Results in Engineering 27 (2025) 106324

Contents lists available at ScienceDirect

Results in Engineering

journal homepage: www.sciencedirect.com/journal/results-in-engineering

Research paper

Real-time path planning in dynamic environments using LSTM-augmented A* search

Manar Lashin [b,c], Shady Y. El-mashad [c,d], Abdullah T. Elgammal [a,b,*]

[a] The British University in Egypt, El-Sherouk City, Egypt
[b] Electrical Eng. Dept., Benha Faculty of Engineering, Benha University, Benha, Egypt
[c] Faculty of Computer Science, Benha National University, El-Obour City, Benha, Egypt
[d] Department of Electrical Engineering, Faculty of Engineering at Shoubra, Benha University, Cairo, Egypt

ARTICLE INFO

Keywords:
Path planning
Dynamic obstacle avoidance
LSTM networks A* algorithm
Real-time navigation

ABSTRACT

This paper presents a novel predictive heuristic framework for simulated real-time path planning in dynamic environments, integrating Long Short-Term Memory (LSTM) neural networks, Kalman filtering, and the A* search algorithm. The proposed *LSTM-Augmented A** method uses historical obstacle trajectories to predict future positions, significantly reducing the computational overhead associated with frequent re-planning and enhancing collision avoidance. To robustly handle prediction uncertainties and measurement noise, an adaptive Kalman filter is integrated alongside the LSTM predictions, forming a hybrid prediction model. The entire study—including obstacle modeling, hybrid prediction methods, path-planning algorithm implementation, and performance validation—is conducted using MATLAB® and its Deep Learning Toolbox simulations. Initially, extensive simulations in synthetic environments are used to evaluate the framework's responsiveness to complex spatiotemporal obstacle dynamics. Subsequently, rigorous validation is performed using established benchmark simulation maps, notably `Berlin_0_256.map`, which represent realistic complexities and noisy conditions. Simulation results demonstrate substantial improvements in path efficiency, computational speed, prediction accuracy, path smoothness, and safety metrics. The proposed integration of LSTM predictions and Kalman filtering within the A* heuristic enables proactive, near-optimal path generation while preserving theoretical guarantees of admissibility and completeness. These findings underline the robustness and practical applicability of combining deep-learning predictions with classical heuristic methods in simulated dynamic path-planning scenarios.

1. Introduction

Path planning is a foundational challenge in robotics and autonomous systems, wherein an agent must compute a collision-free trajectory from a start to a goal configuration while accounting for environmental constraints and uncertainties [1]. Classical methods such as Dijkstra's algorithm [2] and A* [3] have been extensively used in structured, static settings due to their optimality and completeness guarantees. However, their applicability diminishes in real-world scenarios characterized by dynamic obstacles, sensor noise, and rapidly changing spatial configurations [4]. The classical A* algorithm, while optimal in static environments, becomes computationally prohibitive when frequent re-planning is required, as even small environmental changes can invalidate precomputed heuristics. Variants like D* Lite [4] and Anytime Repairing A* [5] attempt to mitigate this by incrementally updating paths, yet their performance can degrade in densely dynamic or high-dimensional environments [6].

To address these challenges, recent research has increasingly turned to hybridizing classical search algorithms with deep learning, particularly through the integration of temporal prediction models such as Long Short-Term Memory (LSTM) networks. These networks can infer motion patterns from historical data, enabling anticipatory planning in dynamic contexts. Some approaches embed LSTM with Convolutional Neural Networks (CNNs) for scene understanding and trajectory prediction before path computation via A* [7]. While these methods yield promising results in perception-rich settings, they often introduce additional computational overhead, retraining complexity, and dependency on visual input modalities.

* Corresponding author at: The British University in Egypt, El-Sherouk City, Egypt.
E-mail addresses: manar.hosny@cs.bnu.edu.eg, manar.lashin@bhit.bu.edu.eg (M. Lashin), shady.elmashad@bnu.edu.eg (S.Y. El-mashad), abdalla.elgammal@bue.edu.eg, abdalla.elgammal@bhit.bu.edu.eg (A.T. Elgammal).

https://doi.org/10.1016/j.rineng.2025.106324
Received 16 March 2025; Received in revised form 17 June 2025; Accepted 15 July 2025
Available online 21 July 2025
2590-1230/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Table 1
Comparison of LSTM-A* with classical and Learning-Based planning approaches.

Evaluation Criterion	LSTM-A* (Ours)	Classical A* [3]	D* Lite [4]	Anytime A* [12]	RL-LSTM [13]	SAC-LSTM [11]	RL-based [14]
Algorithmic Basis	LSTM + Kalman + A*	Heuristic Search	Incremental Search	Heuristic + Anytime Repair	RL with LSTM	RL with SAC + LSTM	RL Policy Learning
Real-Time Capability	High	Moderate (static env.)	Good (incremental)	High (anytime replanning)	Medium	Medium	High (after training)
Prediction Model	LSTM + Kalman Filter	None	None	None	LSTM within policy network	LSTM-based policy	Learned policy only
Prediction Accuracy	90–95%	N/A	N/A	N/A	85–90%	Environment-dependent	Variable
Computational Load	Low–Medium	Low (static) / High (dynamic)	Medium	Medium	High (training)	High (training)	Very High (training)
Adaptability to Dynamics	High (proactive)	Low (reactive)	Good (updates)	Moderate	High (policy-learned)	High (but brittle)	Task-dependent
Visual Input Dependency	No	No	No	No	No	Optional	Often Required
Robustness to Noise	High (Kalman-enhanced)	Low	Moderate	Moderate	Moderate	Moderate	Domain-limited
Optimality Guarantee	Near-Optimal	Optimal (static only)	Near-Optimal	Suboptimal (bounded)	Near-Optimal	Policy-driven	Policy-driven
Training Requirements	Moderate (LSTM only)	None	None	None	High	Very High	Very High
Scalability	High	Moderate (grid-based)	Moderate	Moderate	Moderate	Moderate	Task-specific
Theoretical Guarantees	Yes (admissibility, completeness)	Yes	Yes	Yes	Not explicit	No	No

In robotics, the combination of LSTM with RRT* has enabled considerable progress in multi-agent coordination, obstacle avoidance, and adaptive trajectory generation. Approaches such as MADDPG-LSTM [8] use recurrent neural networks to model temporal dependencies among agents, improving policy learning in collaborative settings. Similarly, LSTM-based predictors trained on LiDAR data have been applied to mobile robots for dynamic obstacle avoidance, demonstrating high validation accuracy in real-world-like environments [9]. Hybrid methods that combine LSTM with control-theoretic frameworks like Model Predictive Contouring Control (MPCC) have also been proposed to reduce computational costs in constrained, industrial domains [10]. However, despite their effectiveness, these models often rely on black-box reinforcement learning, are sensitive to environmental shifts, and lack theoretical guarantees such as admissibility or bounded suboptimality.

In this context, the present work introduces a novel LSTM-augmented A* framework that bridges the gap between deterministic planning and predictive modeling. Unlike sampling-biased RRT*-based variants or policy-optimized reinforcement learning methods such as SAC-LSTM [11], our approach integrates an LSTM-Kalman hybrid predictor directly into the A* heuristic function. This design facilitates real-time, foresight-driven planning while retaining the theoretical foundations of classical graph search. It avoids the overhead of policy retraining, reduces the frequency of re-planning by anticipating obstacle motion, and supports path smoothness via post-processing with safety constraints. Crucially, this method is validated not only on synthetic datasets but also on complex benchmarks such as `Berlin_0_256`, where it consistently delivers high prediction accuracy, collision-free trajectories, and low computational latency.

This paper makes the following core contributions: (1) it proposes a novel integration of LSTM-based trajectory predictions into the cost function of the A* algorithm, enhancing proactive responsiveness in dynamic environments; (2) it introduces a hybrid prediction model using Kalman filtering to improve robustness against noisy and uncertain measurements; and (3) it empirically validates the approach across synthetic and benchmark scenarios, demonstrating improvements in planning efficiency, safety, and real-time applicability.

To contextualize our contribution within the broader research landscape, Table 1 presents a comparative analysis of the LSTM-Augmented A* approach against established methodologies, showing its superior balance of computational efficiency, prediction accuracy, robustness, and theoretical rigor across a range of evaluation metrics. The remainder of this paper is structured as follows: Section 2 presents an overview of the environment modeling and map generation process. Section 3 describes the LSTM network architecture and its application in dynamic obstacle prediction. Section 4 introduces the hybrid path planning framework, detailing the integration of LSTM predictions with A*. Section 5 discusses the experimental setup, evaluation metrics, and simulation results. Section 6 validates the framework on benchmark maps and real-world scenarios, while Section 7 concludes the paper and outlines future research directions.

2. Environment modeling and map generation

The generation of static and dynamic obstacle maps is a critical component of this study, as it enables the simulation of realistic environments for path planning. Static obstacles are generated using Poisson Disk Sampling, a well-established algorithm in computational geometry that ensures a uniform spatial distribution of obstacles while maintaining a minimum distance between them. Introduced by Bridson [15], this algorithm works by iteratively placing points within the map space, ensuring that each new point adheres to a minimum distance constraint $d_{\min}$ from all previously placed points. The process begins with the selection of a random initial point $\mathbf{p}_0 = (x_0, y_0)$ within the map boundaries, where $x_0 \in [1, W]$ and $y_0 \in [1, H]$, with W and H representing the width and height of the map, respectively. For each existing point $\mathbf{p}_i$, new candidate points $\mathbf{p}_{\text{candidate}}$ are generated within an annulus of radius $[d_{\min}, 2d_{\min}]$ around $\mathbf{p}_i$. The candidate points are sampled uniformly in polar coordinates as follows:

$$\mathbf{p}_{\text{candidate}} = \mathbf{p}_i + r \cdot (\cos\theta, \sin\theta), \tag{1}$$

where $r \sim U(d_{\min}, 2d_{\min})$ and $\theta \sim U(0, 2\pi)$. A candidate point $\mathbf{p}_{\text{candidate}}$ is accepted if it satisfies the minimum distance constraint with all existing points:

$$|\mathbf{p}_{\text{candidate}} - \mathbf{p}_j| \geq d_{\min} \quad \forall j \in \{1, 2, \ldots, i\}. \tag{2}$$

This ensures a spatially balanced distribution of static obstacles, essential for simulating realistic environments. Dynamic obstacles are introduced to model moving entities within the environment. Their initial positions are generated using a randomized placement algorithm that ensures no static or dynamic obstacles overlap. The movement of dynamic obstacles is simulated over a series of snapshots, with each ob-

Table 2
Network architecture of the LSTM model.

