,k}, v_{y,\,k}]^{\mathrm{T}}$ is the corresponding velocity. The transition matrix $\boldsymbol{F}$ of (1a) is defined in two shapes, constant velocity (CV) and constant turn (CT) shapes, to satisfy the requirement in generating maneuvering-target trajectories. According to [32], the CV shape is defined as

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & s_\tau & 0 \\ 0 & 1 & 0 & s_\tau \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2}$$

for the case of aircraft cruising. The CT shape is defined as

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & \frac{\sin(\alpha s_\tau)}{\alpha} & \frac{\cos(\alpha s_\tau)-1}{\alpha} \\ 0 & 1 & \frac{1-\cos(\alpha s_\tau)}{\alpha} & \frac{\sin(\alpha s_\tau)}{\alpha} \\ 0 & 0 & \cos(\alpha s_\tau) & -\sin(\alpha s_\tau) \\ 0 & 0 & \sin(\alpha s_\tau) & \cos(\alpha s_\tau) \end{bmatrix}, \tag{3}$$

for the case of aircraft maneuvering. In the above equations, s_τ is the sampling interval of trajectories, which is set to be 0.1 s in our LAST database; α is the turn rate of maneuvering target. For radar tracking, $\boldsymbol{z}_k = [\theta_k, \ \ r_k]^{\mathrm{T}}$ is the radar observation vector which contains azimuth value θ_k and distance value r_k. In addition, $h(\cdot)$ of (1b) is denoted as follows:

$$\underbrace{\begin{bmatrix} \theta_k \\ r_k \end{bmatrix}}_{\boldsymbol{z}_k} = \begin{bmatrix} \arctan \frac{d_{y,k}}{d_{x,k}} \\ \sqrt{d_{x,k}^2 + d_{y,k}^2} \end{bmatrix} + \underbrace{\begin{bmatrix} m_\theta \\ m_r \end{bmatrix}}_{\boldsymbol{m}}, \tag{4}$$

where $\boldsymbol{m} = [m_\theta, \ \ m_r]^{\mathrm{T}}$ is the noise of observation vector, which contains the azimuth and distance parts, i.e., m_θ and m_r.

Using the SSM, $\boldsymbol{z}_{1:K} \triangleq \{\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_K\}$ and $\boldsymbol{x}_{1:K} \triangleq \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_K\}$ can be generated, given the parameters of the SSM, i.e., the number of time steps K, initial state x_0, noises $\boldsymbol{n}$ and $\boldsymbol{m}$, and transition matrix $\boldsymbol{F}$.

Table 2
Ranges of Maneuvering Target Trajectories.

Content	Ranges
Distance from radar	$0.5 \sim 20$ (*nautical mile*)
Velocity of aircraft	$0 \sim 340$ (m/s)
Maneuvering turn rate	$-10 \sim 10$ $(°/s)$

Before generating the samples of our LAST database, we must determine the range of target trajectory segments to ensure that they cover all possible maneuvering cases.

As shown in Table 2, the distance from the radar to the target covers the main detection range of the common airport surveillance radar (ASR), i.e., 0.5 $\sim$ 20 nautical miles (NM) (about 926 m $\sim$ 37,040 m), according to [33]. Then, we set the velocity of our maneuvering target in the range of 0–340 m/s because civilian aircrafts rarely exceed the sound velocity [34], i.e., 340 m/s. The turn rate α, which decides the maneuvering cases, ranges from $-10°/s$ to $10°/s$ because it is normally less than $10°/s$ for the common civilian aircraft according to [7]. The smallest interval between turn rates is $0.1°/s$, which provides sufficient differentiation to track the maneuvering target.

Accordingly, we derive the parameters of the SSM as follows:

(1) In our LAST database, the length of trajectory segments t_{segmet} is defined to be 5 s because the fixed length of trajectory segments is required in our DeepMTT network. In practice, a long trajectory can be considered a combination of several trajectory segments.

(2) We calculate the initial state $\boldsymbol{x}_0$ using polar decomposition. Thus, all generated trajectory segments satisfy the main radar detection ranges. Specifically, to ensure that the entire trajectory is in the main radar detection ranges, the initial distance of our target is set to range from $926 + V_{\max} \times t_{\text{segmet}}$ to $37,040 - V_{\max} \times t_{\text{segmet}}$, where $V_{\max} = 340$ m/s is the maximum velocity of aircrafts as discussed above. Then, we uniformly sample a random distance D_{random} from this range to obtain the initial positions ($d_{x,0}$, $d_{y,0}$) of the trajectory in the X and Y directions as

$$d_{x,0} = D_{\text{random}} \cdot \cos(\theta_{\text{distance}}), \tag{5a}$$

$$d_{y,0} = D_{\text{random}} \cdot \sin(\theta_{\text{distance}}), \tag{5b}$$

where θ_{distance}, which is uniformly sampled from -180° to 180°, is the intersection angle between the north and the direction from the radar to the target. Likewise, the initial velocities ($v_{x,\,0}$, $v_{y,\,0}$) in the X and Y directions are calculated as

$$v_{x,0} = V_{\text{random}} \cdot \cos(\theta_{\text{velocity}}), \tag{6a}$$

$$v_{y,0} = V_{\text{random}} \cdot \sin(\theta_{\text{velocity}}), \tag{6b}$$

where V_{random} is the random velocity and uniformly sampled from 0 m/s to 340 m/s. In addition, θ_{velocity}, which is uniformly sampled from -180° to 180°, is the intersection angle between the north and the initial direction of the target velocity. Accordingly, we obtain the initial state of the trajectory segment as $\boldsymbol{x}_0 \triangleq [d_{x,\,0},\, d_{y,\,0},\, v_{x,\,0},\, v_{y,\,0}]^{\mathrm{T}}$.

(3) According to [35], the transition noise $\boldsymbol{n}$ and observation noise $\boldsymbol{m}$ are defined as follows:

$$\begin{aligned} &\boldsymbol{n} = [n_d, n_d, n_v, n_v]^{\mathrm{T}}, \\ &n_d \sim \mathcal{N}(n_d; 0, \sigma_d^2), n_v \sim \mathcal{N}(n_v; 0, \sigma_v^2), \end{aligned} \tag{7}$$

$$\begin{aligned} &\boldsymbol{m} = [m_\theta, m_r]^{\mathrm{T}}, \\ &m_\theta \sim \mathcal{N}(m_\theta; 0, \sigma_\theta^2), m_r \sim \mathcal{N}(m_r; 0, \sigma_r^2). \end{aligned} \tag{8}$$

In (7), $\sigma_d = 0.5\sigma_a s_\tau^2$ and $\sigma_v = \sigma_a s_\tau$ are the standard deviations of transition noise for distance and velocity, respectively; σ_a is the standard deviation of accelerated velocity noise that is randomly sampled in the range of $[8\,m/s^2, 13\,m/s^2]$ according to [7]. In (8), the deviations of azimuth noise σ_θ and distance noise σ_r are randomly sampled in the range of [0.401°, 0.516°] and $[8\,m, 13\,m]$ according to [19,36], respectively.

(4) The maneuvering cases depend on the value of turn rate α, which is randomly sampled from $-10°/s$ to $10°/s$ with the sampling interval of $0.1°/s$ as discussed. If $\alpha = 0$, we use the transition equation of (1a) with the transition matrix $\boldsymbol{F}$ in the CV shape of (2) to generate the trajectory. Otherwise, we use (1a) with $\boldsymbol{F}$ in the CT shape (3) to generate the trajectory.

After all settings have been obtained for the SSM, we generate 10 million samples, which are randomly and equally grouped into 100 thousand batches, i.e., each batch contains 100 samples. We randomly select 50,000 ground-truths of trajectory segments from our LAST database and visualize them in Fig. 2. As shown in this figure, the trajectory

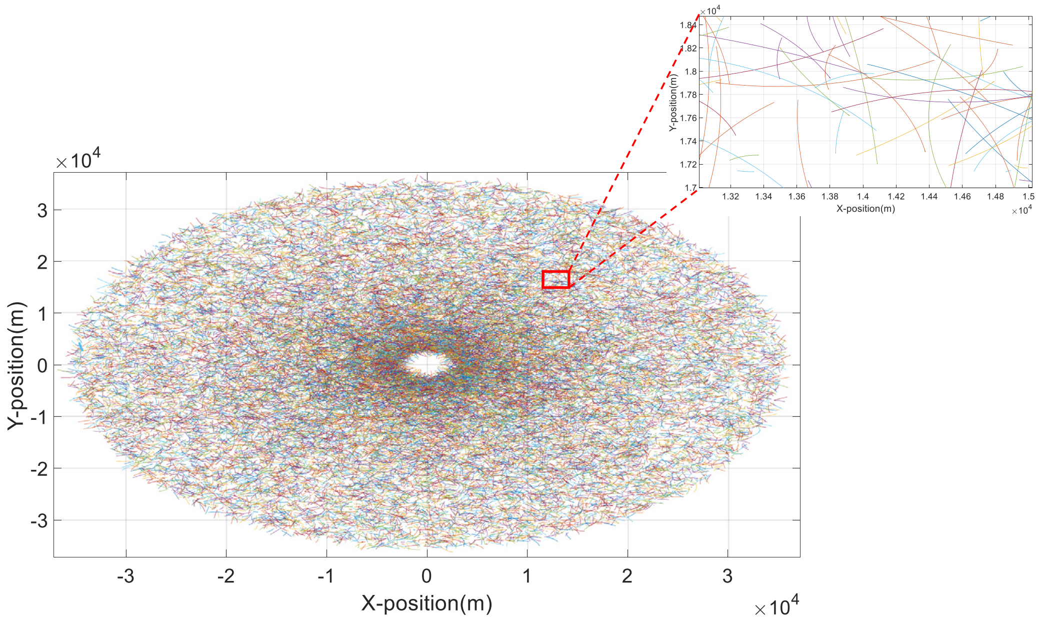


Fig. 2. The ground-truth of trajectories. All 50,000 ground-truths are depicted in the same figure. A small area marked by red rectangle is zoomed into a bigger figure, i.e., the upper right subfigure, to give more details of the trajectories. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

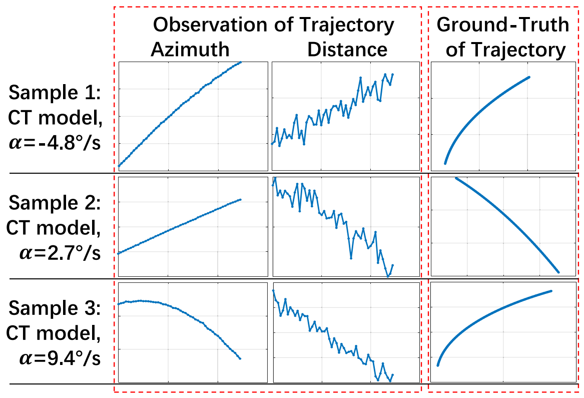


Fig. 3. Three samples in the LAST database. Each sample contains the ground-truth of a trajectory segment and the two corresponding observations, i.e., azimuth and distance, respectively.

segments in our LAST database cover the main detection area of ASR with different types of maneuvering cases. The true samples of our LAST database are shown in Fig. 3. Specifically, the first and second columns of Fig. 3 are the observations of trajectory segments, including the azimuth and distance data, respectively. The last column is the ground-truth of the trajectory segment. Note that the fluctuations in the azimuth and distance are caused by noises $\boldsymbol{n}$ and $\boldsymbol{m}$ in the SSM.