Layer	Type	Activations	Learnable Parameters	States
1	Sequence Input	$4(C) \times 1(B) \times 1(T)$	-	-
2	LSTM	$128(C) \times 1(B)$	Input Weights: 512×4 Recurrent Weights: 512×128 Bias: 512×1	Hidden State: 128×1 Cell State: 128×1
3	Fully Connected	$2(C) \times 1(B)$	Weights: 2×128 Bias: 2×1	-
4	Regression Output	$2(C) \times 1(B)$	-	-

stacle's position updated within a predefined movement range $[-1, 1]$ in both the x and y directions. The position update for a dynamic obstacle at time t is given by:

$$\mathbf{p}_t = \mathbf{p}_{t-1} + \Delta\mathbf{p}, \tag{3}$$

where $\Delta\mathbf{p} = (\Delta x, \Delta y)$ and $\Delta x, \Delta y \sim U(-1, 1)$. Collision detection is implemented to prevent dynamic obstacles from overlapping with static obstacles, ensuring the physical plausibility of the generated maps. The number of dynamic obstacles increases incrementally over time, simulating the gradual introduction of new moving entities into the environment.

The generated maps are stored as 3D matrices $\mathbf{M} \in \{0, 1\}^{W \times H \times T}$, where each slice $\mathbf{M}_{:,:,t}$ represents a snapshot of the environment at time step t. Key metrics, such as free space percentage, obstacle density, and the average distance between static obstacles, are computed to quantify the characteristics of the generated maps. These metrics are defined as follows:

$$\text{Free Space Percentage} = 100 \cdot \left(1 - \frac{\sum_{i,j,t} \mathbf{M}_{i,j,t}}{W \cdot H \cdot T}\right), \tag{4}$$

$$\text{Obstacle Density} = \frac{N_{\text{static}} + N_{\text{dynamic}}}{W \cdot H}, \tag{5}$$

where N_{static} and N_{dynamic} are the numbers of static and dynamic obstacles, respectively. Additionally, the average distance between static obstacles is calculated as:

$$\text{Average Distance} = \frac{1}{N_{\text{static}}^2} \sum_{i=1}^{N_{\text{dynamic}}} \sum_{j=1}^{N_{\text{static}}} |\mathbf{p}_i - \mathbf{p}_j|. \tag{6}$$

This methodology combines Poisson Disk Sampling for static obstacle generation with a dynamic obstacle model that integrates collision detection and incremental obstacle introduction. The Poisson Disk Sampling ensures a spatially uniform distribution of static obstacles, while the dynamic model introduces temporal variability by simulating moving entities with realistic constraints. This dual approach enhances the realism of the simulated environments and provides a versatile platform for testing algorithmic performance under diverse and dynamic conditions. The generated maps are visualized in six snapshots, as shown in Fig. 1. Red squares represent static obstacles, while dynamic obstacles are depicted as blue squares. Clearance areas around the obstacles are indicated by dashed lines, with light red for static and light blue for dynamic obstacles. The block diagram in Fig. 2 visually represents the flow of information and the various stages of the proposed framework. The process begins with the input of a dynamic map, which is processed and features are extracted. The LSTM network then predicts the future positions of dynamic obstacles, providing valuable temporal context for the path planning algorithm. These predictions are seamlessly integrated into the A* algorithm, augmenting its heuristic search process. The A* algorithm uses the predicted obstacle positions to adjust its search strategy in real-time, effectively steering the path planning away from potential collisions. This enhanced approach significantly reduces the need for constant re-planning and computationally expensive recalculations, improving both the efficiency and effectiveness of the path planning process. The final output is an optimal path that avoids both static and predicted dynamic obstacles, ensuring safe navigation through complex and dynamic environments.

3. LSTM networks for dynamic obstacle prediction

Long Short-Term Memory (LSTM) networks are specialized Recurrent Neural Networks (RNNs) designed to model sequential data by capturing long-term dependencies. Traditional RNNs suffer from the vanishing gradient problem, which limits their ability to learn long-range dependencies in sequences. LSTMs address this limitation through a gating mechanism that regulates the flow of information, making them particularly effective in tasks such as time series prediction, natural language processing, and trajectory forecasting.

The LSTM cell operates through a series of equations that define its gating mechanisms and state updates. The forget gate, which determines the extent to which information from the previous cell state is retained, is computed as:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f), \tag{7}$$

where $\mathbf{f}_t$ is the forget gate activation vector, σ is the sigmoid function, $\mathbf{W}_f$ and $\mathbf{b}_f$ are learnable weights and biases, $\mathbf{h}_{t-1}$ is the previous hidden state, and $\mathbf{x}_t$ is the current input. The input gate, which controls the update of the cell state, is given by:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i), \tag{8}$$

where $\mathbf{i}_t$ is the input gate activation vector. The candidate cell state, which represents potential updates to the cell state, is computed as:

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c), \tag{9}$$

where $\tilde{\mathbf{c}}_t$ is the candidate cell state. The cell state is then updated as:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{10}$$

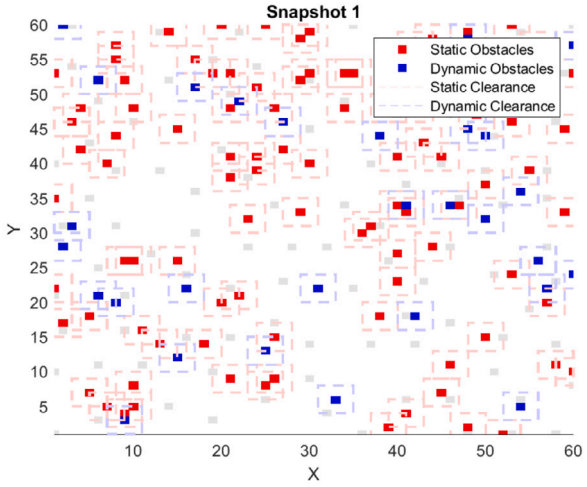
where $\mathbf{c}_t$ is the updated cell state, and $\odot$ denotes element-wise multiplication. The output gate, which determines the hidden state output, is computed as:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o), \tag{11}$$

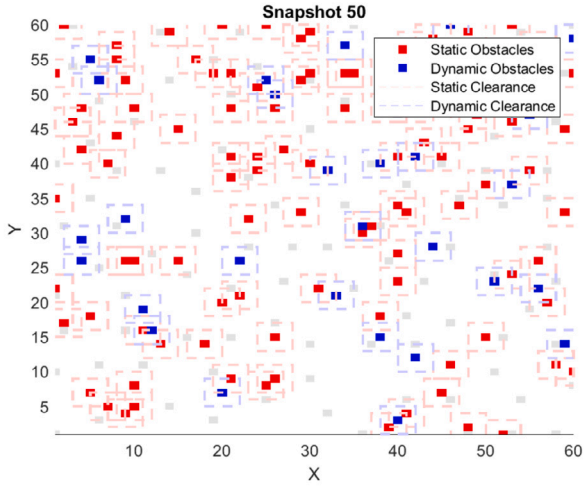
where $\mathbf{o}_t$ is the output gate activation vector. Finally, the hidden state is updated as:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{12}$$

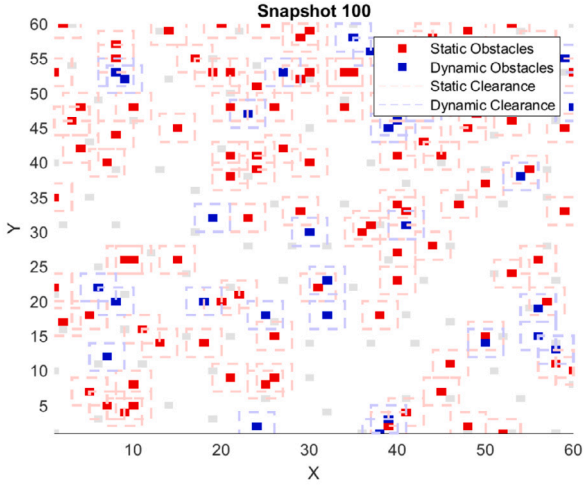
where $\mathbf{h}_t$ is the current hidden state. The architecture of the LSTM network used in this study is summarized in Table 2. The network consists of four main layers: a sequence input layer, an LSTM layer with 128 hidden units, a fully connected layer, and a regression output layer. The sequence input layer accepts input data with four dimensions, while the LSTM layer captures temporal dependencies using gating mechanisms. The fully connected layer maps the LSTM outputs to 2 dimensions, representing the predicted obstacle positions. Finally, the regression output layer computes the mean squared error (MSE) loss during training. This architecture is designed to model sequential data effectively, making it suitable for predicting dynamic obstacle trajectories in real-time path



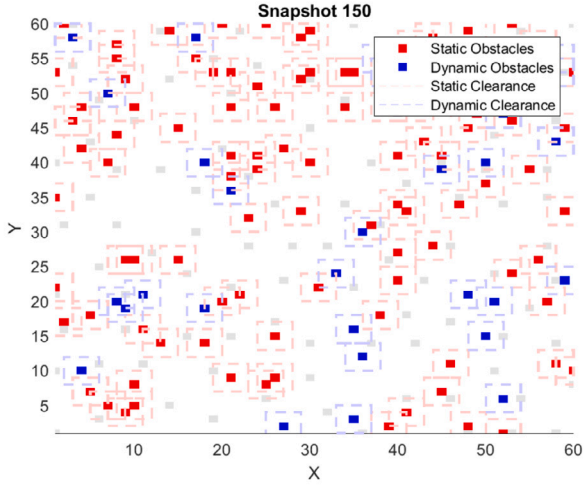
(a) Snapshot 1



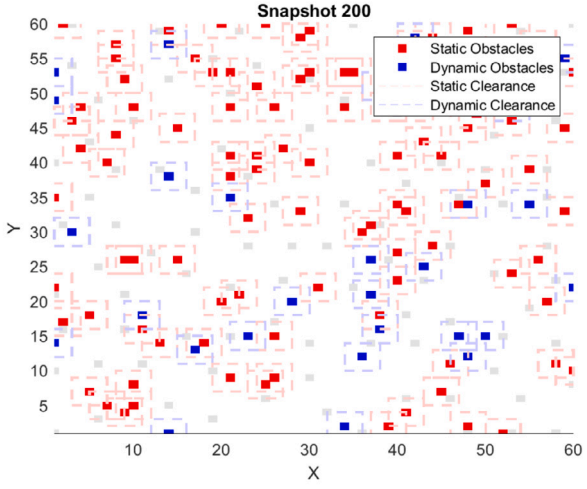
(b) Snapshot 50



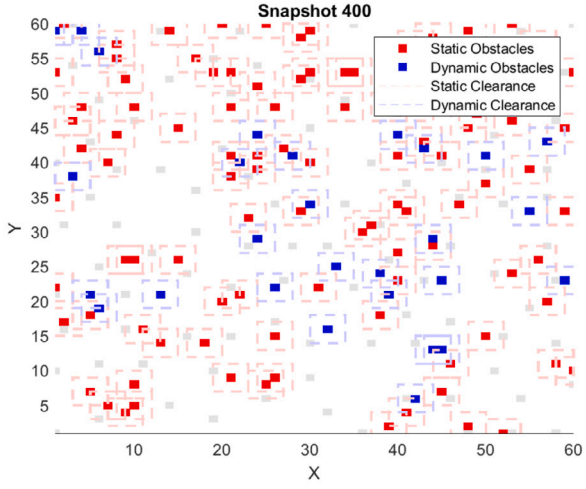
(c) Snapshot 100



(d) Snapshot 150



(e) Snapshot 200



(f) Snapshot 400

Fig. 1. Six snapshots of the generated maps showing static and dynamic obstacles with clearance areas. Static obstacles are represented by red squares, dynamic obstacles by blue squares, and clearance areas by dashed lines (light red for static and light blue for dynamic).

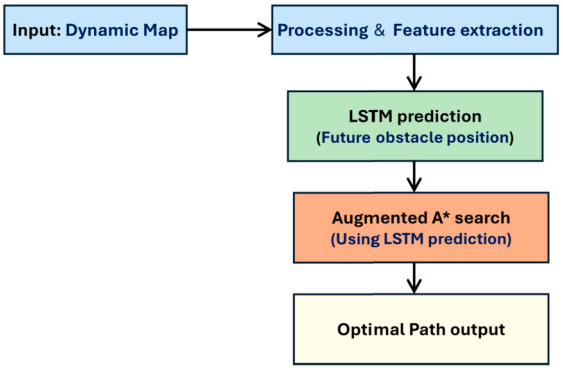


Fig. 2. Block diagram illustrating the integration of LSTM predictions with the A* algorithm.

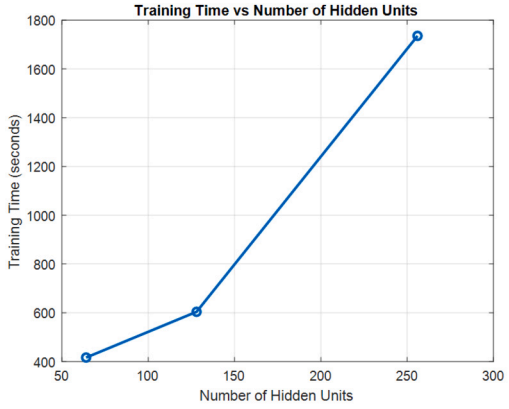


Fig. 3. Training time as a function of the number of LSTM hidden units.

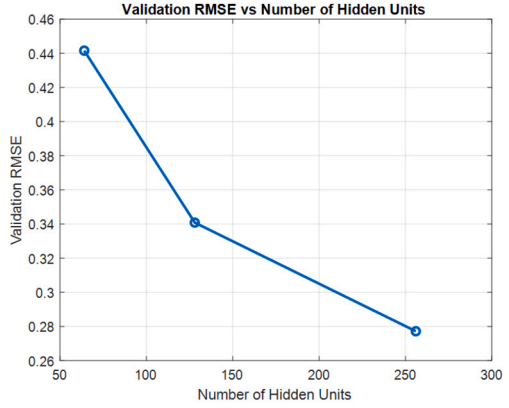


Fig. 4. Validation RMSE versus the number of hidden units.

planning applications [16]. The LSTM architecture and training setup are summarized in Table 4.

An ablation study was conducted by varying the number of hidden units (64, 128, 256) to assess the trade-off between predictive accuracy and computational cost. As shown in Fig. 3, training time increases substantially with larger network sizes. Meanwhile, Fig. 4 demonstrates that although increasing hidden units improves accuracy (lower RMSE), the marginal gain beyond 128 units is not justified by the computational overhead. Therefore, the 128-unit configuration was selected as the optimal balance for real-time deployment.

The model is trained using the Adam optimizer with a learning rate of 0.001, a mini-batch size of 32, and a maximum of 100 epochs. Training data is preprocessed to remove outliers and normalized to have zero mean and unit variance. The LSTM model is evaluated using standard

Table 3
Evaluation metrics for the LSTM model on training and testing data.

Metric	Training Data	Testing Data
Mean Squared Error (MSE)	0.0024	0.0024
Root Mean Squared Error (RMSE)	0.0493	0.0492
Mean Absolute Error (MAE)	0.0421	0.0420
R-squared (R^2)	0.9976	0.9976

Table 4
Summary of LSTM model structure, training parameters, and evaluation metrics.

Parameter	Value/Description
Architecture	
Input Layer	Sequence input layer with size matching feature dimensions
LSTM Layer	1 hidden layer with 128 units, output mode: 'last'
Fully Connected Layer	Maps LSTM outputs to predicted obstacle positions
Output Layer	Regression layer for continuous outputs
Training Parameters	
Optimizer	Adam
Learning Rate	0.001
Mini-Batch Size	32
Max Epochs	100
Shuffle	Every epoch
Gradient Threshold	1
Data Preprocessing	
Lookback Window	5 time steps
Training Data Ratio	80%
Outlier Removal	IQR-based filtering
Normalization	Zero mean and unit variance
Evaluation Metrics	
Mean Squared Error (MSE)	$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
Root Mean Squared Error (RMSE)	$\mathrm{RMSE} = \sqrt{\mathrm{MSE}}$
Mean Absolute Error (MAE)	$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$
R-squared (R^2)	$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$

regression metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2). These metrics collectively measure the accuracy of the predicted trajectories.

The trained LSTM model is integrated with the A* path planning algorithm to provide real-time predictions of dynamic obstacle positions. The performance of the LSTM network is summarized in Table 3. The model achieved an MSE of 0.0024 and an RMSE of 0.0493 on the training data, indicating high accuracy in predicting obstacle trajectories. The model demonstrated consistent performance on the testing data with an MSE of 0.0024 and an RMSE of 0.0492. The MAE values of 0.0421 (training) and 0.0420 (testing) further confirm the model's precision. Additionally, the R^2 values of 0.9976 for both datasets highlight the model's strong ability to explain the variance in the data.

Fig. 5 depicts the distribution of prediction errors produced by the trained LSTM network on the testing dataset. The horizontal axis represents the magnitude of the prediction error (the absolute difference between the predicted and ground-truth positions). In contrast, the vertical axis indicates how frequently each error value occurs across all test samples. The unimodal shape of the histogram, with a peak near 0.06, shows that most predictions lie within a small error band. Only a tiny fraction of samples exhibit errors greater than 0.10, indicating that the LSTM network consistently achieves accurate predictions for most test instances and rarely produces large deviations. This distribution underscores the reliability of the trained model in forecasting obstacle trajectories for dynamic path planning.

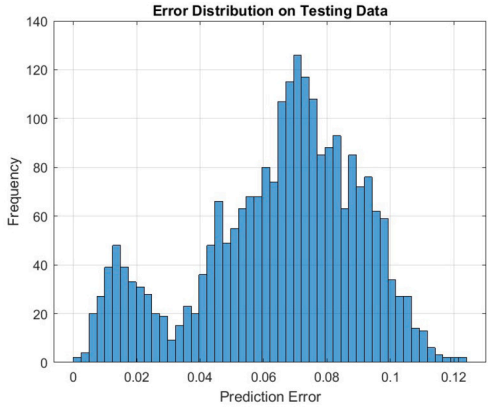


Fig. 5. Error distribution of the trained LSTM network on the testing dataset.