3. Deep Learning Maneuvering Target-Tracking Algorithm

In this section, we present the details of our DeepMTT algorithm[3] Unlike the traditional tracking algorithms, our DeepMTT algorithm tracks the target based on a DeepMTT network, which is trained off-line. The DeepMTT network can output residuals to directly correct the UKF tracking results, by which the maneuvering trajectories are timely and precisely estimated. The DeepMTT algorithm contains two stages: training and tracking stages, as shown in Fig. 4. In the training stage, the original observations and ground-truth are first pre-processed to generate the modified input-output pairs, which are suitable for training the DeepMTT network. The modified inputs are normalized trajectory segments tracked by UKF, and the modified outputs are residuals between the ground-truth and the estimated trajectory segments. Then, with those modified input-output pairs, the DeepMTT network is trained to predict the residuals using a loss function defined by the root mean square error (RMSE). In the tracking phase, the same pre-process step is used to estimate the trajectory segments. Those estimations of segments are corrected by the residuals predicted from the DeepMTT network. Finally, an entire maneuvering trajectory is estimated with the corrected segments using a reconstruction step. The details of our DeepMTT network are presented as follows.

3.1. Pre-processing step

There are two major problems when we directly use samples in our database to train the network: 1) the activations of the units in the first hidden layer fall into the 0-gradient region (which we call the saturated region in this paper), and 2) there is partial data feature loss.

Observation 1. If the input data for network unit are larger than 18, the activations in units, such as tanh and hard-tanh activations, have fallen into the saturated region and block the back-propagation of loss.

Analysis 1. In back-propagation, the loss of the current unit δ is calculated as follows:

$$\delta = \phi'(u) L_p, \tag{9}$$

[3] Open resource codes: https://github.com/ljx43031/DeepMTT-algorithm.

where $\phi'(u)$ is the derivative of activation function $\phi(u)$ in the current unit with input u. In addition, L_p is the sum of the loss of the previous layer. According to [30], the tanh activation function is defined as

$$\phi_{\tanh}(u) = \frac{e^{2u} - 1}{e^{2u} + 1}. \tag{10}$$

Hence, the derivative of tanh function is calculated as follows:

$$\phi'_{\tanh}(u) = 1 - \phi_{\tanh}(u)^2. \tag{11}$$

Moreover, according to [37], the hard-tanh activation function is defined as

$$\phi_{\text{hard-tanh}}(u) = \max(\min(u, 1), -1). \tag{12}$$

Thus, the derivative of hard-tanh function is calculated as follows:

$$\phi'_{\text{hard-tanh}}(u) = \begin{cases} 1, & -1 < u < 1 \\ 0, & \text{else} \end{cases} \tag{13}$$

Those derivatives are shown in Fig. 5. Obviously, when input $|u| > 18$, we have $|\phi'_{\tanh}(u)| < 8.8818e-16$ and $\phi'_{\text{hard-tanh}}(u) = 0$. Hence, the input $|u| > 18$ has pushed the activation of a unit towards the 0-gradient region [37], i.e., saturated region, which makes the loss vanish in this unit and hinders the loss from back-propagating.

This completes the analysis of Observation 1.

First, we notice that the distances of the input data are 926–37,040 m. According to observation 1, this large input value has pushed the activations of the units in the first hidden layer into the saturated region and hindered the back-propagation of loss. Second, the azimuths of input data are too small in comparison with the distances. When they are fed into the network and combined with distances based on a random weight matrix, their data features are "submerged" by the distance data. To solve these problems, a pre-processing step is developed to modify the input-output pairs of samples, which are suitable for training. Thus, the loss of our DeepMTT network can be guaranteed to converge to a satisfactory small value. The pre-processing step is summarized in Fig. 6. As shown in Fig. 6, on the one hand, we filter the azimuth and distance data by the UKF algorithm with the transition matrix in the CV shape (CV–UKF), by which the input data with observation space have been changed into the target state space. Thus, we can obtain the estimated trajectory segment: $\hat{\boldsymbol{x}}_{1:K} \triangleq \{\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_K\}$ containing the position and velocity data. Then, $\hat{\boldsymbol{x}}_{1:K}$ is normalized to generate the final input data $\hat{\boldsymbol{x}}^N_{1:K}$ for the DeepMTT network as follows:

$$\begin{aligned} \hat{\boldsymbol{x}}^N_{1:K} &\triangleq \{\hat{\boldsymbol{x}}^N_1, \hat{\boldsymbol{x}}^N_2, \ldots, \hat{\boldsymbol{x}}^N_K\} \\ &= \{\hat{\boldsymbol{x}}_1/C_{\max}, \hat{\boldsymbol{x}}_2/C_{\max}, \ldots, \hat{\boldsymbol{x}}_K/C_{\max}\}, \end{aligned}$$

where $C_{\max}$ is the maximum absolute value in the elements of $\hat{\boldsymbol{x}}_{1:K}$. Obviously, the range of normalized data in $\hat{\boldsymbol{x}}^N_{1:K}$ is [-1,1], which avoids the activation of a unit falling into the saturated region. Moreover, the difference between the velocity and position data is much smaller than that between the azimuth and distance data. Thus, the problem of partial data feature loss can be relieved to a great extent.

Meanwhile, the desired output of the DeepMTT network is the residual sequence $\boldsymbol{r}_{1:K}$, which is denoted as:

$$\boldsymbol{r}_{1:K} = \boldsymbol{x}_{1:K} - \hat{\boldsymbol{x}}_{1:K}, \tag{14}$$

where $\boldsymbol{x}_{1:K}$ is the ground-truth of the trajectory segment. Taking $\hat{\boldsymbol{x}}_{1:K}$ as a reference of the ground-truth, $\boldsymbol{r}_{1:K}$ is the relative error between the ground-truth and its reference. Obviously, $\boldsymbol{r}_{1:K}$ mainly contains the information of diversity among different trajectories causing by different movement models without the information of absolute data in the trajectory such as positions and velocities. As a result, $\boldsymbol{r}_{1:K}$ has less information that must be learned than the ground-truth $\boldsymbol{x}_{1:K}$. Hence, it is easier to learn by our DeepMTT network.

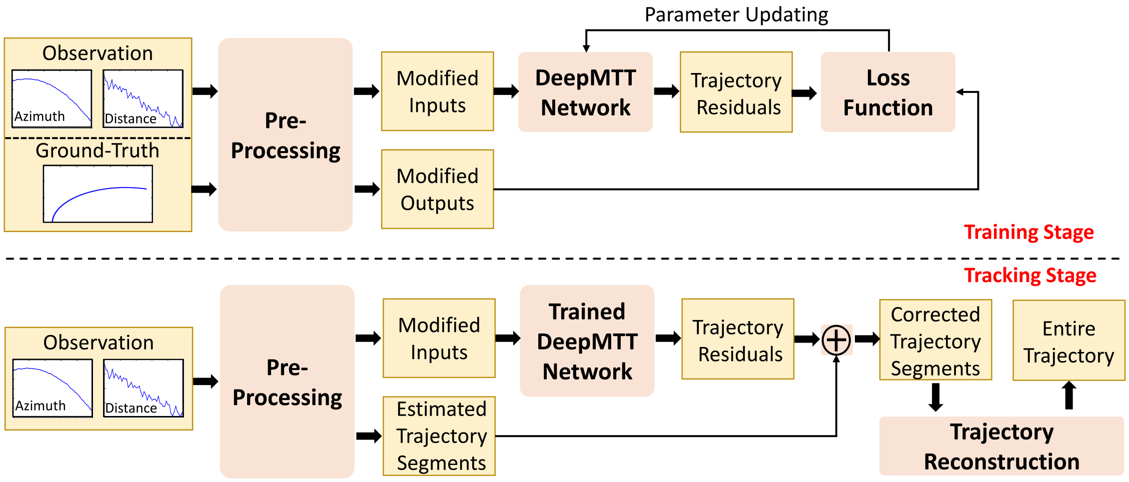


Fig. 4. The framework of DeepMTT algorithm. The upper part shows the training stage, in which the pre-processing data is used to train the DeepMTT network according to a designed loss function. The bottom part shows the tracking stage, in which the trained DeepMTT network is used to generate the estimations of trajectory segments. Then, an entire trajectory is reconstructed by those segments.

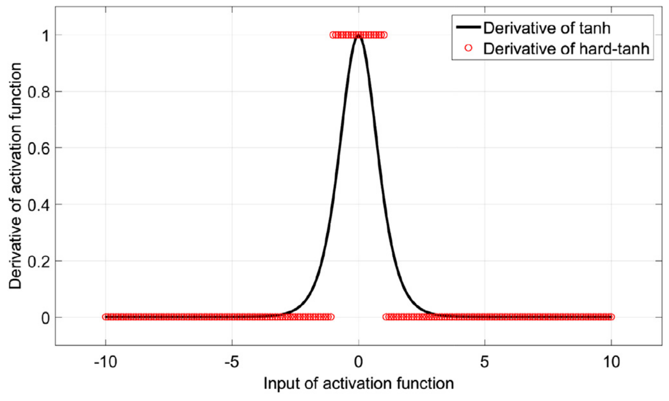


Fig. 5. Derivatives of tanh and hard-tanh activation functions.

3.2. *The DeepMTT network*

The DeepMTT network is used to learn the information of input sequence $\hat{\boldsymbol{x}}^N_{1:K}$, to output the estimations of $\boldsymbol{r}_{1:K}$. Once the network is well trained, it can directly output those estimations within an acceptable error range, without any information of the ground-truth. The DeepMTT network is developed with three bidirectional LSTM layers, a filtering layer, a maxout layer, and a linear output layer. By feeding the input data $\hat{\boldsymbol{x}}^N_{1:K}$ to our DeepMTT network, the regression prediction of $\boldsymbol{r}_{1:K}$ is yielded. Then, the corrected trajectory segment can be estimated as follows:

$$\tilde{\boldsymbol{x}}_{1:K} = \tilde{\boldsymbol{r}}_{1:K} + \hat{\boldsymbol{x}}_{1:K}, \tag{15}$$

where $\tilde{\boldsymbol{r}}_{1:K}$ is the regression prediction of $\boldsymbol{r}_{1:K}$ output by the DeepMTT network. The structure of our DeepMTT network is summarized in Fig. 7.