4. Hybrid path planning: LSTM-augmented A*

Integrating Long Short-Term Memory (LSTM) networks with the A* algorithm provides a robust framework for real-time path planning in dynamic environments. The A* algorithm is a heuristic search method to determine the shortest path from a graph's start node **s** to a goal node **g**. The graph comprises nodes representing states (for example, positions on a map) and edges representing transitions between states with associated costs. The algorithm evaluates nodes using a cost function $\mathbf{f}(\mathbf{n})$, which is defined as:

$$\mathbf{f}(\mathbf{n}) = \mathbf{g}(\mathbf{n}) + \mathbf{h}(\mathbf{n}), \tag{13}$$

where $\mathbf{g}(\mathbf{n})$ is the actual cost incurred to reach node **n** from the start node **s**, and $\mathbf{h}(\mathbf{n})$ is the heuristic estimate of the cost from node **n** to the goal node **g**. The heuristic $\mathbf{h}(\mathbf{n})$ must be admissible, meaning it never overestimates the true cost and consistently satisfies the triangle inequality. A common choice for the heuristic is the Euclidean distance:

$$\mathbf{h}(\mathbf{n}) = \sqrt{(\mathbf{x_n} - \mathbf{x_g})^2 + (\mathbf{y_n} - \mathbf{y_g})^2}, \tag{14}$$

where $(\mathbf{x_n}, \mathbf{y_n})$ and $(\mathbf{x_g}, \mathbf{y_g})$ denote the coordinates of node **n** and the goal **g**, respectively. A priority queue is employed in the A* algorithm to manage and retrieve nodes during the search process efficiently. This queue maintains nodes along with their associated $\mathbf{f}(\mathbf{n})$ values so that the node with the lowest $\mathbf{f}(\mathbf{n})$ is processed first. The efficient operation of the algorithm is predicated on this mechanism since it ensures that the most promising nodes are expanded first, thereby reducing the number of unnecessary computations. Initially, the algorithm inserts the start node **s** into the priority queue, setting

$$\mathbf{g}(\mathbf{s}) = \mathbf{0} \quad \text{and} \quad \mathbf{f}(\mathbf{s}) = \mathbf{h}(\mathbf{s}). \tag{15}$$

Then, while the priority queue is not empty, the algorithm retrieves and removes the node **n** with the lowest $\mathbf{f}(\mathbf{n})$. The algorithm terminates and returns the computed path if **n** equals the goal node **g**. Otherwise, the node **n** is moved from the open set to the closed set, and for each neighbor **m** of **n**, the algorithm computes

$$\mathbf{g}(\mathbf{m}) = \mathbf{g}(\mathbf{n}) + \mathbf{c}(\mathbf{n}, \mathbf{m}), \tag{16}$$

where $\mathbf{c}(\mathbf{n}, \mathbf{m})$ represents the cost of moving from node **n** to node **m**. If node **m** is either not in the open set or if the newly computed $\mathbf{g}(\mathbf{m})$ is lower than its previous value, the values of $\mathbf{g}(\mathbf{m})$ and

$$\mathbf{f}(\mathbf{m}) = \mathbf{g}(\mathbf{m}) + \mathbf{h}(\mathbf{m}) \tag{17}$$

are updated, and **m** is inserted into the priority queue. This procedure ensures that when the goal node is reached, it is reached via the optimal path, assuming the admissibility of the heuristic. The standard A* algorithm faces two significant challenges in dynamic environments where obstacles may be moving. The first challenge is the need for continuous re-planning at each time step, as the previously computed path may become invalid when obstacles move. This constant need for re-computation introduces a significant computational overhead, rendering the standard A* algorithm less suitable for real-time applications. The second challenge is recalculating the heuristic function $\mathbf{h}(\mathbf{n})$ to account for updated positions of dynamic obstacles, which is computationally expensive and further intensifies the re-planning burden. To address these challenges, the standard A* algorithm is enhanced by integrating predictive capabilities provided by an LSTM network. The LSTM is trained on historical trajectories of obstacles to predict their future positions. Given a sequence of past obstacle positions over a lookback window of length **l**:

$$\mathbf{X}_t = [\mathbf{x}_{t-l}, \mathbf{x}_{t-l+1}, \ldots, \mathbf{x}_{t-1}], \tag{18}$$

the network predicts the obstacle's future position $\mathbf{y}_t$ at time t:

$$\mathbf{y}_t = \mathrm{LSTM}(\mathbf{X}_t; \mathbf{W}), \tag{19}$$

where $\mathbf{W}$ represents the trained weights of the LSTM network. Input data are normalized using the mean $\boldsymbol{\mu}_X$ and standard deviation $\boldsymbol{\sigma}_X$ derived from the training data:

$$\mathbf{X}_t^{\mathrm{norm}} = \frac{\mathbf{X}_t - \boldsymbol{\mu}_X}{\boldsymbol{\sigma}_X}. \tag{20}$$

The predicted outputs are then denormalized to their original scale using the mean $\boldsymbol{\mu}_y$ and standard deviation $\boldsymbol{\sigma}_y$ of the target values:

$$\mathbf{y}_t = \mathrm{LSTM}(\mathbf{X}_t^{\mathrm{norm}}) \cdot \boldsymbol{\sigma}_y + \boldsymbol{\mu}_y. \tag{21}$$

To integrate the LSTM predictions into the A* algorithm, the heuristic function $\mathbf{h}(\mathbf{n})$ is modified to account for predicted obstacle positions:

$$\mathbf{h}(\mathbf{n}) = \mathbf{h}_{\mathrm{base}}(\mathbf{n}) + \lambda \sum_i \max\left(0, d_{\mathrm{clear}} - \|\mathbf{n} - \mathbf{o}_i\|\right), \tag{22}$$

where $\mathbf{h}_{\mathrm{base}}(\mathbf{n})$ (e.g., Euclidean distance) is the original heuristic function, λ is a weighting factor, d_{clear} is a predefined safety distance, and $\mathbf{o}_i$ denotes the predicted position of obstacle i. This predictive approach enhances the A* algorithm's suitability for dynamic environments, significantly reducing computational overhead by mitigating frequent replanning due to moving obstacles.

To address these challenges, the A* algorithm is enhanced by integrating an LSTM network that predicts future obstacle positions. After obtaining the initial path, smoothing is applied to reduce unnecessary deviations. The smoothed path, $\mathbf{P}_{\mathrm{smooth}}$, is derived by minimizing the objective function:

$$\mathcal{L}(\mathbf{P}) = \alpha \sum_{i=1}^{M} \|\mathbf{p}_i - \mathbf{p}_i^{\mathrm{orig}}\|^2 + \beta \sum_{i=2}^{M-1} \|\mathbf{p}_{i+1} - 2\mathbf{p}_i + \mathbf{p}_{i-1}\|^2, \tag{23}$$

where $\mathbf{p}_i$ is the i-th point on the smoothed path, $\mathbf{p}_i^{\mathrm{orig}}$ is its corresponding point on the original path, α and β are weighting factors, and M is the total number of path points. To ensure safety, the smoothed path must satisfy the clearance constraint:

$$\|\mathbf{p}_i - \mathbf{o}_j\| \geq d_{\mathrm{clear}}, \quad \forall i, j, \tag{24}$$

where $\mathbf{o}_j$ represents the position of obstacle j, and d_{clear} is the minimum clearance distance. The integrated system's performance is evaluated through several metrics. First, the *Path Length* measures the total length of the generated path:

$$L = \sum_{i=1}^{M-1} \|\mathbf{p}_{i+1} - \mathbf{p}_i\|. \tag{25}$$

The second metric is the *Computation Time*, representing the time required to compute the path. The third metric is the *Clearance*, defined as the minimal distance between the path and any obstacle:

Table 5
Simulation results for different start and goal points.

Scenario	Start Point	Goal Point	Path Length (units)	Number of Steps	Computation Time (s)
1	[X: 2.00, Y: 4.00]	[X: 58.00, Y: 56.00]	78.58	63	1.2568
2	[X: 57.00, Y: 57.00]	[X: 3.00, Y: 4.00]	78.08	66	1.6765
3	[X: 24.00, Y: 2.00]	[X: 21.00, Y: 56.00]	59.54	57	1.0003
4	[X: 60.00, Y: 3.00]	[X: 3.00, Y: 57.00]	87.12	76	2.9582
5	[X: 1.00, Y: 29.00]	[X: 59.00, Y: 37.00]	64.63	63	1.0237
6	[X: 41.00, Y: 9.00]	[X: 3.00, Y: 57.00]	64.63	55	0.9427

$$d_{\min} = \min_{i,j} \|\mathbf{p}_i - \mathbf{o}_j\|. \tag{26}$$

Finally, the system's safety is quantified using the *Collision Rate*, the percentage of path points that violate the clearance threshold d_{clear}:

$$\text{Collision (\%)} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}\left(\min_j \|\mathbf{p}_i - \mathbf{o}_j\| < d_{\text{clear}} \right) \times 100\%, \tag{27}$$

where $\mathbb{I}(\cdot)$ is the indicator function.

5. Theoretical guarantees of LSTM-augmented A*

The integration of LSTM predictions with the A* algorithm preserves three fundamental properties: *completeness*, *admissibility*, and *optimality*. Crucially, this integration maintains the original graph structure and search space connectivity, ensuring that:

- **Completeness** is preserved: If a valid path exists from start node **s** to goal node **g**, the algorithm will find it.
- **Optimality** is maintained when the heuristic remains admissible.

5.1. Admissibility condition

The modified heuristic combines an admissible base heuristic with a safety penalty term:

$$\mathbf{h}(\mathbf{n}) = \underbrace{\mathbf{h}_{\text{base}}(\mathbf{n})}_{\substack{\text{Admissible} \\ \text{base heuristic}}} + \underbrace{\lambda \sum_{i=1}^{N} \max(0, d_{\text{clear}} - \|\mathbf{n} - \mathbf{o}_i\|)}_{\substack{\text{Safety} \\ \text{penalty term}}} \tag{28}$$

The heuristic in Eq. (28) remains admissible if:

$$\lambda \sum_{i=1}^{N} \max(0, d_{\text{clear}} - \|\mathbf{n} - \mathbf{o}_i\|) \leq \Delta(\mathbf{n}) \quad \forall \mathbf{n} \in \mathcal{G} \tag{29}$$

where:

- $\mathcal{G}$ is the search graph containing all possible nodes **n**
- $\Delta(\mathbf{n}) := \mathbf{h}^*(\mathbf{n}) - \mathbf{h}_{\text{base}}(\mathbf{n})$ is the *heuristic slack*, representing the maximum allowable penalty at node **n**
- The inequality ensures the total obstacle penalty never exceeds the admissible margin

where $\Delta(\mathbf{n}) = \mathbf{h}^*(\mathbf{n}) - \mathbf{h}_{\text{base}}(\mathbf{n})$ is the heuristic slack.

The admissibility follows from three properties:

1. **Base admissibility**: $\mathbf{h}_{\text{base}}(\mathbf{n}) \leq \mathbf{h}^*(\mathbf{n})$ by definition of admissible heuristics.
2. **Non-negative penalty**: $\max(0, d_{\text{clear}} - \|\mathbf{n} - \mathbf{o}_i\|) \geq 0$ ensures $\mathbf{h}(\mathbf{n}) \geq \mathbf{h}_{\text{base}}(\mathbf{n})$.
3. **Bounded modification**: When Eq. (29) holds, the total penalty never exceeds the available slack $\Delta(\mathbf{n})$, thus $\mathbf{h}(\mathbf{n}) \leq \mathbf{h}^*(\mathbf{n})$.

5.2. Practical considerations

5.2.1. Parameter selection

- **Discrete grids**: With minimum edge cost $c_{\min} = 1$, choosing $\lambda \leq (N \cdot d_{\text{clear}})^{-1}$ guarantees admissibility.
- **Continuous spaces**: $\Delta(\mathbf{n})$ can be bounded through:
  - Worst-case analysis of obstacle density
  - Empirical estimation from training data

5.2.2. Safety-optimality trade-off

In dynamic environments, we may intentionally violate Eq. (29) to prioritize safety. This leads to:

- **Bounded suboptimality** (controlled by λ), following ϵ-admissible approaches [5]
- **Improved collision avoidance**, consistent with safety-critical planning principles [1]

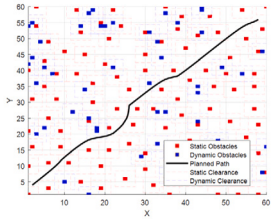
This trade-off is justified by our experimental results (Section 5), which demonstrate that small violations of admissibility yield substantially safer paths with minimal length increases.

6. Simulation-based evaluation

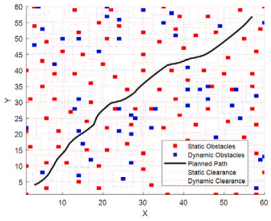
The LSTM-augmented A* algorithm was assessed through simulations conducted in a dynamic 60×60 grid environment populated with static and dynamic obstacles. While static obstacles remained fixed, dynamic obstacles followed diverse, non-deterministic trajectories. The LSTM model was trained on sequences capturing both position and velocity information, enabling more accurate forecasting of obstacle motion patterns. These predictions were integrated into the A* planning heuristic to guide pathfinding.

Simulations encompassed six distinct scenarios with varying start and goal positions. Evaluation focused on three metrics: *Path Length*, *Number of Steps*, and *Computation Time*. Results presented in Table 5 confirm that the framework consistently generates efficient, collision-free paths. Visualizations in Figs. 6a–6f further illustrate its capacity to adapt to complex, dynamic environments. Although experiments were conducted within a fixed-size grid, the modular architecture and localized nature of prediction and planning allow for straightforward scalability to larger maps. Future deployment on extended environments will involve tiling or regional partitioning strategies, ensuring that the algorithm's real-time performance remains tractable even as spatial complexity increases.

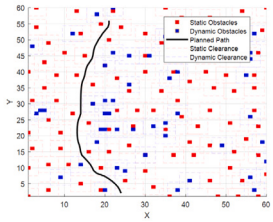
To illustrate the effectiveness of the LSTM-augmented A* algorithm, Figs. 6a through 6f show planned paths across multiple dynamic scenarios. Additionally, Figs.6a through 10 visualize the algorithm's real-time responses to moving obstacles. These snapshots demonstrate how the algorithm proactively avoids collisions without requiring continuous replanning, effectively using LSTM-based predictions to anticipate obstacle trajectories. This predictive approach significantly reduces computational overhead and enhances the reliability and robustness of real-time path planning in dynamic environments.



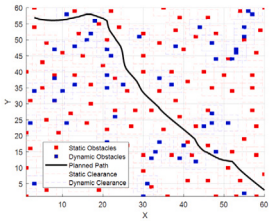
(a) Path for Scenario 1: Start [2.00, 4.00], Goal [58.00, 56.00].



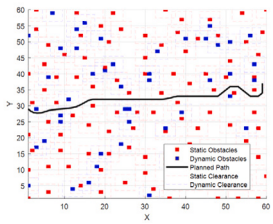
(b) Path for Scenario 2: Start [57.00, 57.00], Goal [3.00, 4.00].



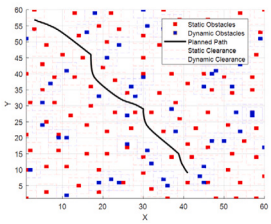
(c) Path for Scenario 3: Start [24.00, 2.00], Goal [21.00, 56.00].



(d) Path for Scenario 4: Start [60.00, 3.00], Goal [3.00, 57.00].



(e) Path for Scenario 5: Start [1.00, 29.00], Goal [59.00, 37.00].



(f) Path for Scenario 6: Start [41.00, 9.00], Goal [3.00, 57.00].

Fig. 6. Visualization of planned paths for the six simulation scenarios.

7. Evaluation on realistic benchmarks and dynamic scenarios

Benchmark-based evaluations were conducted to assess the proposed LSTM-Kalman-A* framework under realistic conditions. These experiments use the publicly available `Berlin_0_256.map` dataset [17], which presents a highly non-uniform and obstacle-rich layout. Unlike earlier synthetic trajectories, this benchmark scenario provides a closer approximation to practical deployment challenges, including partial observations, occlusions, and abrupt motion behavior.

To enhance the model's robustness under uncertainty, particularly in scenarios involving occlusions or sensor degradation, the LSTM network was redesigned to process sequences that include both positional and velocity information. This extension equips the model with greater sensitivity to dynamic motion patterns, allowing it to anticipate trajectory evolution more effectively and improving its adaptability to abrupt or nonlinear changes in obstacle behavior.

The LSTM implementation integrates a BiLSTM network trained on sequences enriched with both position and velocity components. This structure enables the model to capture temporal coherence and directional motion dynamics. Kalman filtering is also incorporated through a recursive estimation process that corrects predicted positions based on uncertainty-aware fusion. Unlike prior static-weight fusion, the hybrid output now reflects dynamic weighting proportional to the inverse variance of prediction confidence. Let the environment state at time t be defined as:

$$\mathbf{S}(t) = (\mathbf{X}(t), \mathbf{O}(t), \mathbf{G}), \tag{30}$$

where $\mathbf{X}(t) \in \mathbb{R}^2$ denotes the agent position, $\mathbf{G}$ the goal, and $\mathbf{O}(t) = \{\mathbf{o}_1(t), \ldots, \mathbf{o}_n(t)\}$ the set of dynamic obstacles, each given by:

$$\mathbf{o}_i(t) = \begin{bmatrix} \mathbf{p}_i(t) \\ \mathbf{v}_i(t) \end{bmatrix}. \tag{31}$$

Predictions from the BiLSTM are computed as:

$$\boldsymbol{\phi}_l(t+1) = f\left(\mathbf{W}_l \cdot \mathbf{O}(t-k:t) + \mathbf{b}_l\right), \tag{32}$$

while the Kalman Filter provides:

$$\hat{\mathbf{x}}(t+1) = \mathbf{F}\hat{\mathbf{x}}(t) + \mathbf{K}(t)\left[\mathbf{z}(t) - \mathbf{H}\hat{\mathbf{x}}(t)\right], \tag{33}$$

with gain matrix and innovation covariance given by:

$$\mathbf{S}(t) = \mathbf{H}\mathbf{P}(t)\mathbf{H}^T + \mathbf{R}, \tag{34}$$

$$\mathbf{K}(t) = \mathbf{P}(t)\mathbf{H}^T\mathbf{S}(t)^{-1}. \tag{35}$$

The combined prediction uses a confidence-aware fusion:

$$\mathbf{P}(t+1) = \alpha(t)\boldsymbol{\phi}_l(t+1) + (1-\alpha(t))\hat{\mathbf{x}}(t+1), \tag{36}$$

with adaptive weight:

$$\alpha(t) = \sigma\left(\frac{\|\boldsymbol{\phi}_l(t) - \mathbf{z}(t)\|_2}{\|\hat{\mathbf{x}}(t) - \mathbf{z}(t)\|_2}\right). \tag{37}$$

The planning cost combines distance, heuristic, and risk:

$$J(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x}) + \rho(\mathbf{x}), \tag{38}$$

$$\rho(\mathbf{x}) = \sum_{i=1}^{n} \exp\left(-\frac{\|\mathbf{x} - \mathbf{P}(o_i(t))\|^2}{2\sigma^2}\right), \tag{39}$$

where $\mathbf{P}(o_i(t))$ denotes the predicted position of obstacle i at time t. Safety constraints are defined by:

$$\|\mathbf{X}(t) - \mathbf{o}_i(t)\|_2 \geq \delta(\|\mathbf{v}_i(t)\|_2), \tag{40}$$

$$\delta(v) = \delta_0 + 2\kappa|v|, \tag{41}$$

with optimal path minimizing:

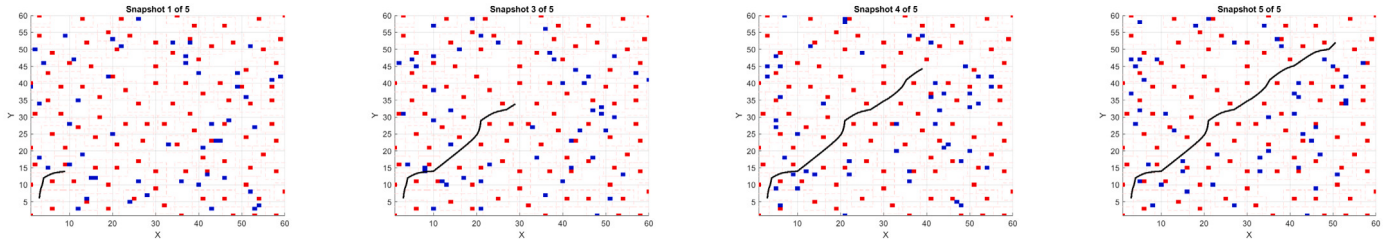
$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}} \int_0^T \left[J(\boldsymbol{\pi}(t)) + \lambda\|\ddot{\boldsymbol{\pi}}(t)\|^2\right] dt. \tag{42}$$

Quantitative results are summarized in Table 6, measuring trajectory efficiency, safety, smoothness, and prediction error. Visualizations in Fig. 18 show how planned paths adapt in real-time to varying obstacle behaviors.