Feed-forward data flow:

The details of the feed-forward data flow are shown in Fig. 7. First, we filter the input data sequence $\hat{\boldsymbol{x}}^N_{1:K}$ with a five-point sliding window to reduce the noise of the input data. This window is set to be a 5×4 matrix as follows:

$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\ A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} \\ A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} \\ A_{5,1} & A_{5,2} & A_{5,3} & A_{5,4}, \end{bmatrix} \tag{16}$$

where $A_{i,j}$ with $i \in [1:5]$ and $j \in [1:4]$ is the learnable parameter of our sliding window. As we know, $\boldsymbol{x}^N_k$ includes 4 elements: the elements of distance in the X and Y directions $\hat{d}^N_{x,k}$ and $\hat{d}^N_{y,k}$ and elements of velocity in the X and Y directions $\hat{v}^N_{x,k}$ and $\hat{v}^N_{y,k}$. Each element in $\boldsymbol{x}^N_k$ is filtered by each independent column in $\boldsymbol{A}$ with the time steps as follows:

$$\hat{d}^F_{x,k} = A_{1,1}\hat{d}^N_{x,k-4} + A_{2,1}\hat{d}^N_{x,k-3} \dots + A_{5,1}\hat{d}^N_{x,k}, \tag{17}$$

$$\hat{d}^F_{y,k} = A_{1,2}\hat{d}^N_{y,k-4} + A_{2,2}\hat{d}^N_{y,k-3} \dots + A_{5,2}\hat{d}^N_{y,k}, \tag{18}$$

$$\hat{v}^F_{x,k} = A_{1,3}\hat{v}^N_{x,k-4} + A_{2,3}\hat{v}^N_{x,k-3} \dots + A_{5,3}\hat{v}^N_{x,k}, \tag{19}$$

$$\hat{v}^F_{y,k} = A_{1,4}\hat{v}^N_{y,k-4} + A_{2,4}\hat{v}^N_{y,k-3} \dots + A_{5,4}\hat{v}^N_{y,k}. \tag{20}$$

Note that $\hat{\boldsymbol{x}}^N_{-3}$, $\hat{\boldsymbol{x}}^N_{-2}$, $\hat{\boldsymbol{x}}^N_{-1}$ and $\hat{\boldsymbol{x}}^N_{0}$ are set to be zero vectors $\boldsymbol{O}$ because they do not exist. Furthermore, the symbol $\hat{\boldsymbol{x}}^F_k$ in Fig. 7 is defined as $\hat{\boldsymbol{x}}^F_k \triangleq [\hat{d}^F_{x,k}, \hat{d}^F_{y,k}, \hat{v}^F_{x,k}, \hat{v}^F_{y,k}]^{\mathrm{T}}$.

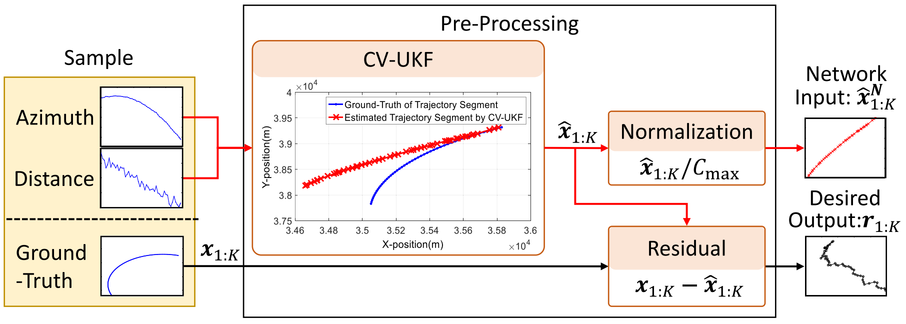
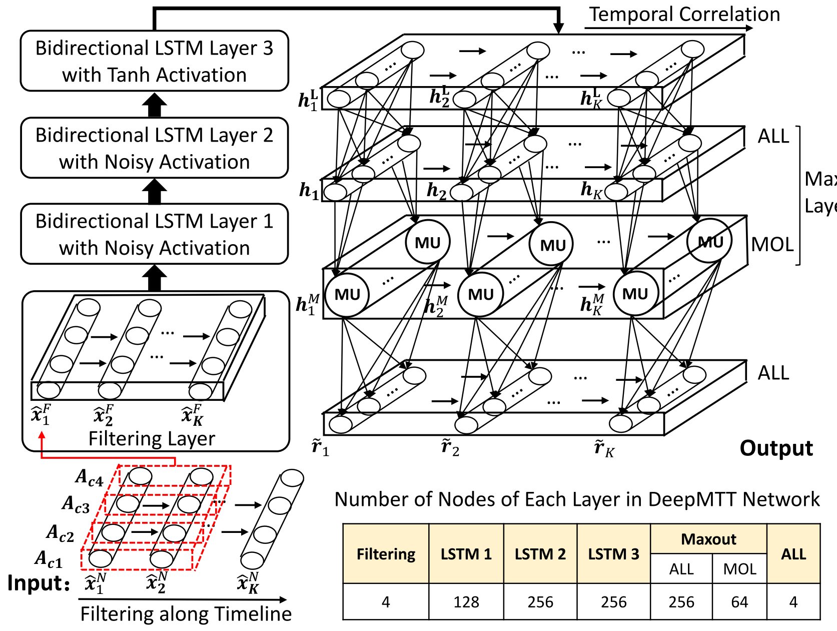

Pre-Processing
Sample
Azimuth
Distance
Ground -Truth
CV-UKF
Ground-Truth of Trajectory Segment
Estimated Trajectory Segment by CV-UKF
Y-position(m)
X-position(m)
$\boldsymbol{x}_{1:K}$
$\hat{\boldsymbol{x}}_{1:K}$
Normalization
$\hat{\boldsymbol{x}}_{1:K}/C_{\max}$
Residual
$\boldsymbol{x}_{1:K} - \hat{\boldsymbol{x}}_{1:K}$
Network Input: $\hat{\boldsymbol{x}}^N_{1:K}$
Desired Output: $\boldsymbol{r}_{1:K}$

Fig. 6. Pre-processing for trajectory data. This pre-processing outputs the normalized state sequence as network input and the residuals as the ground-truth of network output.


Temporal Correlation
Bidirectional LSTM Layer 3 with Tanh Activation
Bidirectional LSTM Layer 2 with Noisy Activation
Bidirectional LSTM Layer 1 with Noisy Activation
Filtering Layer
$\hat{x}_1^F$ $\hat{x}_2^F$ $\hat{x}_K^F$
h_1^L h_2^L h_K^L
h_1 h_2 h_K
h_1^M h_2^M h_K^M
MU
ALL
Maxout Layer
MOL
ALL
$\tilde{r}_1$ $\tilde{r}_2$ $\tilde{r}_K$
Output
A_{c4} A_{c3} A_{c2} A_{c1}
Input:
$\hat{x}_1^N$ $\hat{x}_2^N$ $\hat{x}_K^N$
Filtering along Timeline
Number of Nodes of Each Layer in DeepMTT Network

Filtering	LSTM 1	LSTM 2	LSTM 3	Maxout		ALL
				ALL	MOL	
4	128	256	256	256	64	4

Fig. 7. Structure of the DeepMTT network. In this structure, the input data flow along with the arrows through all the layers, and finally become the prediction of $\boldsymbol{r}_{1:K}$.

Second, because the LSTM structure is “good at” learning the temporal features in the sequential data, three bidirectional LSTM layers are integrated into our DeepMTT network. These LSTM layers are used to process $\hat{\boldsymbol{x}}^F_{1:K}$ and expected to output the sequence $\boldsymbol{h}^L_{1:K}$, which contains the temporal correlation information of residual $\boldsymbol{r}_{1:K}$. Thus, our DeepMTT network can be trained to predict the temporal correlation in trajectories. Specifically, the first and second layers have 128 and 256 units, respectively. The units in these two layers have the noisy activation [37]. The third layer has 256 units with the tanh activation. In the first two layers, the noisy activation function is defined as follows:

$$\phi_n(u,\xi) = \phi_p(u) + \sigma(u)\xi, \tag{21}$$

where u is the input of the noisy activation function. In addition, ξ is noise following a normal distribution. In addition, $\phi_p(u)$ is a hard-tanh activation function defined as

$$\phi_p(u) = \begin{cases} 0.5 & u > 1 \\ 0.5u & -1 \le u \le 1 \\ -0.5 & u < -1, \end{cases} \tag{22}$$

In addition, $\sigma(u)$ is a scaling function of noise ξ as follows,

$$\sigma(u) = (\mathrm{sigmoid}(\phi_p(u) - u) - 0.5)^2. \tag{23}$$

One example of the noisy activation function is shown in Fig. 8.

As introduced in [37], the noisy activation function enables the nodes to make hard-decisions, by which no information is lost through the soft-gating architecture. In addition, the added noise can alleviate the problem of gradient vanishing in the saturated region of nodes because the gradient is non-zero in that region. Thus, the loss can be back-propagated through the layers to properly train the network, even when the nodes are saturated. Furthermore, unlike the original activation function [37], we multiply the hard-tanh activation function by 0.5, as shown in (22). This trick guarantees that the noise is added to our activation function in the saturated region and unsaturated region, as shown in Fig. 8. Thus, random changes also occur in the unsaturated region to enable the process of gradient descent to avoid the local minimum in the entire region of nodes.

Third, to enhance the performance of regression fitting of our DeepMTT network, a maxout layer [31] is implemented. This layer is to select a subnetwork in DeepMTT, which can generate the maximum output subset $\boldsymbol{h}^M_{1:K}$ in the entire output space. Thus, a part of the network, which is insensitive to the input data, has been dropped, and the other part that is sensitive to the input data is trained. Therefore, our DeepMTT network becomes easier to learn the information in the input data and achieves good fitting results. Specifically, the maxout layer is designed as follows: at each time step k, $\boldsymbol{h}^L_k$ is passed through an all link layer (ALL) to generate a new tensor $\boldsymbol{h}_k$, which has N nodes. Each node is denoted as $h_k(i) \in \boldsymbol{h}_k$, $i \in [1, N]$. Then, $\boldsymbol{h}_k$ is mapped to the maxout output layer (MOL) with the maxout units (MU). Mathematically, the mapping of $\boldsymbol{h}_k$ can be represented by the following function:

$$h^M_k(i) = \max_{j\in[1:S]} h_k((i-1)S + j), \tag{24}$$

where S is the number of nodes in each subset, and $i \in [1{:}\ I]$, I is the number of subsets $I \times S = N$. (24) shows that each node $h^M_k(i)$ is the maximum node in the subset $\{h_k((i-1)S + j)\}^S_{j=1}$.

Finally, to obtain the desired output of our DeepMTT network, i.e., the prediction of the residual sequence $\tilde{\boldsymbol{r}}_{1:K}$, the output of maxout layer $\boldsymbol{h}^M_{1:K}$ must pass through another ALL to restore itself to the shape of $\boldsymbol{r}_{1:K}$ with the linear activation function: $\phi_o(u) = u$.