Despite being validated on standard-sized maps, the modular design of the LSTM-Kalman-A* framework inherently supports scalability. Since the prediction module operates locally on sequences of obstacle states and the A* planner uses standard graph-based expansion, applying this framework to larger grid maps requires no retraining. The prediction and fusion stages remain computationally bounded per obstacle, and the path planning complexity scales predictably with map size. Therefore, the framework is suitable for deployment in expanded environments, contingent on appropriate graph resolution and hardware support.

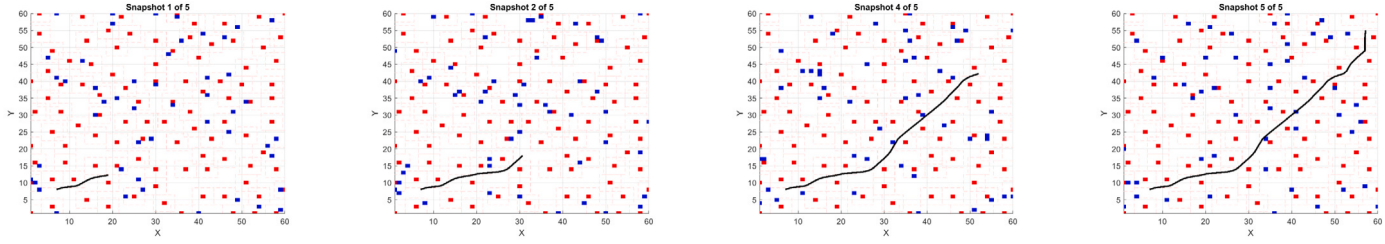
7.1. Robustness and sensitivity analysis

To rigorously evaluate the reliability of the proposed LSTM-Kalman-A* framework under sensor degradation and environmental uncertainty, we conducted 150 trials across 25 scenarios by varying Gaussian noise levels (0–5) and missing data rates (0%–40%). The system achieved a strong mean success rate of 65.6% (± 15.7%) and a low average prediction error of 0.328 (± 0.123), as summarized in Table 9. Under ideal conditions (0 noise, 0% missing), the framework achieved a peak success



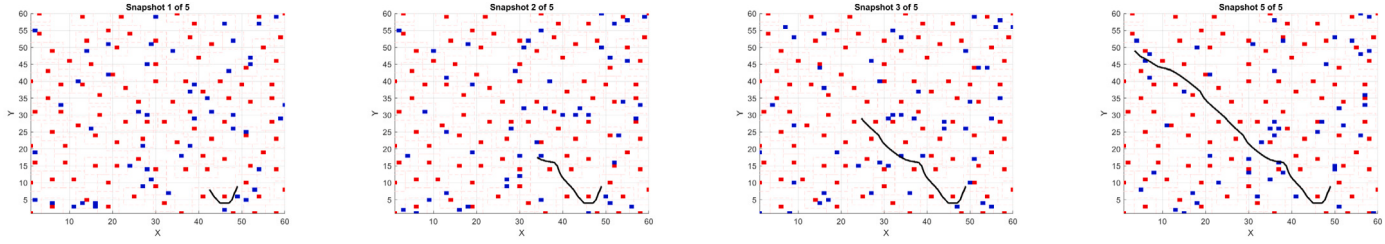
(a) Initial planning. (b) Avoiding obstacle. (c) Midway progress. (d) Reaching goal.

Fig. 7. Snapshots of Path 1 showing initial planning, obstacle avoidance, midway progress, and goal attainment.



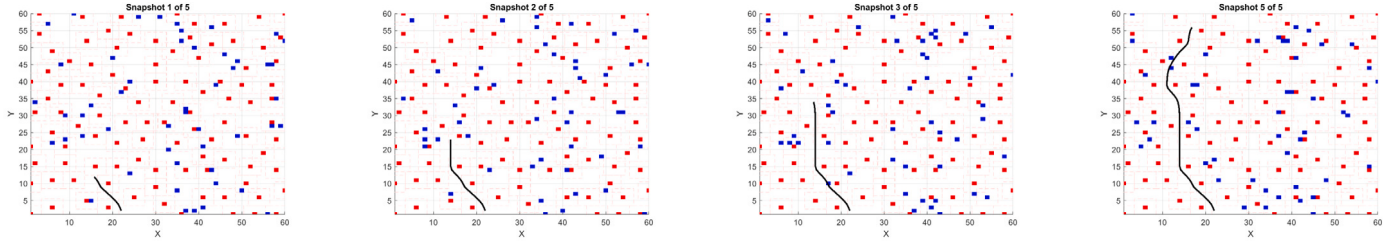
(a) Initial planning. (b) Avoiding obstacle. (c) Midway progress. (d) Reaching goal.

Fig. 8. Snapshots of Path 2 showing initial planning, obstacle avoidance, midway progress, and goal attainment.



(a) Initial planning. (b) Avoiding obstacle. (c) Midway progress. (d) Reaching goal.

Fig. 9. Snapshots of Path 3 showing initial planning, obstacle avoidance, midway progress, and goal attainment.



(a) Initial planning. (b) Avoiding obstacle. (c) Midway progress. (d) Reaching goal.

Fig. 10. Snapshots of Path 4 showing initial planning, obstacle avoidance, midway progress, and goal attainment.

Table 6
Evaluation on Benchmark-Based test cases.

Test Case	Path Length	Euclidean Dist.	Efficiency	Time (s)	Pred. Error
Path1	160.05	146.56	91.57%	15.16	4.57
Path2	254.72	235.64	92.51%	27.41	3.70
Path3	220.17	194.65	88.41%	25.45	4.96
Path4	168.58	151.16	89.67%	18.43	4.96
Path5	211.21	177.60	84.09%	19.71	4.29
Path6	179.01	160.96	89.92%	15.81	4.86
Path7	199.17	163.64	82.16%	34.03	2.80
Path8	218.14	201.56	92.40%	24.60	5.77

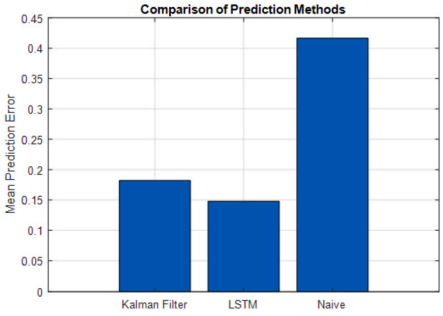


Fig. 11. Mean prediction error comparison between Kalman Filter, LSTM, and Naïve baseline.

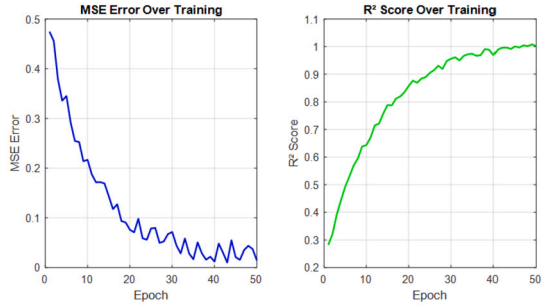


Fig. 12. Training progression of MSE and R^2 over epochs, confirming convergence and accuracy.

Fig. 13. Sensitivity of success rate to missing data and noise level variations.

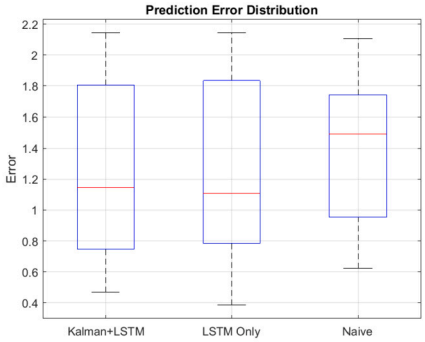


Fig. 14. Prediction error distribution for Kalman+LSTM fusion, LSTM-only, and noisy measurements.

rate of 93.5% and prediction error of 0.090 (Table 10), while maintaining operational viability even in worst-case settings (success rate = 33.9% at 40% missing data and noise level 5).

Fig. 16 shows heatmaps of success rate and prediction error across all degradation configurations. Both metrics degrade monotonically with increasing noise and data loss, confirming graceful performance deterioration. Boxplots in Fig. 17 further illustrate distributional variability: noise significantly increases variance in success rate and error, while missing data induces more stable but degraded accuracy. ANOVA showed that noise and missing data have statistically significant independent effects on performance ($p < 10^{-54}$ and $p < 10^{-41}$), with no significant interaction ($p = 0.437$), validating the robustness of our framework under decoupled perturbations.

Table 7
Performance comparison: Kalman filter vs LSTM vs naive baseline under varying noise and missing rates.

Missing Rate	Noise Level	KF Error	LSTM Error	Naive Error
0.1	4	154.17	148.69	181.47
0.1	4	162.78	146.26	193.67
0.1	6	149.21	140.18	173.43
0.1	6	164.06	154.20	180.87
0.1	8	190.60	163.03	207.26
0.1	8	164.13	140.28	177.62
0.2	4	151.01	133.40	173.37
0.2	4	150.65	142.49	181.06
0.2	6	137.84	124.78	162.25
0.2	6	143.74	120.47	163.90
0.2	8	160.28	140.55	177.93
0.2	8	167.78	136.73	182.69
0.3	4	161.78	124.45	187.28
0.3	4	160.69	140.24	191.77
0.3	6	180.73	157.66	204.24
0.3	6	184.50	153.55	205.65
0.3	8	190.67	168.06	208.70
0.3	8	190.14	169.09	204.16

Table 8
Prediction error variance and standard deviation.

Estimator	Variance	Standard Deviation
Kalman Filter (KF)	270.40	16.44
LSTM	473.53	21.76
Naive Baseline	277.12	16.65

Table 9
Overall performance summary (150 trials).

Metric	Mean	Std. Dev.	Min	Max
Prediction Error	0.328	0.123	0.038	0.613
Success Rate (%)	65.60	15.72	31.44	100.00
Path Length (steps)	16.12	3.67	8.71	23.43
Computation Time (s)	0.858	0.180	0.501	1.202

Comparison of estimation methods in Fig. 11 demonstrates the superiority of our hybrid model. LSTM achieved the lowest mean error (0.15), outperforming Kalman filtering (0.18) and naïve approaches (0.42), justifying their fusion to use complementary strengths—Kalman's temporal smoothing and LSTM's non-linear dynamic modeling. Further analysis under focused degradation settings (noise levels 4–8, missing rates 10%–30%) reinforces earlier observations regarding estimator robustness. Table 7 and Fig. 15 present a detailed comparative breakdown of the Kalman filter, LSTM predictor, and a naive baseline approach. Notably, the LSTM model consistently achieves lower prediction errors than both alternatives, particularly as noise increases. For instance, at 10% missing data and noise level 6, the LSTM error drops to 140.2, outperforming Kalman (149.2) and the naive baseline (173.4). While the LSTM benefits from its nonlinear modeling capability—especially under dynamic uncertainty—the Kalman filter demonstrates reduced adaptability in high-noise regimes due to its linear assumptions. However, its significantly lower variance and tighter error distribution across trials underscore its robustness and consistent behavior under moderate conditions. These findings highlight the complementary strengths of both methods: LSTM excels in capturing complex, velocity-aware dynamics, while Kalman filtering enhances stability and temporal smoothness (Table 8). Such qualities are particularly valuable in edge-case scenarios or

Table 10
Best and worst-case configurations.

Condition	Miss. Rate	Noise	Success (%)	Error	Path Len.	Time (s)
Best Case	0.0	0	93.5 ± 4.4	0.090 ± 0.061	10.01 ± 1.28	0.535 ± 0.034
Worst Case	0.4	5	33.9 ± 2.1	0.574 ± 0.036	22.60 ± 1.07	1.159 ± 0.026

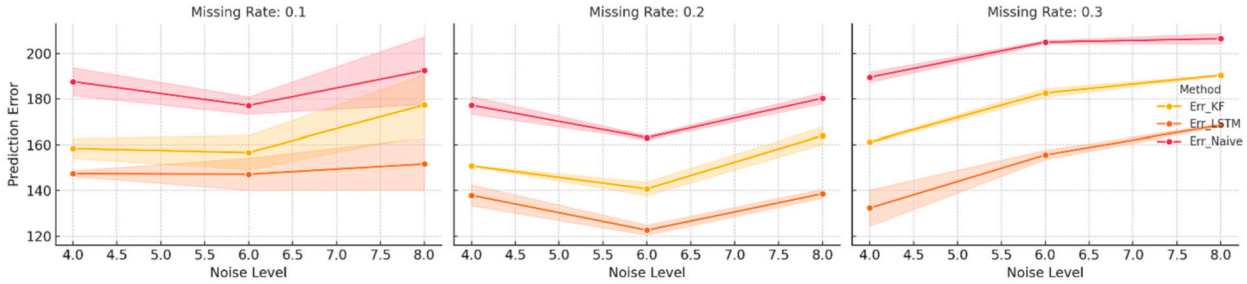


Fig. 15. Prediction error comparison under noise and missing rate variations.

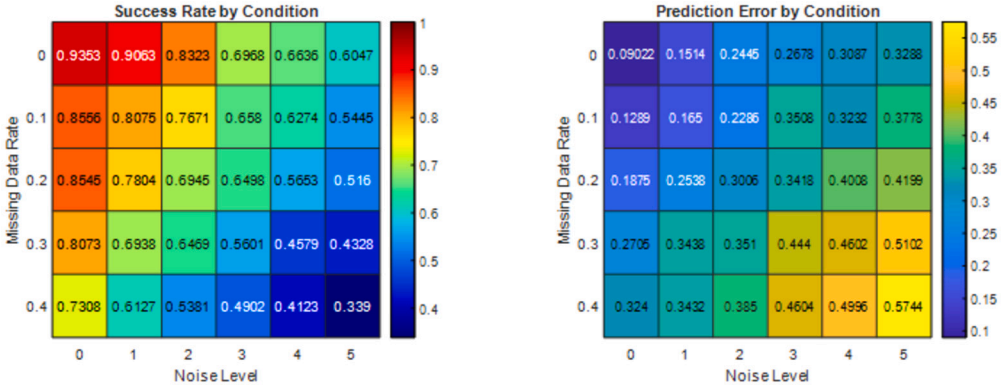


Fig. 16. Effect of Missing Data and Noise on Success Rate and Prediction Error.

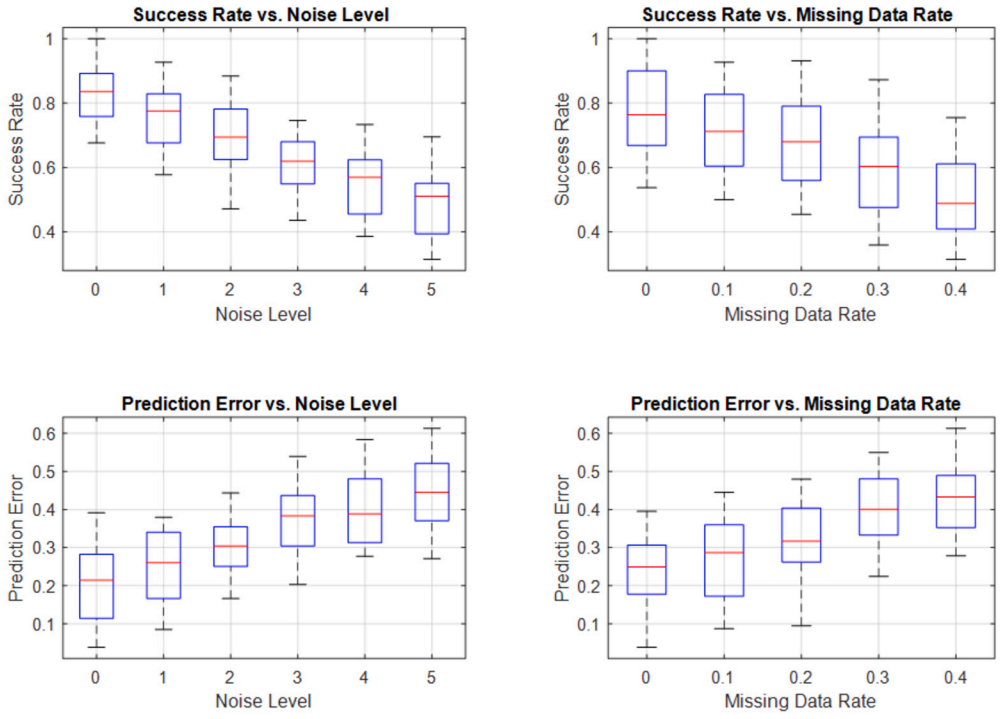
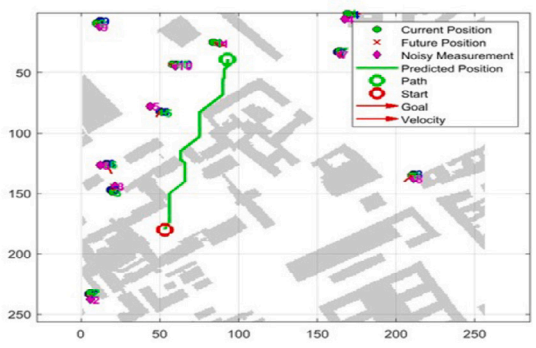
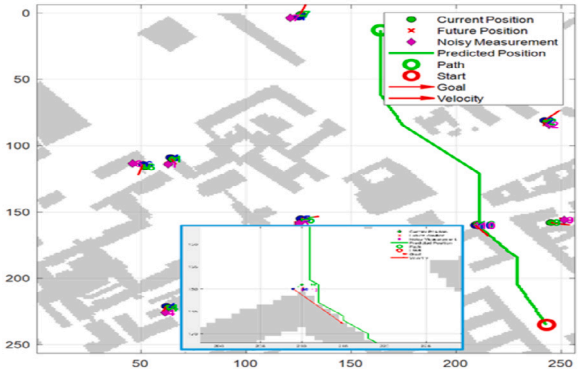


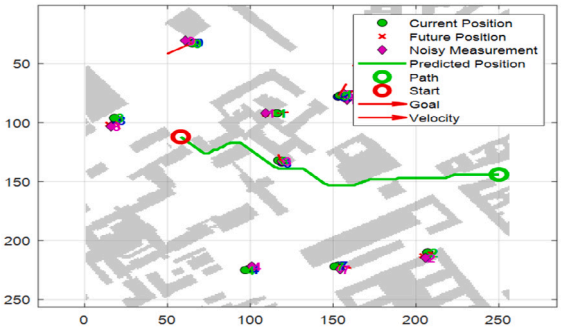
Fig. 17. Boxplots of success rate and prediction error vs. noise level and missing data rate.