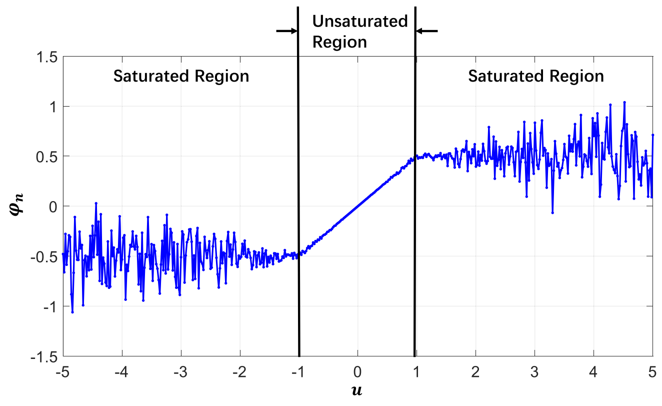


Fig. 8. Noisy activation function. It contains noises both in saturated and unsaturated regions.

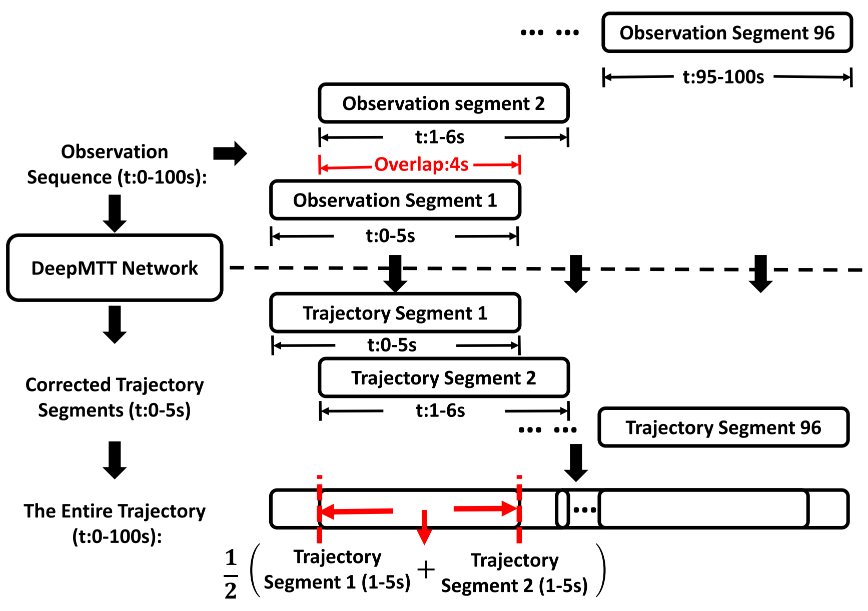


Fig. 9. Reconstruction step in the DeepMTT algorithm. Given the observation segments, DeepMTT network can generate the corresponding corrected trajectory segments, which are combined together with 4 s overlap. The value of the trajectory in overlap region is the mean of all segments.

Error backward propagation:

The loss for training the DeepMTT network is defined as the RMSE loss:

$$\mathcal{L} = \sqrt{\sum_{k=1}^{K} (\tilde{\boldsymbol{r}}_k - \boldsymbol{r}_k)^2}. \tag{25}$$

In the procedure of error backward propagation, the RMSE loss is minimized to train the DeepMTT network using the minibatch gradient descent method. Each minibatch has 100 samples. Finally, the trained DeepMTT network is obtained when the mean of errors of the estimated X and Y positions is less than a threshold ϵ. The training procedure and threshold ϵ are discussed in the next section.

3.3. Reconstruction step

To estimate the entire maneuvering trajectory, the corrected trajectory segments $\tilde{\boldsymbol{x}}_{1:K}$ output from the DeepMTT network are connected with a reconstruction step. In this step, first, the length of the overlap region between adjacent segments is determined, which is used to guarantee the stability in segment combination. With the overlap region, the corrected trajectory segments are jointed in sequence. The value of the trajectory in the overlap region is the average of the adjacent segments. Specifically, we assume that the previously estimated part of the trajectory is $\tilde{\boldsymbol{x}}_{1:K_p}$ with K_p time steps. The overlap region lasts K_o time steps. Then, we extract the next observation segment from $K_p - K_o + 1$ to $2K_p - K_o$ to calculate the next corrected trajectory segments $\tilde{\boldsymbol{x}}^{\text{next}}_{K_p-K_o+1:2K_p-K_o}$ with the beginning point $\tilde{\boldsymbol{x}}_{K_p-K_o}$. Next, we combine $\tilde{\boldsymbol{x}}_{1:K_p}$ and $\tilde{\boldsymbol{x}}^{\text{next}}_{K_p-K_o+1:2K_p-K_o}$ to make the new estimated part of the trajectory: $\tilde{\boldsymbol{x}}'_{1:2K_p-K_o}$. The value of the target state in the overlap region is calculated as:

$$\tilde{\boldsymbol{x}}'_{K_p-K_o+i} = 0.5(\tilde{\boldsymbol{x}}_{K_p-K_o+i} + \tilde{\boldsymbol{x}}^{\text{next}}_{K_p-K_o+i}), \tag{26}$$

where i is from 1 to K_o. We repeat the aforementioned steps to concatenate the remaining corrected trajectory segments and reconstruct an entire trajectory.

Note that the entire trajectory for testing the DeepMTT algorithm is set to be 100 s, and the segment output by the DeepMTT network is 5 s. In this paper, the time of overlap is set to be 4 s; then, the reconstruction step can be summarized in Fig. 9.

As shown in this figure, in the entire tracking procedure, 5-s observation segments are sequentially obtained with 1-s intervals. All observation segments are fed into the DeepMTT network one by one. Then, the corresponding corrected trajectory segments are sequentially calculated and concatenated to be our final trajectory.

4. Simulation results

In this section, we present the simulation results to verify the effectiveness of the proposed DeepMTT algorithm. First, the simulation scenarios of maneuvering target tracking are set to test the performance of the DeepMTT algorithm. Then, the training parameters for our DeepMTT network are discussed. Third, the effectiveness of the DeepMTT structure is validated by the ablation experiments. Finally, the performance of our DeepMTT algorithm is evaluated in comparison with HGMM [19] and MIE-BLUE-IMM [18] algorithms.

4.1. Simulation scenarios

In our simulation, we follow [16] to consider the X-Y plane maneuvering tracking in the ATC system. The target trajectories and corresponding observations can be calculated by (1), (2), (3) and (4) of Section 2. Moreover, based on civil aircraft maneuvering parameters [7,12], we designed 11 maneuvering target trajectories, which cover the radar tracking area and maneuvering turn rate for civil aircrafts, as shown in Table 2.

Specifically, each trajectory lasts for 100 s and is sliced into three parts, whose parameters are defined in Table 3. The trajectory “testing”, whose parameters are shown in the first row in this table, is only for testing our DeepMTT network in the training stage. The other 10 trajectories are used to validate the effectiveness of our DeepMTT algorithm in comparison with state-of-the-art MM tracking algorithms, i.e., HGMM and MIE-BLUE-IMM algorithms. The 10 trajectories in our simulation are shown in Fig. 10. The observation area of the radar in Fig. 10 is between the red dash circle and the black dot circle. The 10 trajectories are randomly located in this observation area with different maneuvering turn rates, including several extreme turn rates, such as $-9.19°$ and $9.13°$, which only appear in emergency situations.

4.2. Discussion on the training parameters

In this subsection, the training parameters, including the learning rate, batch size and training threshold ϵ of estimated position errors, are discussed. In the entire training procedure, we use different learning rates and different batch sizes according to different training losses. At the beginning of the training, the learning rate is set to 10^{-3}, and the

Table 3
Setting of 10 trajectories.

Trajectory	Prior target state $\boldsymbol{x}_0$	The first part	The second part	The third part
testing	[8000 m, 9000 m, 150 m/s, 200 m/s]	30 s, CV model	40 s, CT model, α=3.18°	30 s, CT model, α=-6.54°
1	[-18000 m, 2000 m, 150 m/s, 200 m/s]	30 s, CV model	40 s, CT model, α=3.18°	30 s, CT model, α=-6.54°
2	[-7000 m, -24000 m, 180 m/s, 220 m/s]	40 s, CT model, α=-1.08°	20 s, CV model	40 s, CT model, α=5.34°
3	[12000 m, 13000 m, 230 m/s, 190 m/s]	30 s, CV model	40 s, CT model, α=-7.16°	30 s, CT model, α=4.24°
4	[5000 m, -5000 m, 10 m/s, 330 m/s]	20 s, CV model	60 s, CT model, α=3.26°	20 s, CV model
5	[25000 m, -6000 m, 120 m/s, 230 m/s]	22 s, CV model	56 s, CT model, α=7.16°	22 s, CV model
6	[20000 m, -20000 m, -220 m/s, -200 m/s]	60 s, CT model, α=-0.58°	10 s, CV model	30 s, CT model, α=-2.21°
7	[-15000 m, -25000 m, 100 m/s, 280 m/s]	60 s, CT model, α=0.17°	30 s, CV model	10 s, CT model, α=-9.19°
8	[-25000 m, -15000 m, -120 m/s, 200 m/s]	30 s, CT model, α=-6.18°	50 s, CT model, α=8.33°	20 s, CT model, α=-2.21°
9	[-30000 m, -5000 m, 250 m/s, 180 m/s]	55 s, CT model, α=-1.15°	15 s, CT model, α=9.13°	30 s, CV model
10	[-10000 m, 25000 m, 220 m/s, 213 m/s]	40 s, CT model, α=-3.38°	20 s, CT model, α=6.82°	40 s, CT model, α=-1.17°

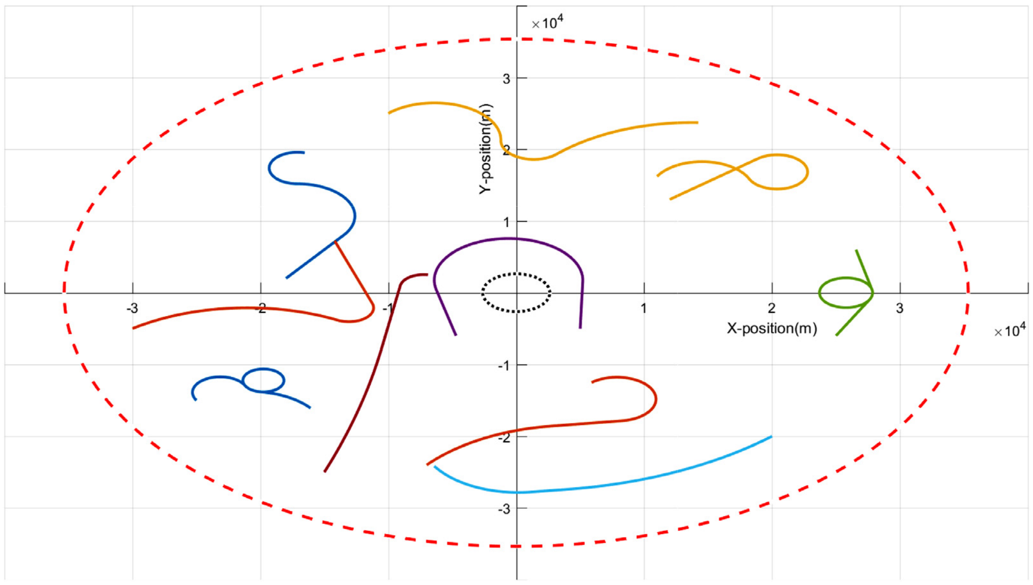


Fig. 10. Ten maneuvering target trajectories. They almost cover different regions of the radar detection ranges and different kinds of maneuvering patterns.

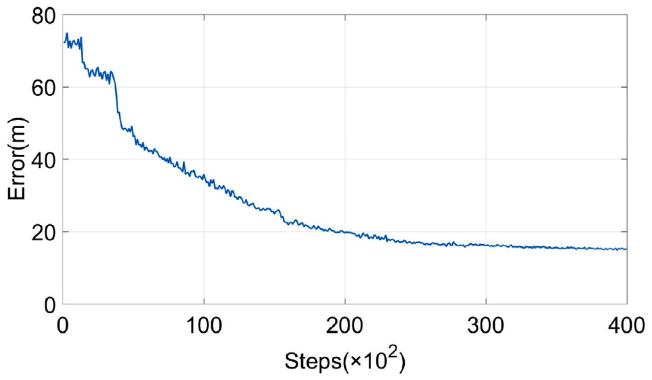


Fig. 11. Errors of the position estimated by the DeepMTT network in training.

batch size is 100 samples. After 5000 training steps, we find that the training loss no longer reduces. This because a large learning rate may make the network parameters fluctuate in a large range, which hinders the training loss from decreasing. Therefore, we reduce the learning rate to 10^{-4} and train the DeepMTT network in another 10,000 training steps. Afterwards, we further reduce the learning rate to be 10^{-5} and shrink the batch size to be 20 samples in each training step for the additional 25,000 training steps. In the entire training stage, the estimated position errors every 10^2 training steps are shown sequentially in Fig. 11.

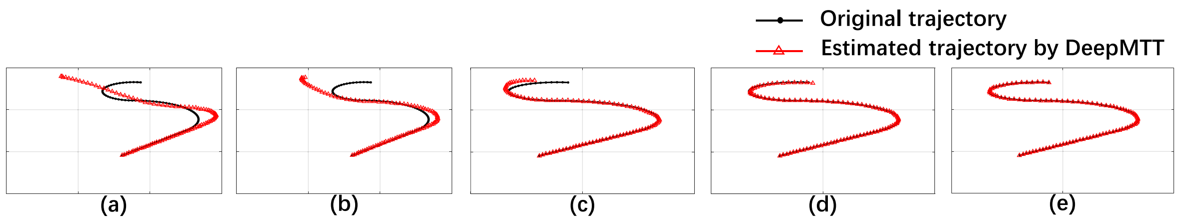
When the errors are approximately 41, 34, 27, 20 and 15 m, we test the performance of our DeepMTT algorithm by tracking the same 100-s maneuvering target trajectory "testing" in Table 3. The results are shown in Fig. 12.

Fig. 12-(a), (b) and (c) shows that when the estimated position errors in training are larger than 20 m, the DeepMTT algorithm cannot precisely track the trajectory of the maneuvering target. In particular, during the second half of the tracking period[4], the tracking algorithm fails to track the target and outputs the wrong positions. On the contrary, when the estimated position errors are smaller than 20 m, as shown in 12-(d) and (e), our tracking algorithm can precisely track the target during both the first and second halves of the tracking period. For example, the RMSE of position estimation in (d) is 34.6 m, which is smaller than that of the state-of-the-art maneuvering target tracking algorithms [18,19].[5] Therefore, we set the threshold ϵ of estimated position errors to be 20 m. Once the DeepMTT network is trained to output smaller errors than 20 m, our DeepMTT algorithm can be guaranteed to track the maneuvering target with a tolerable RMSE.

4.3. Ablation experiments

In this subsection, three ablation experiments are conducted to validate the effectiveness of the filtering layer, maxout layer and noisy activation function in our DeepMTT network. The ablation results are discussed as follows.

[4] The target moves from the bottom up along with the trajectory; thus, the second half of tracking period is the top half of the trajectory in the sub-figures.
[5] The performances of those state-of-the-art algorithms are shown in Section 4.4.



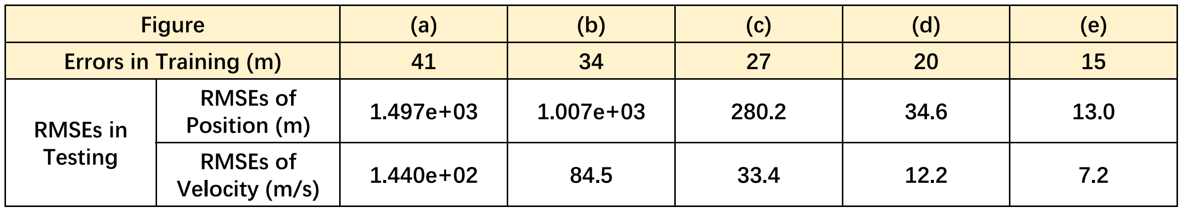
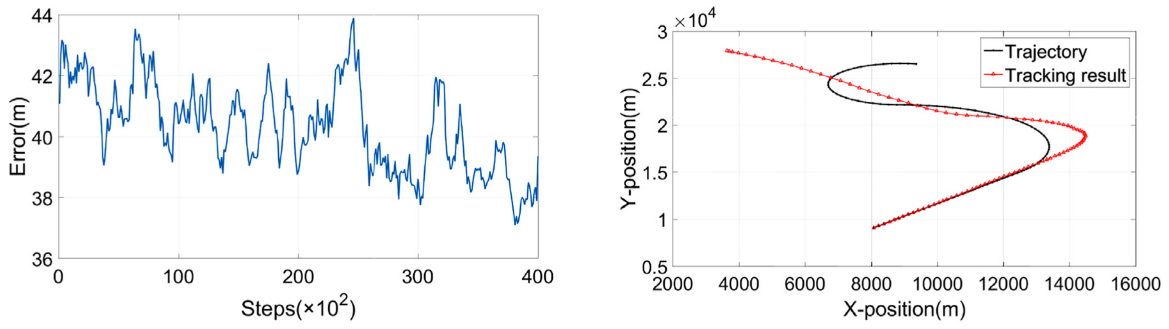


Figure		(a)	(b)	(c)	(d)	(e)
Errors in Training (m)		41	34	27	20	15
RMSEs in Testing	RMSEs of Position (m)	1.497e+03	1.007e+03	280.2	34.6	13.0
	RMSEs of Velocity (m/s)	1.440e+02	84.5	33.4	12.2	7.2

Fig. 12. Testing results in the training procedure.



(a) Errors of position in training

(b) Tracking by the DeepMTT algorithm without the filtering layer

Fig. 13. Training results of the DeepMTT network without the filtering layer.

Table 4
Training results of DeepMTT network without the filtering layer.

Error after training(m)	RMSEs in testing	
	RMSE of position(m)	RMSE of velocity(m/s)
39.4	1.479e+03	1.472e+02

Table 5
Training results of the DeepMTT network without the maxout layer.

Error after training(m)	RMSEs in testing	
	RMSE of position(m)	RMSE of velocity(m/s)
32.1	7.147e+02	62.3

DeepMTT network without the filtering layer:

The DeepMTT network without the filtering layer is trained with 40,000 steps based on our LAST database, which is identical to that in Section 4.2. The training results are shown in Fig. 13.

In Fig. 13(a), we plot the estimated position errors of the DeepMTT network without the filtering layer with the training steps. Obviously, the errors are larger than 20 m after training and seriously fluctuate, although they slightly reduce in the later period of training. Hence, as shown in Fig. 13(b), the DeepMTT algorithm cannot correctly track the trajectory “testing”.

The details of training results are shown in Table 4. In this table, we find that the error after training is 39.4 m. As a result, according to Section 4.2, when the DeepMTT network has no filtering layer, our DeepMTT algorithm cannot correctly track the maneuvering target after training. As we further observe in Table 4, the tracking RMSEs of position and velocity are 1.479e + 03 m and 1.472e + 02 m/s, respectively. These tracking RMSEs are so large that the radar has undoubtedly lost the target.

DeepMTT network without the maxout layer:

The identical training and testing procedures are run in the DeepMTT network without the maxout layer. The training results are shown in Fig. 14.

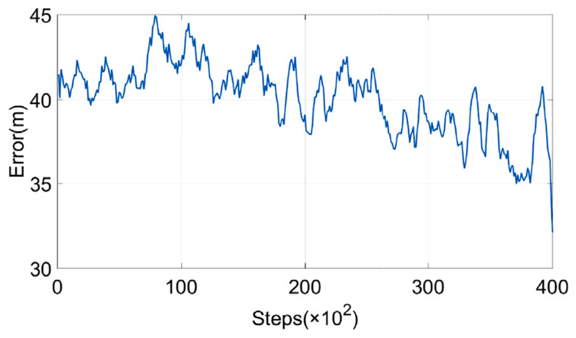
Fig. 14(a) shows that the estimated position errors fluctuate to approximately 32 when the DeepMTT network has no maxout layer. Although the errors are smaller than that of the DeepMTT network without a filtering layer, they remain larger than 20 m. Thus, without the maxout layer, our DeepMTT algorithm cannot precisely track the target, as shown in Fig. 14(b). The details of the tracking RMSE of position and velocity are shown in Table 5.

The values of RMSEs of the position and velocity are larger than those of the state-of-the-art maneuvering target-tracking algorithm [18,19]. Therefore, without the maxout layer, our DeepMTT algorithm cannot perform well even after a long time of training.

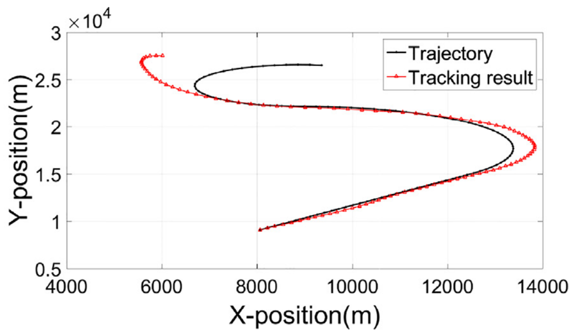
DeepMTT network without the noisy activation function:

We replace the noisy activation function in our DeepMTT network with the original *tanh* activation function. Then, we train and test this network using the procedure of the last subsection. The results are shown in Fig. 15.

As we observe in Fig. 15(a), the errors slowly decline during the training stage and fluctuate at approximately 37 m at the end of the training, which indicates that the parameters in the DeepMTT network fall into local minima, and the error cannot decrease. Thus, without the noisy activation function, it is difficult to obtain the well trained DeepMTT network, and the DeepMTT algorithm cannot precisely track

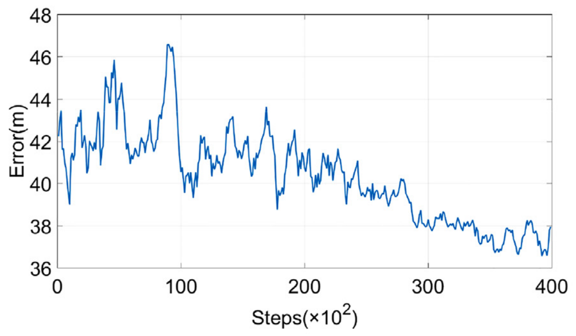


(a) Errors of position in training

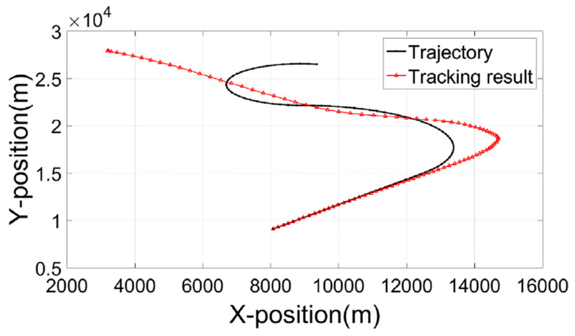


(b) Tracking by the DeepMTT algorithm without the maxout layer

Fig. 14. Training results of the DeepMTT network without the maxout layer.



(a) Errors of position in training



(b) Tracking by the DeepMTT algorithm without the noisy activation function

Fig. 15. Training results of the DeepMTT network without noisy activation function.

Table 6
Training results of the DeepMTT network without the noisy activation function.

Error after training(m)	RMSEs in testing	
	RMSE of position(m)	RMSE of velocity(m/s)
37.9	1.551e+03	1.475e+02

the target, as shown in Fig. 15(b). The details of tracking RMSEs of the position and velocity are shown in Table 6.

Obviously, the values of RMSEs of position and velocity are too large to satisfy the requirement of maneuvering target tracking.

4.4. Evaluation of the DeepMTT algorithm

To evaluate the performance of our DeepMTT algorithm, we set 10 maneuvering target-tracking scenarios with 10 different trajectories (trajectories 1–10), as mentioned in Section 4.1. The parameters of these trajectories are shown in Table 3. In each scenario, 100 Monte Carlo simulations for tracking with our DeepMTT algorithm are run in comparison with the HGMM and MIE-BLUE-IMM algorithms. We evaluate the performance of all three aforementioned algorithms with the means and deviations of the tracking RMSE in those 100 simulations. Note that we apply the default parameter settings of HGMM and MIE-BLUE-IMM in [19] and [18].

The tracking results for all 10 trajectories are shown in Figs. 16–25. In these figures, the green lines with star markers are the tracking results of the HGMM algorithm, the black lines with dot markers are the tracking results of the MIE-BLUE-IMM algorithm, the red lines with circle markers are the tracking results of our DeepMTT algorithm, and the blue lines are the original trajectories. Obviously, our DeepMTT algorithm remains precise and stable for tracking all trajectories that cover different beginning states, velocities, and turn rates. Furthermore, we investigate the means and deviations of the tracking RMSE (including position and velocity) of all 100 MC runs for each trajectory. The results are shown in Tables 7, 8, 9 and 10. In these tables, the tracking RMSEs of the three algorithms are compared according to different parts of different trajectories in Table 3. The smallest tracking RMSE is colored sandy-brown in each row, which indicates that the corresponding algorithm performs best. Obviously, in Tables 7 and 9, all tracking RMSEs of our DeepMTT algorithm are colored sandy-brown, except the ones for the tracking of the second and third parts in trajectory 9. Hence, our DeepMTT algorithm provides the smallest tracking RMSEs for all different maneuvering trajectories in comparison with the other two algorithms. Our DeepMTT algorithm also performs well on the deviations of tracking RMSE, which presents the stability in tracking. Although for some parts of some trajectories, the tracking RMSEs of HGMM and MIE-BLUE-IMM are better, our DeepMTT algorithm retains a comparable performance. In summary, our DeepMTT algorithm outperforms the state-of-the-art HGMMM and MIE-BLUE-IMM algorithms for tracking maneuvering targets.

4.5. Computational complexity

In this subsection, the computational complexity of our DeepMTT algorithm is discussed in comparison with HGMM and MIE-BLUE-IMM algorithms, by means of testing the computational time in single

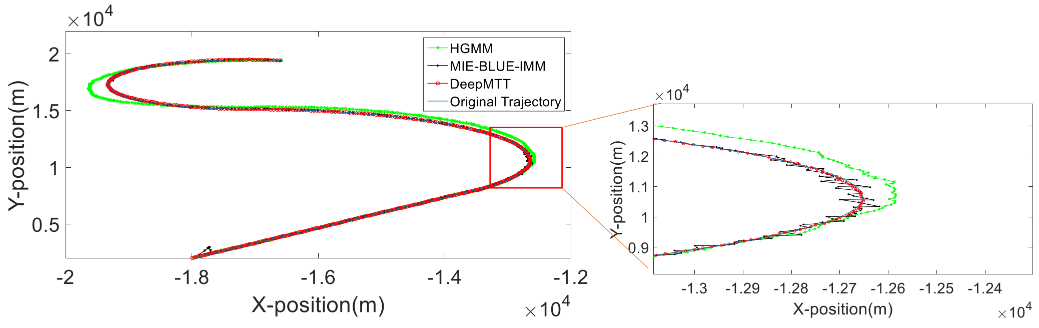


Fig. 16. Tracking results of the 1st trajectory.

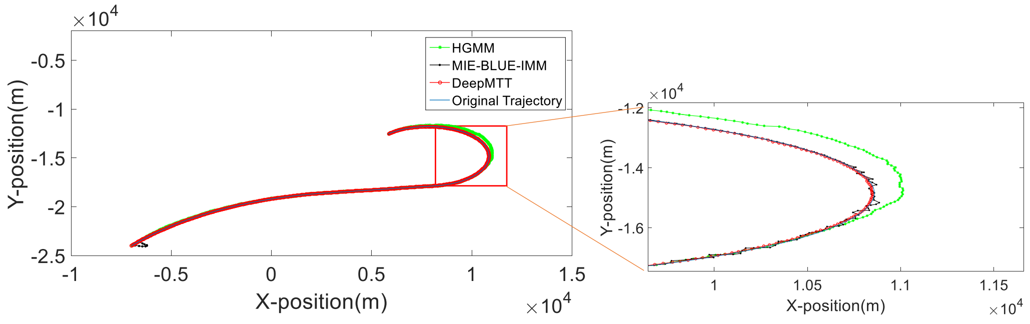


Fig. 17. Tracking results of the 2nd trajectory.

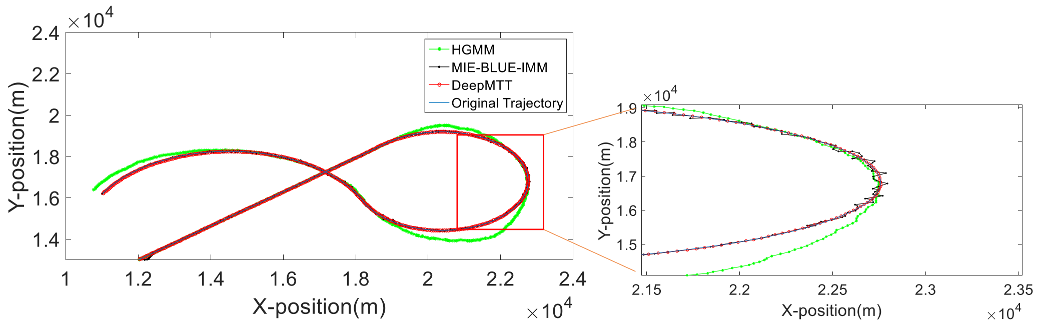


Fig. 18. Tracking results of the 3rd trajectory.

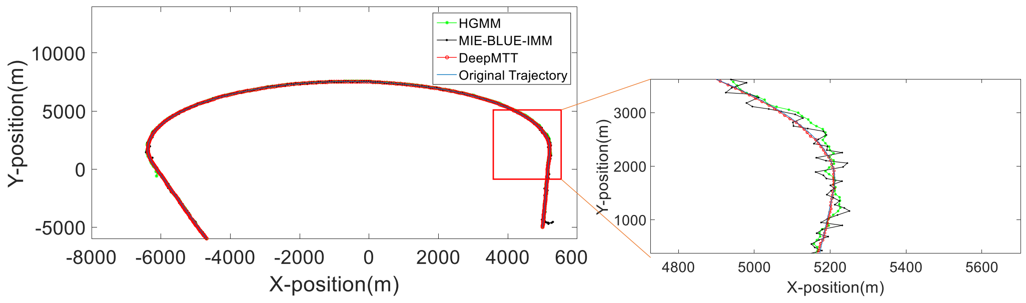


Fig. 19. Tracking results of the 4th trajectory.

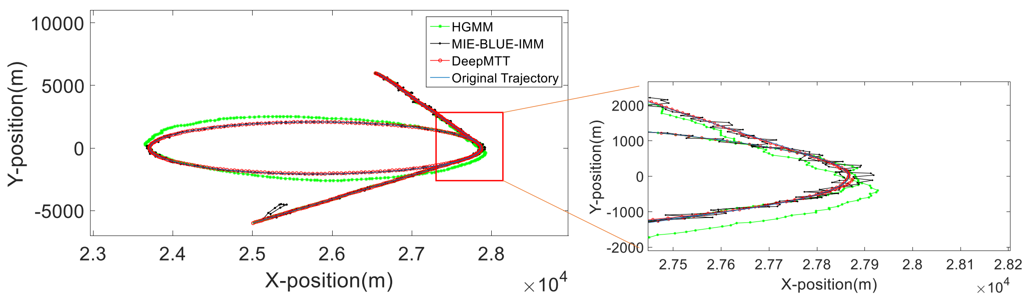


Fig. 20. Tracking results of the 5th trajectory.

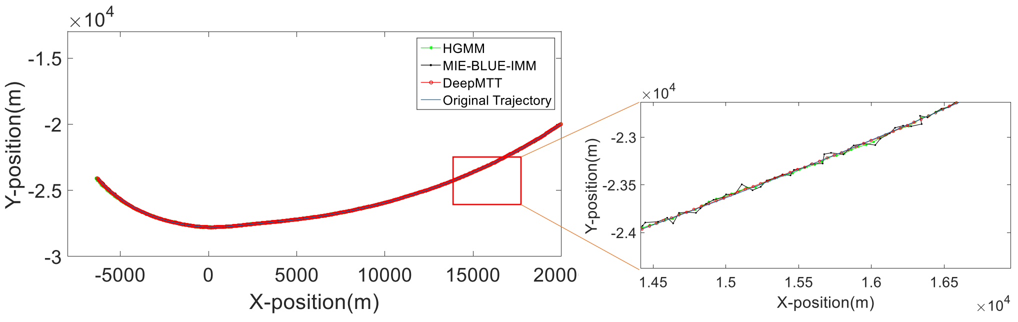


Fig. 21. Tracking results of the 6th trajectory.

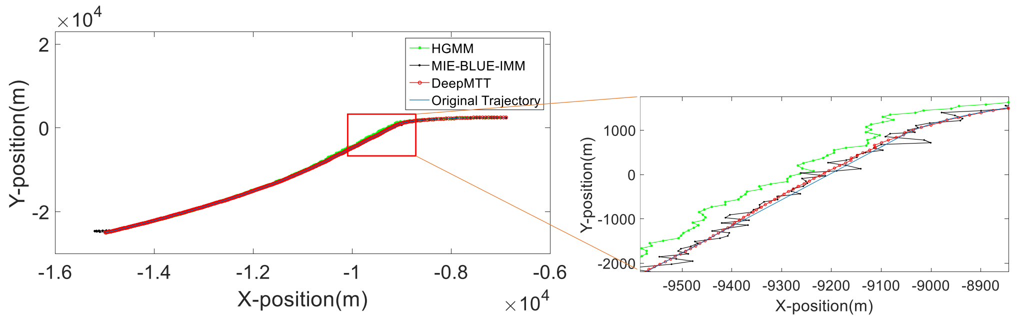


Fig. 22. Tracking results of the 7th trajectory.

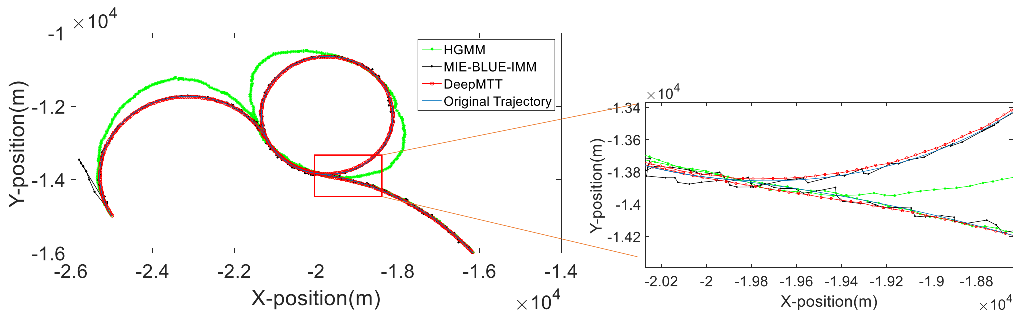


Fig. 23. Tracking results of the 8th trajectory.

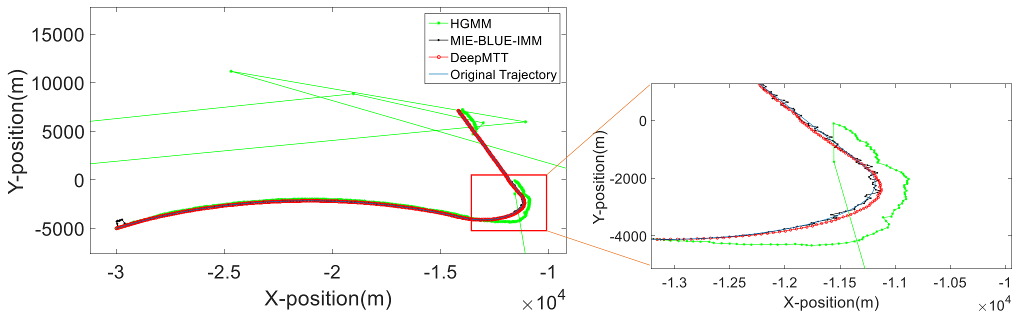


Fig. 24. Tracking results of the 9th trajectory.

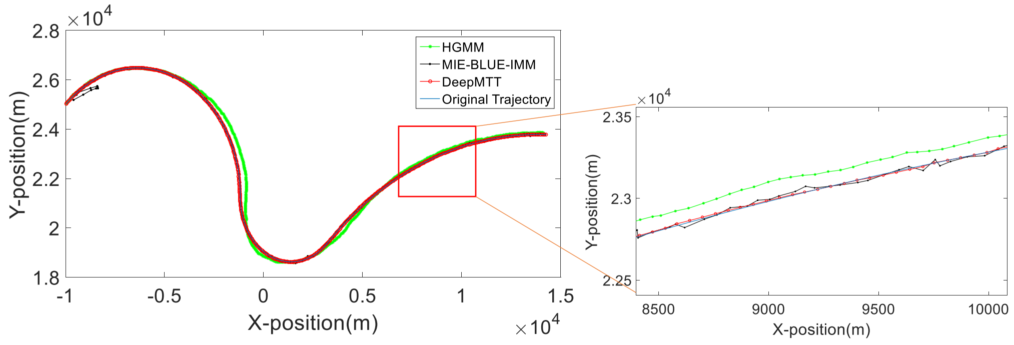


Fig. 25. Tracking results of the 10th trajectory.

Table 7
Means of the position tracking RMSE for all trajectories with HGMM, MIE-BLUE-IMM and DeepMTT algorithms, respectively.

Trajectories		HGMM (m)	MIE-BLUE-IMM (m)	DeepMTT (m)
1	**The first part**	22.96	121.66	12.67
	The second part	138.29	55.24	12.98
	The third part	237.29	57.76	24.90
2	**The first part**	53.47	94.02	13.97
	The second part	23.67	56.45	12.12
	The third part	140.91	55.17	13.24
3	**The first part**	23.45	56.50	12.42
	The second part	278.06	59.89	17.63
	The third part	160.58	58.12	16.74
4	**The first part**	22.76	78.95	11.02
	The second part	35.04	54.32	13.24
	The third part	8.20×10^5	52.80	14.40
5	**The first part**	24.37	209.16	13.66
	The second part	204.75	54.71	16.14
	The third part	102.63	61.06	13.62
6	**The first part**	24.91	179.29	13.61
	The second part	20.84	63.76	12.76
	The third part	52.08	60.70	19.67
7	**The first part**	28.00	64.39	10.67
	The second part	1.60×10^5	52.99	12.23
	The third part	9.58×10^5	51.87	15.69
8	**The first part**	218.87	196.13	15.31
	The second part	236.50	57.70	30.63
	The third part	38.52	58.75	20.29
9	**The first part**	83.61	104.38	10.53
	The second part	150.20	48.86	83.47
	The third part	7.82×10^6	51.49	64.29
10	**The first part**	117.55	183.77	12.90
	The second part	125.73	51.23	16.35
	The third part	101.52	57.96	12.46

Table 8
Deviations of the position tracking RMSE for all trajectories with HGMM, MIE-BLUE-IMM and DeepMTT algorithms, respectively.

Trajectories		HGMM (m)	MIE-BLUE-IMM (m)	DeepMTT (m)
1	**The first part**	2.92	13.99	2.58
	The second part	12.45	3.06(m)	2.77
	The third part	24.45	3.84	15.65
2	**The first part**	6.87	10.06	2.26
	The second part	3.72	4.61	2.65
	The third part	11.77	2.80	2.57
3	**The first part**	3.08	3.36	2.37
	The second part	31.43	3.82	12.35
	The third part	20.29	4.21	11.82
4	**The first part**	3.55	6.48	2.73
	The second part	2.31	2.40	2.32
	The third part	2.66×10^6	4.14	3.85
5	**The first part**	3.37	18.79	2.72
	The second part	15.63	2.74	2.27
	The third part	19.23	5.83	3.49
6	**The first part**	2.82	16.42	2.06
	The second part	3.67	9.95	3.88
	The third part	7.55	3.87	10.87
7	**The first part**	2.62	6.20	1.78
	The second part	4.96×10^5	18.94	2.51
	The third part	2.69×10^6	25.47	6.70
8	**The first part**	25.31	17.36	3.00
	The second part	18.12	11.69	25.71
	The third part	7.43	14.95	13.42
9	**The first part**	6.46	12.97	1.95
	The second part	17.39	3.29	34.68
	The third part	3.51×10^6	3.15	47.14
10	**The first part**	12.95	19.45	2.35
	The second part	17.98	4.57	9.91
	The third part	11.37	3.47	2.66

iteration of tracking. For fair comparison, we test all the algorithms with the same Intel Core i7-3770 CPU at 3.4 GHz and 4 GB RAM. In the tracking process, our DeepMTT algorithm consumes 16.4 ms to compute one iteration in tracking process, and HGMM and MIE-BLUE-IMM algorithms consume 201.8 ms and 64.1 ms, respectively. Obviously, our DeepMTT algorithm is faster than both HGMM and MIE-BLUE-IMM algorithms. Hence, our DeepMTT algorithm is suitable for real-time application.

Table 9
Means of the velocity tracking RMSE for all trajectories with HGMM, MIE-BLUE-IMM and DeepMTT algorithms, respectively.

Trajectories		HGMM (m/s)	MIE-BLUE-IMM (m/s)	DeepMTT (m/s)
1	**The first part**	12.38	159.55	3.91
	The second part	45.30	161.19	9.08
	The third part	84.74	162.34	13.88
2	**The first part**	18.57	178.53	4.59
	The second part	12.84	153.39	4.11
	The third part	68.83	183.68	7.85
3	**The first part**	13.86	190.53	8.45
	The second part	108.93	191.92	7.51
	The third part	59.96	189.75	7.71
4	**The first part**	12.72	168.39	4.14
	The second part	25.12	212.51	5.27
	The third part	2.16×10^6	184.42	6.35
5	**The first part**	11.12	160.03	4.25
	The second part	88.87	168.01	7.51
	The third part	25.73	147.30	5.35
6	**The first part**	13.31	182.39	6.61
	The second part	10.73	161.23	9.74
	The third part	23.49	186.33	11.80
7	**The first part**	14.26	168.58	2.92
	The second part	6.13×10^5	163.29	5.07
	The third part	3.21×10^6	190.15	9.91
8	**The first part**	75.73	150.80	7.64
	The second part	98.08	152.08	12.88
	The third part	19.41	148.39	8.56
9	**The first part**	25.34	176.82	4.01
	The second part	111.65	198.92	27.13
	The third part	2.03×10^7	180.54	9.80
10	**The first part**	45.85	195.18	5.59
	The second part	71.72	196.58	9.55
	The third part	29.01	186.52	5.28

Table 10
Deviations of the velocity tracking RMSE for all trajectories with HGMM, MIE-BLUE-IMM and DeepMTT algorithms, respectively.

Trajectories		HGMM (m/s)	MIE-BLUE-IMM (m/s)	DeepMTT (m/s)
1	**The first part**	1.61	2.11	0.91
	The second part	1.30	1.53	1.03
	The third part	2.95	1.59	3.33
2	**The first part**	0.97	1.95	0.79
	The second part	1.74	2.59	1.04
	The third part	1.76	1.58	1.08
3	**The first part**	1.70	2.05	1.17
	The second part	2.96	1.54	2.56
	The third part	2.40	1.78	1.48
4	**The first part**	1.58	2.45	1.14
	The second part	0.69	1.18	0.72
	The third part	6.28×10^6	2.36	2.44
5	**The first part**	1.26	2.94	0.93
	The second part	1.96	1.50	1.47
	The third part	3.53	2.40	2.08
6	**The first part**	1.16	2.71	1.14
	The second part	1.61	3.71	2.31
	The third part	1.28	1.96	2.96
7	**The first part**	1.13	1.67	0.51
	The second part	2.55×10^6	2.06	0.99
	The third part	9.32×10^6	2.27	3.05
8	**The first part**	1.99	2.31	0.89
	The second part	2.44	1.56	5.75
	The third part	2.06	2.33	3.02
9	**The first part**	1.01	2.12	0.60
	The second part	3.14	1.91	7.54
	The third part	1.13×10^7	2.08	4.05
10	**The first part**	1.82	1.94	0.88
	The second part	3.27	2.05	3.28
	The third part	1.72	2.07	1.34

5. Conclusions

In this paper, we have proposed a new bidirectional LSTM-based DeepMTT algorithm for civil aircraft tracking. First, a generative trajectory database of maneuvering target is built to offer sufficient samples of maneuvering trajectories. Then, a new DeepMTT network is proposed based on the bidirectional LSTM structure. Using this network, we can estimate the trajectory segments with the corresponding nonlinear radar observations. Finally, an entire estimated trajectory is estimated with a reconstruction step. The simulation results verify that in comparison with state-of-the-art maneuvering target tracking algorithms, our DeepMTT algorithm improves the performance on maneuvering-civil-aircraft tracking scenarios.

In the application with our DeepMTT algorithm, we need to consider different radar tracking problems with different noises. To handle these problems, different noises have been added to the observations of the samples in our LAST database to improve the generalization performance of our DeepMTT network. For example, the deviations of azimuth noise σ_θ and distance noise σ_r are randomly sampled in the range of $[0.401^\circ, 0.516^\circ]$ and $[8\,m, 13\,m]$, respectively. Hence, our DeepMTT network is fit for all the observation noise within the aforementioned ranges. However, for total different noise levels, our DeepMTT network do need to be fined tune to achieve the proper tracking results.

There are three promising directions for future work. First, this paper only considers the X-Y plane scenario of maneuvering target tracking. We can extend it into three-dimensional tracking scenarios. Second, although our training data cover the entire range of turn rate from -10° to 10°, the tracking performance continues to seriously decrease when the target moves with a constant turn rate close to $\pm 10^\circ$. Hence, a better network should be designed to solve this problem, which is another promising future work. Third, because different parts of our network will take different effects on target tracking. How to specify these different effects in different layers is valuable for further understanding the network and improving the tracking performance. Hence, the layer effect specification is also a promising future work.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.inffus.2019.06.012.

References

[1] R. Kalman, A new approach to linear filtering and prediction problems, J. Basic Eng. 82 (1) (1960) 35–45.
[2] J.K. Uhlmann, Simon J. Julier, A new extension of the kalman filter to nonlinear systems, Signal Processing, Sensor Fusion, and Target Recognition VI. International Society for Optics and Photonics 3068 (1997) 182–193.
[3] S. Julier, J. Uhlmann, H.F. Durrantwhyte, A new method for the nonlinear transformation of means and covariances in filters and estimators, IEEE Trans. Autom. Control 45 (3) (2000) 477–482.
[4] M. Arulampalam, S. Maskell, N. Gordon, T. Clapp, A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking, IEEE Trans. Signal Process. 50 (2) (2002) 174–188.
[5] B. Zhang, Z. Li, A. Perina, A.D. Bue, V. Murino, J. Liu, Adaptive local movement modeling for robust object tracking, IEEE Trans. Circuits Syst. Video Technol. 27 (7) (2017) 1515–1526.
[6] Z. Chu, D. Zhu, S.X. Yang, Observer-based adaptive neural network trajectory tracking control for remotely operated vehicle, IEEE Trans. Neural Netw. Learn. Syst. 28 (7) (2017) 1633.
[7] X. Li, Y. Bar-Shalom, Design of an interacting multiple model algorithm for air traffic control tracking, IEEE Trans. Control Syst. Technol. 1 (3) (1993) 186–194.
[8] H. Wang, T. Kirubarajan, Y. Bar-Shalom, Precision large scale air traffic surveillance using imm/assignment estimators, IEEE Trans. Aerosp. Electron. Syst. 35 (1) (1999) 255–266.
[9] X.R. Li, Y. Bar-Shalom, Multiple-model estimation with variable structure, IEEE Trans. Autom. Control 41 (4) (1996) 478–493.
[10] X.R. Li, V.P. Jilkov, Survey of maneuvering target tracking. part v. multiple-model methods, IEEE Trans. Aerosp. Electron. Syst. 41 (4) (2005) 1255–1321.
[11] H.A. Blom, Y. Bar-Shalom, The interacting multiple model algorithm for systems with markovian switching coefficients, IEEE Trans. Autom. Control 33 (8) (1988) 780–783.

[12] E. Daeipour, Y. Bar-Shalom, Imm tracking of maneuvering targets in the presence of glint, IEEE Trans. Aerosp. Electron. Syst. 34 (3) (1998) 996–1003.
[13] W. Li, Y. Jia, J. Du, F. Yu, Gaussian mixture phd smoother for jump markov models in multiple maneuvering targets tracking, in: Proceedings of the 2011 American Control Conference, 2011, pp. 3024–3029, doi:10.1109/ACC.2011.5991161.
[14] L. Gao, J. Xing, Z. Ma, J. Sha, X. Meng, Improved imm algorithm for nonlinear maneuvering target tracking, Procedia Engineering 29 (2012) 4117–4123.
[15] V. Petridis, A. Kehagias, A multi-model algorithm for parameter estimation of time–varying nonlinear systems, Automatica 34 (4) (1998) 469–475.
[16] X.R. Li, Z. Zhao, X.B. Li, General model-set design methods for multiple-model approach, IEEE Trans. Autom. Control 50 (9) (2005) 1260–1276.
[17] X.R. Li, Multiple-model estimation with variable structure. ii. model-set adaptation, IEEE Trans. Autom. Control 45 (11) (2000) 2047–2060, doi:10.1109/9.887626.
[18] H. Sheng, W. Zhao, J. Wang, Interacting multiple model tracking algorithm fusing input estimation and best linear unbiased estimation filter, IET Radar Sonar Navig. 11 (1) (2017) 70–77.
[19] L. Xu, X.R. Li, Z. Duan, Hybrid grid multiple-model estimation with application to maneuvering target tracking, IEEE Trans. Aerosp. Electron. Syst. 52 (1) (2016) 122–136, doi:10.1109/TAES.2015.140423.
[20] T. Li, J.M. Corchado, H. Chen, J. Bajo, Track a smoothly maneuvering target based on trajectory estimation, in: 2017 20th International Conference on Information Fusion (Fusion), 2017, pp. 1–8, doi:10.23919/ICIF.2017.8009731.
[21] T. Li, H. Chen, S. Sun, J.M. Corchado, Joint smoothing and tracking based on continuous-time target trajectory function fitting, IEEE Trans. Autom. Sci. Eng. (2018) 1–8, doi:10.1109/TASE.2018.2882641.
[22] L. Chin, Application of neural networks in target tracking data fusion, IEEE Trans. Aerosp. Electron. Syst. 30 (1) (1994) 281–287.
[23] F. Amoozegar, M.K. Sundareshan, Target tracking by neural network maneuver modeling, in: Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on, 6, IEEE, 1996, pp. 3932–3937.
[24] D. Smith, S. Singh, Approaches to multisensor data fusion in target tracking: a survey, IEEE Trans. Knowl. Data Eng. 18 (12) (2006) 1696–1710.
[25] F. Amoozegar, S.H. Sadati, Target tracking by neural network maneuver detection and input estimation, in: Radar Conference, 1995., Record of the IEEE 1995 International, IEEE, 1995, pp. 143–148.
[26] Y.C. Wong, M.K. Sundareshan, Data fusion and tracking of complex target maneuvers with a simplex-trained neural network-based architecture, in: Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on, 2, IEEE, 1998, pp. 1024–1029.
[27] F.-B. Duh, C.-T. Lin, Tracking a maneuvering target using neural fuzzy network, IEEE Trans. Syst. Man Cybern., 34 (1) (2004) 16–33.
[28] S.S. Kim, S.I. Chien, C.K. An, S.D. Lee, T.H. Lee, Recurrent neural network, KITE J. Electron. Eng. 5 (1) (1994) 106–112.
[29] A. Graves, J. Schmidhuber, Framewise phoneme classification with bidirectional lstm and other neural network architectures, Neural Netw. 18 (5–6) (2005) 602–610.
[30] A. Graves, et al., Supervised sequence labelling with recurrent neural networks, 385, Springer, 2012.
[31] I.J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, Y. Bengio, Maxout networks, in: Proceedings of the International Conference on Machine Learning (ICML) (2013), 2013, pp. 1319–1327.
[32] X.R. Li, V.P. Jilkov, Survey of maneuvering target tracking. part i. dynamic models, IEEE Trans. Aerosp. Electron. Syst. 39 (4) (2003) 1333–1364.
[33] R. Weber, J. Schanne, Airport Surveillance Radar Model 11 (ASR-11) FAA Test and Evaluation Master Plan (TEMP), U.S*A*. Department of Transportation, 1998. DOT/FAA/CT-TN97/27
[34] J. Steele, The fastest passenger jets in the sky, Accessed March 29, 2017 (https://thepointsguy.com/2017/03/fastest-passenger-jets/).
[35] J. Liu, Z. Wang, M. Xu, A kalman estimation based rao-blackwellized particle filtering for radar tracking, IEEE Access PP (99) (2017). 1–1
[36] J. Lan, X.R. Li, Tracking of maneuvering non-ellipsoidal extended object or target group using random matrix., IEEE Trans. Signal Process. 62 (9) (2014) 2450–2463.
[37] C. Gulcehre, M. Moczulski, M. Denil, Y. Bengio, Noisy activation functions, in: International Conference on Machine Learning, 2016, pp. 3059–3068.