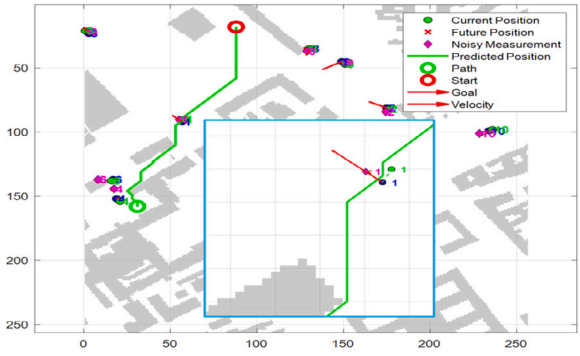
(a) Experiment 1



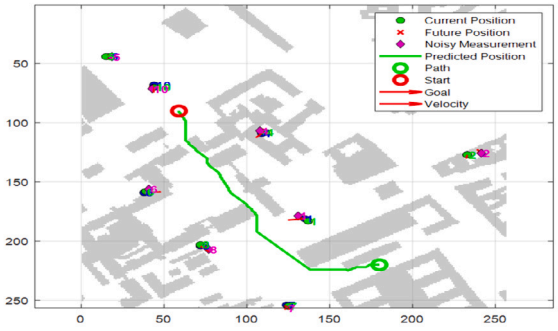
(b) Experiment 2



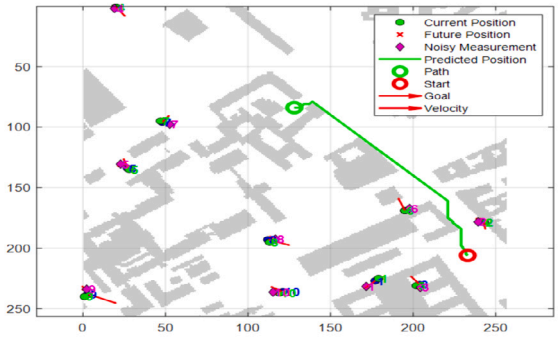
(c) Experiment 3



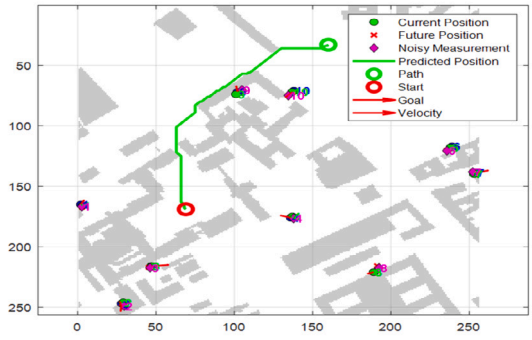
(d) Experiment 4



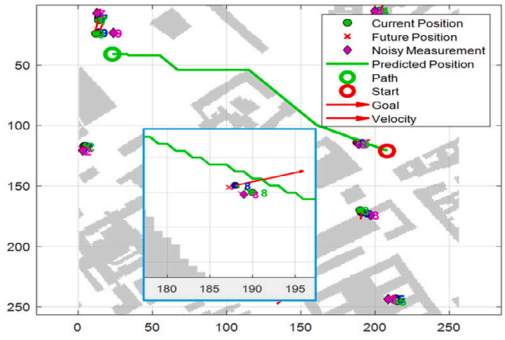
(e) Experiment 5



(f) Experiment 6



(g) Experiment 7



(h) Experiment 8

Fig. 18. Representative results of the LSTM-augmented A* algorithm.

real-world deployments involving non-Gaussian noise, abrupt motion changes, or sensor dropouts. The integration of Kalman filtering into the hybrid architecture is therefore not only justified but strategically beneficial—improving predictive resilience and operational reliability across a wide range of conditions. Training diagnostics (Fig. 12) confirm convergence stability: MSE steadily declines while R^2 exceeds 0.99 by epoch 50, reflecting strong model generalization. Sensitivity analysis (Fig. 13) quantifies the effect of perturbations: performance is especially sensitive to missing data above 20% (drop of ∼0.75) and noise between levels 2–3 (drop of ∼0.085), identifying key thresholds for real-world deployment.

7.2. Environmental correlation and robust fusion strategy

To investigate whether environmental factors affect prediction performance, we analyzed the correlation between prediction errors and two representative variables: (i) the Euclidean distance between each dynamic obstacle and the agent's starting point, and (ii) the obstacle's velocity magnitude. The updated Kalman+LSTM fusion model yielded a mean prediction error of 1.25 with a standard deviation of 0.57, slightly outperforming both the LSTM-only (mean = 1.26, std = 0.60) and naïve noisy input models (mean = 1.38, std = 0.50).

These results are visualized in Fig. 14, which shows the distribution of prediction errors across the three models. Notably, while all methods demonstrate low median errors, the fusion approach slightly narrows the interquartile range compared to LSTM-only, indicating improved consistency. The naïve approach, though competitive in mean, shows heavier tails and greater sensitivity to outliers.

Environmental correlation analysis further supports the robustness of the model. The correlation between obstacle distance and prediction error was negligible ($r = 0.08$, $p = 0.836$), and the correlation with obstacle speed was also weak ($r = -0.16$, $p = 0.650$). These non-significant correlations suggest that the model's error distribution remains stable across different spatial proximities and motion dynamics.

We attribute this resilience to the adaptive fusion mechanism, which dynamically adjusts the contribution of Kalman and LSTM predictions based on both instantaneous deviation and training-derived uncertainty. This blend reduces susceptibility to overfitting and noise-induced volatility, especially under variable motion patterns.

8. Conclusions and future work

This paper introduced a novel hybrid path-planning framework for dynamic environments by integrating Long Short-Term Memory (LSTM) networks with the A* search algorithm. The proposed LSTM-augmented A* algorithm uses predictive obstacle trajectories to proactively generate collision-free paths while preserving the theoretical guarantees—completeness, admissibility, and optimality—of classical A*. Extensive evaluations using both synthetic datasets and standard benchmark maps, including the complex and realistic `Berlin_0_256.map`, demonstrated the algorithm's ability to consistently produce smooth, efficient, and safe paths with enhanced computational performance.

A crucial advancement presented in this study was the integration of Kalman Filters (KFs) with the LSTM-based predictions. Although LSTM effectively captures temporal patterns in obstacle motion, incorporating Kalman Filters significantly enhances robustness by effectively managing noisy measurements and rapid trajectory changes. This combined predictive approach thus considerably improves reliability in dynamic, uncertain, and realistically complex scenarios.

For future research, promising directions include extending the current framework to multi-agent path planning scenarios and validating the approach using real-world sensor data. These enhancements will further increase the framework's applicability and practical impact, particularly in autonomous navigation systems where real-time adaptability and safety are paramount.

Overall, this study establishes a robust methodological foundation for integrating deep learning predictions with heuristic search algorithms, advancing the state-of-the-art in real-time dynamic path planning and autonomous navigation.

9. Declaration

During the preparation of this work the authors used [Grammarly.AI TOOL / SERVICE] in order to [enhance language and readability]. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

CRediT authorship contribution statement

Manar Lashin: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Shady Y. El-mashad:** Writing – review & editing, Validation, Supervision, Methodology. **Abdullah T. Elgammal:** Writing – review & editing, Writing – original draft, Validation, Methodology, Investigation, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.rineng.2025.106324.

Data availability

Data will be made available on request.

References

[1] S.M. LaValle, Planning Algorithms, Cambridge University Press, 2006.
[2] E.W. Dijkstra, A note on two problems in connexion with graphs, in: Edsger Wybe Dijkstra: His Life, Work, and Legacy, 2022, pp. 287–290.
[3] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.
[4] S. Koenig, M. Likhachev, D* lite, in: Eighteenth National Conference on Artificial Intelligence, 2002, pp. 476–483.
[5] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, S. Thrun, Anytime search in dynamic graphs, Artif. Intell. 172 (14) (2008) 1613–1643.
[6] D. Silver, Cooperative pathfinding, in: Proceedings of the Aaai Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 1, no. 1, 2005, pp. 117–122.
[7] Y. Wang, D. Cao, Application of machine learning to intelligent vehicle path planning, in: 2024 IEEE 2nd International Conference on Image Processing and Computer Applications (ICIPCA), IEEE, 2024, pp. 1028–1031.
[8] E. Zhao, N. Zhou, C. Liu, H. Su, Y. Liu, J. Cong, Time-aware maddpg with lstm for multi-agent obstacle avoidance: a comparative study, Complex Intell. Syst. 10 (3) (2024) 4141–4155.
[9] E. Mulás-Tejeda, A. Gómez-Espinosa, J.A. Escobedo Cabello, J.A. Cantoral-Ceballos, A. Molina-Leal, Implementation of a long short-term memory neural network-based algorithm for dynamic obstacle avoidance, Sensors 24 (10) (2024) 3004.
[10] J. Xin, T. Xu, J. Zhu, H. Wang, J. Peng, Long short-term memory-based multi-robot trajectory planning: learn from mpcc and make it better, Adv. Intell. Syst. 6 (9) (2024) 2300703.
[11] Y. Zhang, P. Chen, Path planning of a mobile robot for a dynamic indoor environment based on an sac-lstm algorithm, Sensors 23 (24) (2023) 9802.
[12] M. Likhachev, G.J. Gordon, S. Thrun, Ara*: anytime a* with provable bounds on sub-optimality, Adv. Neural Inf. Process. Syst. 16 (2003).
[13] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, R. Tachibana, Deep reinforcement learning for high precision assembly tasks, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2017, pp. 819–825.
[14] L. Tai, G. Paolo, M. Liu, Virtual-to-real deep reinforcement learning: continuous control of mobile robots for mapless navigation, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2017, pp. 31–36.

[15] R. Bridson, Fast Poisson disk sampling in arbitrary dimensions, SIGGRAPH sketches 10 (1) (2007) 1.
[16] G. James, D. Witten, T. Hastie, R. Tibshirani, et al., An Introduction to Statistical Learning, vol. 112, Springer, 2013.
[17] N. Sturtevant, Benchmarks for grid-based pathfinding, IEEE Trans. Comput. Intell. AI Games 4 (2) (2012) 144–148, [Online]. Available: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf.