aerospace

MDPI

Article

Real-Time On-the-Fly Motion Planning for Urban Air Mobility via Updating Tree Data of Sampling-Based Algorithms Using Neural Network Inference

Junlin Lou *, Burak Yuksek, Gokhan Inalhan and Antonios Tsourdos

School of Aerospace, Transport and Manufacturing, Cranfield University, Bedfordshire MK43 0AL, UK; burakyuksek@gmail.com (B.Y.); inalhan@cranfield.ac.uk (G.I.); a.tsourdos@cranfield.ac.uk (A.T.)
* Correspondence: junlin.lou@cranfield.ac.uk

Abstract: In this study, we consider the problem of motion planning for urban air mobility applications to generate a minimal snap trajectory and trajectory that cost minimal time to reach a goal location in the presence of dynamic geo-fences and uncertainties in the urban airspace. We have developed two separate approaches for this problem because designing an algorithm individually for each objective yields better performance. The first approach that we propose is a decoupled method that includes designing a policy network based on a recurrent neural network for a reinforcement learning algorithm, and then combining an online trajectory generation algorithm to obtain the minimal snap trajectory for the vehicle. Additionally, in the second approach, we propose a coupled method using a generative adversarial imitation learning algorithm for training a recurrent-neural-network-based policy network and generating the time-optimized trajectory. The simulation results show that our approaches have a short computation time when compared to other algorithms with similar performance while guaranteeing sufficient exploration of the environment. In urban air mobility operations, our approaches are able to provide real-time on-the-fly motion re-planning for vehicles, and the re-planned trajectories maintain continuity for the executed trajectory. To the best of our knowledge, we propose one of the first approaches enabling one to perform an on-the-fly update of the final landing position and to optimize the path and trajectory in real-time while keeping explorations in the environment.

Keywords: motion planning; urban air mobility; machine learning; reinforcement learning; generative adversarial imitation learning

check for updates

Citation: Lou, J.; Yuksek, B.; Inalhan, G.; Tsourdos, A. Real-Time On-the-Fly Motion Planning for Urban Air Mobility via Updating Tree Data of Sampling-Based Algorithms Using Neural Network Inference. *Aerospace* **2024**, *11*, 99. https://doi.org/10.3390/aerospace11010099

Academic Editor: Michael Schultz

Received: 1 December 2023
Revised: 5 January 2024
Accepted: 10 January 2024
Published: 22 January 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Urban air mobility (UAM), a novel air transportation concept designed for urban environments, has garnered significant interest in recent years from the aerospace and transportation sectors [1,2]. UAM's goal is to enhance the efficiency of transporting people and goods through specialized vehicles like electric vertical take-off and landing (eVTOL) aircraft and small unmanned air vehicles (sUAVs) [3]. These vehicles, characterized by high levels of automation, navigate autonomously from take-off to landing without human operators. However, this innovative transportation approach faces numerous complex challenges, including environmental concerns, urban infrastructure considerations, and specific issues related to the UAM platform itself. These include building infrastructure interactions [1], dense traffic of aerial vehicles [2], micro-weather patterns [3,4], urban emergencies or disasters [1], and the quality and reliability of communication, navigation, and surveillance systems [5,6]. These elements contribute to the formation of various static, dynamic, and uncertain no-fly zones, obstacles, and geo-fences in urban airspace [7], presenting three primary safety challenges for automated aerial vehicles. Firstly, there is a need for real-time, adaptive re-planning in response to environmental uncertainties. Secondly, the nonlinear kinematics and dynamics of these systems, coupled with aerodynamic

and power constraints, limit the vehicles' cruising speed and acceleration. Thirdly, rapid and feasible trajectory generation becomes crucial, especially when destination vertiports change due to congestion or when emergency scenarios necessitate on-the-fly adjustments. In this paper, we tackle these issues by integrating methodologies from sampling-based path planning [8], trajectory optimization, recurrent neural networks (RNNs) [9], reinforcement learning (RL) [10], generative adversarial imitation learning (GAIL) [11], and transformers [12]. Our approach aims to enable efficient, real-time re-planning under uncertain airspace conditions.

In our study, we have developed two distinct algorithms for addressing the challenges of path planning and trajectory optimization in urban air mobility (UAM) operations: a coupled approach and a decoupled one. These algorithms represent a significant advancement over traditional path planning methods, primarily due to their markedly reduced computational times, which enable real-time implementation in UAM scenarios. The coupled algorithm is intricately designed to ensure that the vehicle adheres to kinematic and dynamic constraints while also achieving the quickest possible journey to its destination. This approach integrates the constraints directly into the planning process, thereby ensuring efficient and safe navigation through urban airspace. On the other hand, the decoupled algorithm focuses on optimizing straight path segments, making it particularly suitable for scenarios where the aerial vehicle must adhere to strict scheduling constraints. This algorithm first plans the path and then optimizes the trajectory, allowing for greater flexibility in dealing with dynamic urban environments. Central to both approaches is the use of tree data generated by the RRT* algorithm or its variants. The RRT* algorithm is known for its asymptotic optimality and probabilistic completeness [13], characteristics that are critical in ensuring a comprehensive exploration of the configuration space to yield feasible and optimal paths. However, RRT*-based algorithms typically require extensive exploration time, making them unsuitable for real-time path planning applications. Our algorithms address this limitation by utilizing the tree data generated from previous RRT* explorations. These tree data consist of a series of nodes, each with specific attributes such as coordinates, cost to the tree root, and the index of its parent node. In more complex models, these nodes may also include velocity and acceleration data. In our tree structure, all nodes maintain optimal paths from the tree root, which is defined as the current position of the vehicle or the position of the departure vertiport. The primary innovation in our approach lies in the real-time update of these tree data in dynamic environments. We aim to continuously guarantee the optimized attributes and connections (parent indices) of the nodes on the tree, as well as the feasibility and optimality of the flight path to the destination. Traditional graph-rewiring techniques based on RRT*, which involve significant forward and backward propagation, are too time-consuming for real-time updates [14]. Instead, we utilize RL policies or GAIL generators to guide the update of tree connections and node attributes. This method replaces computationally expensive calculations with more efficient neural network inference processes. Moreover, our approaches offer unparalleled flexibility in UAM operations. They can adapt to changes in the destination of the aerial vehicle at any moment, generating a feasible and optimal path from the vehicle's current position to the new destination in real-time. This flexibility is a direct result of the intensive exploration that results in tree nodes spread throughout the configuration space and the inherent nature of the tree data, where paths from the root to all nodes are optimized. To the best of our knowledge, our paper introduces some of the first methodologies capable of performing on-the-fly updates of the destination and optimizing the path and trajectory in real-time while maintaining the comprehensive explorations of the environment. This feature is especially valuable in urban settings where sudden changes in destination or route may be necessary due to various unforeseen circumstances such as traffic congestion, weather conditions, or emergency situations. Furthermore, our algorithms are designed to be robust against the dynamic and unpredictable nature of urban airspaces. By continuously updating the tree data based on the latest environmental information, our approaches ensure that the trajectory remains feasible and optimal, even in the face of rapid changes in the

urban landscape. This aspect is crucial for maintaining safety standards and operational efficiency in UAM applications. In summary, the coupled and decoupled algorithms that we propose in this paper represent a significant leap forward in the field of UAM path planning and trajectory optimization. By leveraging advanced techniques in machine learning, such as RL and GAIL, combined with the efficient use of RRT*-based tree data, our approaches not only reduce computational time but also enhance the flexibility and robustness of UAM operations. This makes them highly suitable for the dynamic and complex environment of urban air mobility, where real-time updates and adaptability are key to successful operation.

The structure of this paper is outlined as follows: Section 2 delves into the existing literature relevant to our study, highlighting both the contributions and limitations of previous methods in motion planning and re-planning. In Section 3, we introduce a novel decoupled planning algorithm. This algorithm leverages reinforcement learning and a piecewise polynomial trajectory generation method to produce minimum snap trajectories with significantly reduced computation times, while also ensuring comprehensive environmental exploration. Section 4 is dedicated to discussing our coupled planning algorithm, which employs a generative adversarial imitation learning framework. This approach is adept at generating time-optimized trajectories, again with remarkably short computation durations, and it maintains a thorough exploration of the configuration space. Finally, Section 5 provides the concluding remarks of this paper, summarizing our key findings and contributions to the field of motion planning in urban air mobility.

2. Related Works

Our methodologies are influenced by a wide array of prior research in the field. We utilize the sampling-based rapidly exploring random tree (RRT) algorithm [15], known for efficiently producing collision-free paths. Its advanced version, RRT* [13], further improves upon this by ensuring that the cost of the solution progressively approaches the optimal. The kinodynamic RRT* [16] expands on RRT* by incorporating kinematic and dynamic constraints, thus generating time-optimal trajectories. A notable development in this area is the RL-RRT [17], which stands as a leading kinodynamic planner in terms of efficiency. This approach integrates reinforcement learning (RL) policies with RRT and kinodynamic motion planning, capable of generating high-quality, time-optimal trajectories. In our decoupled motion planning approach, the trajectory generation is based on a sequence of waypoints provided by the path planner. For instance, the minimum snap trajectory algorithm for quadrotors [18] demonstrates that quadrotor dynamics, with four inputs, can be described as differentially flat, thus allowing for trajectory formulation as a polynomial function optimized using quadratic programming (QP). Additionally, there are algorithms that create more aggressive trajectories for quadrotors. These are based on time allocation for waypoints, ensuring collision-free trajectories and state estimation for quadrotors, and have been extended to trajectory generation for fixed-wing aircraft with Dubins-type dynamics [19]. In the realm of trajectory optimization, the Operator Splitting Quadratic Program (OSQP) [20] stands out as a state-of-the-art QP solver, enabling the real-time optimization and generation of polynomial trajectories [21].

In our exploration of related works, various algorithms have been examined for their application in dynamic environment path re-planning. An enhanced bidirectional RRT algorithm is explored in [22], which introduces a tree pruning technique for rapid path updates. This technique retains only a small number of nodes within the configuration space, leading to an incomplete exploration of this space. The simulation results indicate that this limitation adversely affects the quality of the paths generated. Alternative strategies have been proposed, such as on-the-fly real-time planning using simplified tree data [14]. However, this method compromises the thoroughness of configuration space exploration to achieve real-time re-planning capabilities. Considering the complexity and dynamic nature of urban airspace, intensive exploration is crucial for enhancing flight path connectivity and safety. A hybrid method combining virtual force with the A* search

algorithm is presented in [23]. This approach shows promise for simpler scenarios, yet it struggles with the explicit construction of configuration space, a significant challenge in more intricate planning problems, particularly in high-dimensional spaces. The use of inverse reinforcement learning (IRL) [24], a traditional imitation learning algorithm, for aerial vehicle path re-planning in dynamic environments is investigated in [25], alongside the RRT*. Despite its applications, IRL faces challenges such as complex implementation and the necessity of reinforcement learning in an inner loop, leading to high computational costs. Contrasting IRL, generative adversarial imitation learning (GAIL) offers simpler implementation and lower computational demands. Crucially, unlike IRL which learns a cost function, GAIL focuses on learning a policy network. This distinction allows GAIL to excel in more complex planning problems, as it directly learns policies for behavior generation rather than the behaviors themselves.

3. An Reinforcement-Learning-Based Approach for Generating Minimal Snap Trajectory

In this section, we introduce a decoupled re-planning algorithm that combines reinforcement learning with online minimum snap trajectory generation algorithms. This approach is designed to re-plan trajectories with considerably reduced computational times while still ensuring a thorough exploration of the environment. A prominent challenge in utilizing reinforcement learning for this purpose is the impracticality of using the complete tree data as an input to the policy network [26]. The tree data, encompassing an extensive number of nodes each with multiple attributes, can be seen as a high-dimensional vector. This complexity exceeds the processing capabilities of standard reinforcement learning techniques. To overcome this obstacle, we employ recurrent neural networks (RNNs) as the policy network within our RL framework. This choice not only addresses the issue of handling large-scale tree data but also solves several other related challenges. Figure 1 illustrates the structure of this policy network.

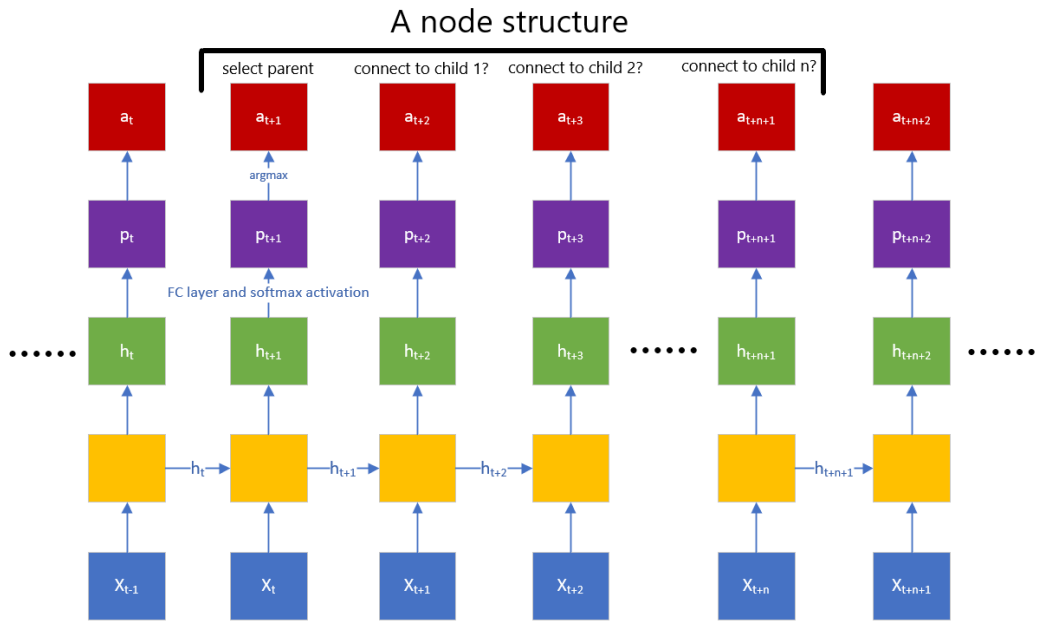


Figure 1. Recurrent neural networks policy network of RL.

3.1. Obtain Waypoints via Reinforcement Learning with RNN Policy Network

Proximal policy optimization (PPO) [27] is utilized in the training phase of the reinforcement learning framework. This algorithm is an evolution of the trust region policy optimization (TRPO) [28], which employs a second-order trust region method to maximize the objective function. In contrast, PPO utilizes a first-order method, simplifying implementation while maintaining, if not surpassing, the performance of TRPO. Moreover, PPO

presents several distinct advantages for the approach outlined in this section, particularly when compared with other policy gradient learning algorithms. Firstly, PPO exhibits a lower sensitivity to the learning rate. This aspect facilitates an easier tuning of hyper-parameters and results in a more stable learning process. Secondly, the hyper-parameters in PPO demonstrate robustness across a diverse range of tasks [29], enhancing its applicability. Finally, PPO is capable of achieving comparable performance to other policy gradient methods with fewer state, action, and reward samples, making it a more efficient choice in terms of data requirements.

The asymptotic optimality characteristic of the RRT* algorithm is primarily attributed to its ChooseParent and Rewire operations. During the ChooseParent operation, the algorithm assesses a set of potential parent nodes within a defined Euclidean distance of a newly added child node. This process involves checking for collisions and selecting the most suitable parent node based on certain criteria to update the tree. Subsequently, the Rewire operation involves examining nodes within a specified neighborhood of the new node. If no collision is detected and a reduction in the cumulative cost is possible, the new node is then designated as a parent to some of these neighboring nodes.

Our RNN-based policy network comprises several dozen node structures, with the cumulative number of these structures in multiple networks equaling the total number of nodes in the tree data. Typically, a policy network contains about 20 node structures. Each node structure in the network consists of $n_c + 1$ tokens, with each token responsible for generating an action. Here, n_c represents the count of nearby nodes. The network's architecture is defined by matrices W_{ih}, W_{hh}, and W_{ho}, representing input-to-hidden, hidden-to-hidden, and hidden-to-output connections, respectively. The bias vectors b_h and b_o are associated with the hidden and output layers. Each node structure transforms its input data F_t into X_t using embedding matrices E_1 and E_2: $X_t = E_1 \cdot \text{one-hot}(F_t^{\text{index}}) + E_2 \cdot F_t^{\text{pos}}$. The first token's output in each node structure is an n_c-dimensional one-hot vector, computed as $\text{softmax}(W_{ho} \cdot h_t + b_o)$, that denotes the selection of the parent node. The outputs of the subsequent tokens, from the second up to the $n_c + 1$th token, are two-dimensional one-hot vectors indicating the selection of corresponding nodes as child nodes, also determined through softmax functions. During training, the agent learns to add or modify connections within an environment that already includes sampled nodes and some existing connections. At each time step, the RNN updates its hidden state h_t using $h_t = \text{ReLU}(W_{ih} \cdot X_t + W_{hh} \cdot h_{t-1} + b_h)$ and computes the output y_t as $y_t = \text{softmax}(W_{ho} \cdot h_t + b_o)$. This approach helps to circumvent issues associated with excessively long sequences. Furthermore, each node is limited to observing only the states of its n nearest neighbors. This partial observability of the environment is addressed by the RNN's ability to retain information across time steps, with the hidden state h_t being a function of both the current input X_t and the previous hidden state h_{t-1}. The parameter n_c is a hyper-parameter influencing the speed of convergence and training outcomes of the reinforcement learning algorithm. Since our approach does not include a configuration space sampling step, the asymptotic optimality condition does not dictate the choice of n_c value. Upon user-defined n_c, an iteration over tree data is performed to identify the n_c nearest nodes for each node. Prior to training, tree data are updated to include these n_c nearest nodes for every node.

Figure 2 shows the details of the token of the RNN policy network. Part of the node information F_t undergoes data pre-processing to obtain X_t. This process is mathematically represented as follows: X_t is the result of concatenating two transformed components of F_t. The first component, the one-hot encoded node index F_t^{index}, is transformed by an embedding matrix E_1 (dimension $k_d \times k_v$) into a k_d-dimensional vector. The second component, the node's two-dimensional position F_t^{pos}, is transformed by another embedding matrix E_2 (dimension $l_d \times 2$) into an l_d-dimensional vector. Therefore, $X_t = [\text{Embed}_1(\text{one-hot}(F_t^{index})), \text{Embed}_2(F_t^{pos})]$. For each token, the input X_t and the feature vector h_t output by the hidden layer of the previous token are input into the current token's hidden layer to obtain h_{t+1}. This can be represented as $h_{t+1} = \text{ReLU}(W_{ih} \cdot X_t + W_{hh} \cdot h_t + b_h)$, where W_{ih} and W_{hh} are the weight matrices and b_h is the bias vector. The state

s_t at time t is given by $s_t = [F_t, h_t]$. A fully connected layer followed by softmax activation is then applied to h_{t+1}, resulting in a probability distribution p_{t+1}. The action a_{t+1} is determined by applying the argmax function on p_{t+1}; formally, $a_{t+1} = \text{argmax}(p_{t+1})$. Hence, the policy network is denoted as $\pi(a_{t+1}|[F_t, h_t]; \theta)$, where θ represents the network parameters.

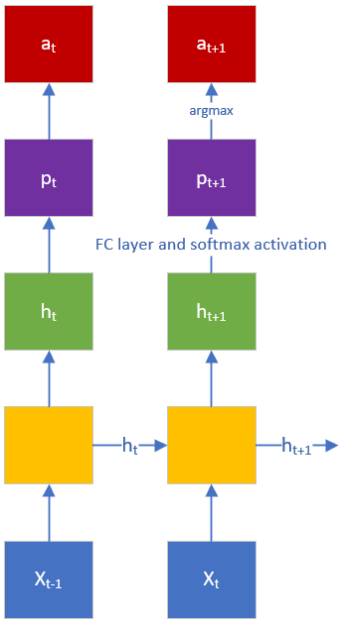


Figure 2. Details of token of RNN policy network.

Figure 3 shows the details of data pre-processing. F_t is a three-dimensional vector [$index$, $posx$, $posy$] including the index, x coordinate, and y coordinate of the node. The start point is always the first node structure with index 0. Each obstacle contains four vectors and each vector contains three parts. For the sets of coordinates of each of the four corners, these four vectors share the same obstacle identification code, and all vectors of obstacles share the same type code. Obstacle information must be input in each token of all node structures, and will be concatenated after the embedding layer, keeping the size of the input vectors the same. The index needs to be one-hot encoded because the index does not represent the numerical feature of the node but a category. For example, if we use the index number directly, the computer will misunderstand that the sum of the first node and the second node is equal to the third node, which is unreasonable. After one-hot encoding, each index number becomes a k_v-dimensional vector, where k_v is the number of nodes in the tree data. This vector is too sparse, so we need to reduce its dimension with embedding layer 1. Embedding layer 1 is a $k_d \times k_v$ matrix, and outputs a k_d-dimensional vector, where k_d is a hyper-parameter that needs to be adjusted by the user. The two-dimensional coordinates of the nodes will be input into embedding layer 2, which amplifies the location features of the nodes by increasing the dimension. Embedding layer 2 is an $l_d \times 2$ matrix that outputs an l_d-dimensional vector. l_d is also a hyper-parameter that needs to be tuned by the user. Then, the output of embedding layer 1 and embedding layer 2 is concatenated into a vector, which is the aforementioned X_t. The matrix parameter of embedding layers 1 and 2 will be learned via reinforcement learning in the next step.

Algorithm 1 shows the process of training the RNN policy network through reinforcement learning. Randomizing obstacles, node indexes, and tree roots when each episode starts is carried out in order to learn the policy of on-the-fly re-planning in a dynamic environment and to guarantee the robustness. There are m nodes in the tree data, and each episode contains $m(n+1)$ steps. To train the policy network, we need to define the reward r_t. In the first n steps of each $n+1$ steps, the rewards remain zero.

$$r_{(i-1)(n+1)+1} = r_{(i-1)(n+1)+2} = \cdots = r_{(i-1)(n+1)+n} = 0 \tag{1}$$

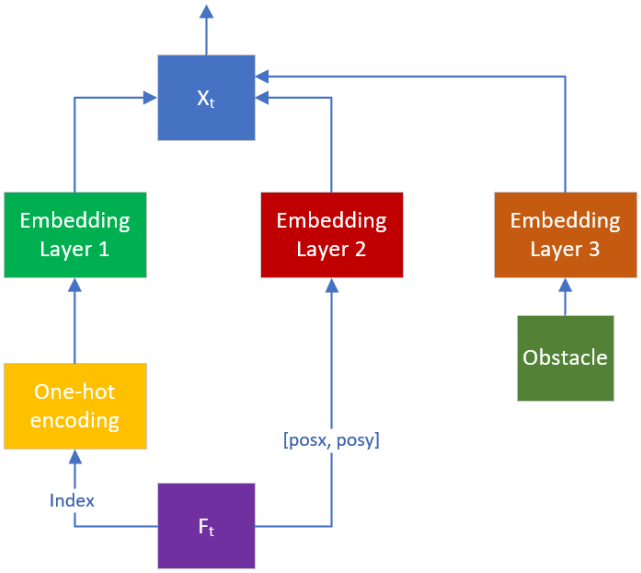


Figure 3. Details of data pre-processing.

Algorithm 1: Training of RNN policy network.

Initialize the parameter θ for new RNN policy network and parameter θ_{old} for old RNN policy network;
for *episode* $\leftarrow 1 : K$ **do**
  clear tree connections, randomize obstacles, tree root and index of nodes;
  $\theta = \theta_{old}$;
  for $i \leftarrow 1 : m$ **do**
    for *step* $\leftarrow 1 : m(n+1)$ **do**
      collect state and action via $\pi_{\theta old}$;
      while *current step* $= i(n+1)$ *is True* **do**
        update tree connections;
        collect reward;
        update parameter θ of RNN policy network via Equation (3);
        $\theta_{old} \leftarrow \theta$;
      end
    end
  end
  Adapt the entropy regularization coefficient in Equation (3) in the light of Equation (6) and then update
  entropy regularization with new policy network π_θ;
  return θ;
end

After the $n+1$ step, all decisions are obtained to ensure a node structure, and then updating tree data, collecting the reward, and updating return (aka cumulative reward) u_t are carried out. The normalized reward includes:

- Connecting each previously unconnected node;
- Each node that has a lower cost
- Each connection that collides with obstacles;
- The Euclidean distance from each node to the tree root;
- A big punishment if the parent is also connected as a child.

At the beginning of each episode, each node is an orphan node and each node's cost to the tree root is infinite. When the tree growing from the root is connected to an orphan node, the cost becomes a finite real number and a fixed reward is obtained. If subsequent

updates of tree data reduce the cost from a node to the tree root, a normalized reward will be obtained.

All steps inside each package of $n+1$ steps have the same return.

$$u_{(i-1)(n+1)+1} = u_{(i-1)(n+1)+2} = \cdots = u_{i(n+1)} = u_{(i-1)(n+1)} + r_{i(n+1)} \tag{2}$$

Afterward, with the proximal policy optimization (PPO) algorithm [27], the parameter of the policy network will be updated once every $n+1$ steps via the clipping surrogate objective shown in Equation (3)

$$\begin{aligned} L^{clip}(\theta) = \mathbb{E}(min(r(\theta)A_{\theta old}([F_t, h_t], a_{t+1})), clip(r(\theta), 1-\epsilon, 1+\epsilon)A_{\theta old}([F_t, h_t], a_{t+1}) \\ +\lambda H([F_t, h_t], a_{t+1})) \end{aligned} \tag{3}$$

where ϵ is a hyper-parameter used to control the clipping ratio. $\mathbb{E}[\cdot]$ denotes the expectation operator obtained by Monte Carlo approximation [30]. $A_{\theta old}[\cdot]$ is an advantage function under the old policy. $r(\theta)$ in Equation (3) denotes the probability ratio

$$r_t(\theta) = \frac{\pi_\theta((a_{t+1}|[F_t, h_t]))}{\pi_{\theta old}((a_{t+1}|[F_t, h_t]))} \tag{4}$$

$H[\cdot]$ is the entropy operator displayed in Equation (5). Entropy is used to measure the probability distribution output by the softmax activation function in the policy network. Smaller entropy leads to a more concentrated probability distribution. We found that entropy shrinks to a small value prematurely during the training process, resulting in insufficient exploration in policy learning, therefore obtaining a defective policy. In the face of this problem, we then introduce entropy regularization [31] to Equation (3).

$$H[s_t; \theta] = -\sum \pi(a_{t+1}|s_t; \theta) \ln \pi(a_{t+1}|s_t; \theta) \tag{5}$$

$$\lambda = \mu \tanh \frac{episode}{\eta} \tag{6}$$

where λ denotes the adaptive entropy regularization coefficient shown in Equation (6), and μ and η are hyper-parameters that determine λ.

Although the standard RNN has an efficient structure for the generation of sequential tree data in our case, it suffers from vanishing and exploding gradient problems [32] due to its simple iterative mechanism. This makes the standard RNN inappropriate for handling long-term dependencies. To address these limitations, we incorporate long short-term memory (LSTM) [33] and a gated recurrent unit (GRU) [34] in our simulation. LSTM, introduced by [33], is designed to have longer memory capabilities than standard RNNs thanks to its unique structure of gates. An LSTM unit contains four interacting components: the forget gate, the input gate, the cell state, and the output gate. The forget gate decides what information should be discarded from the cell state using a sigmoid function. The input gate controls the extent of new information to be added to the cell state, while the cell state carries the network's memory over time with minimal alterations. Finally, the output gate determines the next hidden state. Each of these gates has its parameter matrix, typically of dimensions $h_t \times (h_t + X_t)$, where h_t is the size of the hidden state, and X_t is the size of the input vector. The GRU, proposed by [34], simplifies the architecture of LSTM by combining the forget and input gates into a single update gate. It also merges the cell state and hidden state, resulting in a more compact architecture. The GRU comprises three gates: the reset gate, the update gate, and the candidate hidden state. The reset gate determines how much of the past information to forget, while the update gate controls the extent to which the unit updates its activation or state. The candidate hidden state creates a candidate vector using the current input and the past hidden state. The GRU's parameter matrices for these gates are smaller than those of LSTM, contributing to a reduction in

computational complexity and memory requirements. The choice between LSTM and the GRU typically depends on the specific requirements of the task and the computational resources available. While LSTM offers more control over the information flow within the unit, the GRU provides a more streamlined and computationally efficient architecture. Both have been shown to effectively address the limitations of standard RNNs in handling long-term dependencies in sequential data. Both of LSTM and the GRU will be utilized in our simulations.

3.2. Generate Minimum Snap Trajectory

The subsequent phase in our methodology involves generating a secure trajectory using the waypoints and a predefined total time. For the purposes of simulation, we consider a multi-rotor system, specifically a quadrotor, as our vehicle model. This choice allows us to employ the minimum snap formulation for quadrotor trajectory generation [18]. The formulation demonstrates that piecewise polynomial trajectories are effective in defining the flat outputs for the coordinates in various dimensions, as well as the yaw angle x, y, z, ψ, and their derivatives, ensuring a smooth trajectory. This is feasible due to the differential flatness characteristic inherent in vehicles akin to quadrotors. Given that the cruise phase of UAM vehicles typically occurs within a specific altitude range in urban airspace and involves small altitude changes [1], we will omit the planning of the z coordinate in our model. Additionally, the planning for the yaw angle, ψ, is an aspect that we intend to explore in future work.

Given the start time of trajectory and all end times of segments (τ_0, τ_1, τ_2,$\cdots$,τ_M) and one dimension out of the x, y of the M-segment, the N^{th}-order piecewise polynomial trajectory [35] can be written as Equation (7).

$$f(\tau) = \begin{cases} \sum_{j=0}^{N} c_{1j}(\tau - \tau_0)^j, & \tau_0 \le \tau \le \tau_1, \\ \sum_{j=0}^{N} c_{2j}(\tau - \tau_1)^j, & \tau_1 \le \tau \le \tau_2, \\ & \vdots \\ \sum_{j=0}^{N} c_{Mj}(\tau - \tau_{M-1})^j, & \tau_{M-1} \le \tau \le \tau_M \end{cases} \tag{7}$$

where c_{ij} is the jth-order polynomial coefficient of the i^{th} segment. As minimum snap trajectory is required, we phrase the problem as the generic minimization of the fourth derivative of $f(\tau)$. In that case, the trajectory optimization problem for flat polynomial output is described as Equation (8):

$$\begin{aligned} &\text{minimize} \quad \int_{\tau_0}^{\tau_M} \left(\frac{d^4 f(\tau)}{d\tau^4} \right)^2 d\tau \\ &\text{subject to} \\ &f(\tau_0) = s_0, \\ &f(\tau_M) = s_M, \\ &\frac{d^k f}{d\tau^k}(\tau_i^-) = \frac{d^k f}{d\tau^k}(\tau_i^+), \\ &\text{for } k = 0, 1, \cdots, N-1 \text{ and } i = 1, 2, \cdots, M-1. \end{aligned} \tag{8}$$

where s_0 and s_M are the start and goal position; $\frac{d^k f}{d\tau^k}(\tau_i^-)$ represents the kth-order derivative of the trajectory to the left of the ith segment point; $\frac{d^k f}{d\tau^k}(\tau_i^+)$ represents the k^{th}-order derivative of the trajectory to the right of the ith segment point. Such constraints ensure that, at each segment point, the trajectory and its derivatives up to order $N-1$ (including position, velocity, acceleration, etc.) are continuous.

The objective function (8) can be written as $\min \mathbf{p}^T \mathbf{Q} \mathbf{p}$ and $\min \sum_{i=1}^{M} \mathbf{p}^T \mathbf{Q}_i \mathbf{p}$, where $\mathbf{p} = [\mathbf{p}_1^T, \mathbf{p}_2^T, \cdots, \mathbf{p}_i^T, \cdots, \mathbf{p}_M^T]^T$ is the vector of the polynomial coefficients of all segments [18,19], and $\mathbf{Q}_i$ and $\mathbf{Q}$ are shown as (9) and (10):

$$\mathbf{Q}_i = \begin{bmatrix} 0_{4,4} & 0_{4,N-3} \\ 0_{N-3,4} & \frac{q_r!}{(q_r-4)!}\frac{q_c!}{(q_c-4)!}\frac{t_i^{q_r+q_c-7}-t_{i-1}^{q_r+q_c-7}}{q_r+q_c-7} \end{bmatrix} \tag{9}$$

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 & & & \\ & \mathbf{Q}_2 & & \\ & & \ddots & \\ & & & \mathbf{Q}_M \end{bmatrix} \tag{10}$$

where $4 \leq q_r \leq N, q_r \in \mathbb{N}$ is the row index starting from number 0 and $4 \leq q_c \leq N, q_c \in \mathbb{N}$ is the column index starting from number 0.

The trajectory has a predefined start point, waypoints, and end point, and the vehicle following the trajectory has dynamic constraints. Moreover, to ensure continuity of trajectory, we enforce the same velocity and acceleration for adjacent segments at the segment transition times $(\tau_1, \cdots, \tau_{M-1})$. The conditions mentioned above can be formulated as either linear equality $(\mathbf{A}_{eq}\mathbf{p} = \mathbf{b}_{eq})$ or inequality $(\mathbf{A}_{lq}\mathbf{p} \leq \mathbf{b}_{lq})$ constraints. As a result, we form a quadratic programming problem (QP) for trajectory generation, shown in Equation (11).

$$\begin{aligned} \min \quad & \mathbf{p}^T\mathbf{Q}\mathbf{p} \\ \text{s.t.} \quad & \mathbf{A}_{eq}\mathbf{p} = \mathbf{b}_{eq} \\ & \mathbf{A}_{lq}\mathbf{p} \leq \mathbf{b}_{lq} \end{aligned} \tag{11}$$

In this trajectory optimization formulation, specific constraints have been defined to ensure the practicality and efficiency of the vehicle's trajectory. These constraints are detailed as follows:

1. Equality Constraints$(\mathbf{A}_{eq}\mathbf{p} = \mathbf{b}_{eq})$: These constraints ensure that the vehicle's trajectory precisely passes through predetermined waypoints. Each row in matrix $\mathbf{A}_{eq}$ and vector $\mathbf{b}_{eq}$ corresponds to a waypoint, ensuring that the vehicle reaches these points at specific times. This is critical for maintaining the accuracy and reliability of the trajectory, especially in environments where deviation from the planned path can lead to inefficiencies or safety hazards.
2. Inequality Constraints $(\mathbf{A}_{lq}\mathbf{p} \leq \mathbf{b}_{lq})$: These constraints are imposed to adhere to the vehicle's dynamic limitations, such as maximum allowable velocities and accelerations. They ensure that the trajectory remains within the vehicle's operational capabilities, thereby avoiding scenarios where the vehicle is required to perform beyond its feasible mechanical limits.

In this scenario, the overall duration of the trajectory is predetermined. It is crucial to allocate specific time intervals $(\tau_i - \tau_{i-1})$ to each trajectory segment, as these allocations significantly influence the formation of the cost matrix. Optimizing this time allocation is vital, as it directly impacts the quality of the final trajectory. A conventional approach is to start with an initial estimate of segment times and then refine these estimates iteratively using gradient descent methods [18,19]. However, this process can be time-intensive and challenging to implement in real-time applications. Moreover, we observe that some trajectory segments, particularly those with excessive convexity, may intersect with obstacles, despite the underlying path being collision-free. To mitigate these issues, we introduce two methods. In the first method, we draw inspiration from the work of Chen et al. [35]. We incorporate the concept of a corridor as a constraint in the quadratic programming (QP) problem, which aids in avoiding such intersections and optimizes the trajectory within the defined spatial boundaries.

Figure 4 illustrates our approach of uniformly introducing additional sub-waypoints along the path and establishing an inequality constraint regarding their positions. Concurrently, we transition from a stringent waypoint positional constraint to a more lenient one. This alteration permits the trajectory to approximate the waypoints rather than necessitating direct passage through them. The weak constraint, as formulated in Equation (12), effectively transforms the requirement for the vehicle to pass a specific position at a pre-set time into the more flexible goal of reaching a designated square area within the same timeframe. This adjustment in the algorithm implicitly optimizes the timing by modifying segment transition points. Although this approach increases the number of trajectory segments, it enables the generation of a trajectory in approximately 0.1 s using the state-of-the-art quadratic programming (QP) solver OSQP [20], as it obviates the need for iterative computations. Furthermore, this strategy restricts the trajectory within a corridor demarcated by multiple square areas, ensuring spatial conformity.

$$\begin{aligned} [1, \tau_\phi, \tau_\phi^2, \cdots, \tau_\phi^N] p_\phi &\leq p_\phi(\tau_\phi) + d_{cor} \\ [-1, -\tau_\phi, -\tau_\phi^2, \cdots, -\tau_\phi^N] p_\phi &\leq -p_\phi(\tau_\phi) + d_{cor} \end{aligned} \tag{12}$$

where p_ϕ and τ_ϕ are positions and allocated time for particular waypoints or sub-waypoints, and d_{cor} denotes half the side length of the square area. The value of d_{cor} is set manually by the user, but it cannot be too small in order to keep the square areas connected to form a corridor, and $\sqrt{2}d_{cor}$ cannot be greater than the difference between clearance defined in the path planning algorithm and the flight interval stipulated by the local traffic regulations.

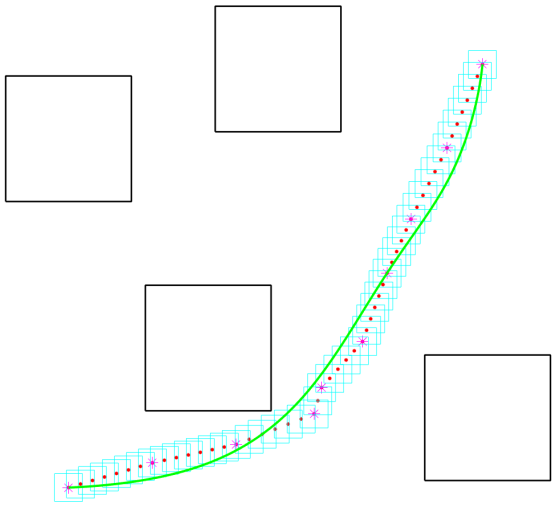


Figure 4. Trajectory generation with corridor for collision avoidance and implicit time optimization. Purple star marks are original waypoints and Red dots are sub-waypoints.

The trajectory optimization problem formulated here bears resemblance to the concepts of interpolating and smoothing splines, which is worth mentioning for a comprehensive understanding.

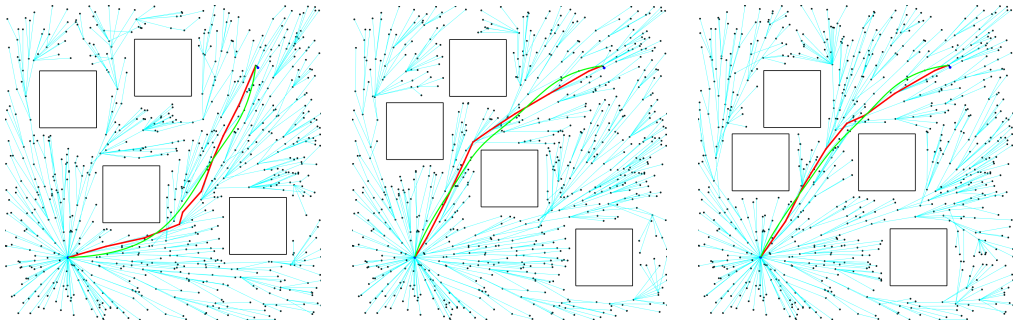
Interpolating Splines: In Equation (11), the vehicle's trajectory is designed to precisely pass through predetermined waypoints, akin to interpolating splines where the curve passes exactly through the set of knots (waypoints). The equality constraints ($\mathbf{A}_{eq}\mathbf{p} = \mathbf{b}_{eq}$) ensure this interpolative nature of the trajectory, maintaining high precision in path planning.

Smoothing Splines: Conversely, the introduction of leniency in waypoint navigation reflects the principles of smoothing splines. Here, the trajectory does not strictly adhere to each waypoint but optimizes for a balance between path fidelity and operational constraints. This aspect is particularly evident in the formulation given by Equation (12), where the

vehicle is required to reach within a specified square area, rather than a precise point, aligning with the smoothing spline's objective of overall curve smoothness and practicality.

Such an approach, incorporating elements of both interpolating and smoothing splines, enables our trajectory optimization model to not only ensure the specific established routes but also to maintain the vehicle's dynamic feasibility and operational safety.

As illustrated in Figure 5, our proposed decoupled approach effectively solves real-time motion re-planning problems in dynamic environments, starting from a fixed point. This approach leverages tree data that have been thoroughly explored, demonstrating that, even without additional configuration space sampling, high-quality path planning or re-planning in dynamic environments is achievable. This is accomplished by updating and optimizing the connections between existing tree nodes, followed by the generation of a collision-free trajectory with optimized snap, in accordance with the planned path. Figure 6 showcases the dynamic capabilities of our vehicle in real-time re-planning. In instances where on-the-fly re-planning is necessary, the closest tree node to the vehicle's position becomes the new tree root (or the starting point for path planning). This method provides a significant advantage by allowing for the direct modification of the start point without the need for complex procedures like a root-moving algorithm [14]. Post obtaining the waypoints, the initial state of the trajectory, considering the vehicle's current position, velocity, and acceleration, is determined, rather than the path's starting point, to generate the trajectory. The simulation results, depicted in Figure 6a–c, illustrate our vehicle's capability to perform real-time motion re-planning during flight, especially when the original trajectory becomes infeasible due to unpredictable and dynamic obstacles. The paths generated by the RNN policy network are represented by red straight line segments, and the minimal snap trajectories are shown as green curves. Additionally, Figure 6e demonstrates the method's effectiveness in rapidly re-planning a safe and optimized trajectory if the final landing point needs alteration, such as in emergency situations. Figure 6d,f confirm that our approach ensures the continuity of the trajectory during re-planning.

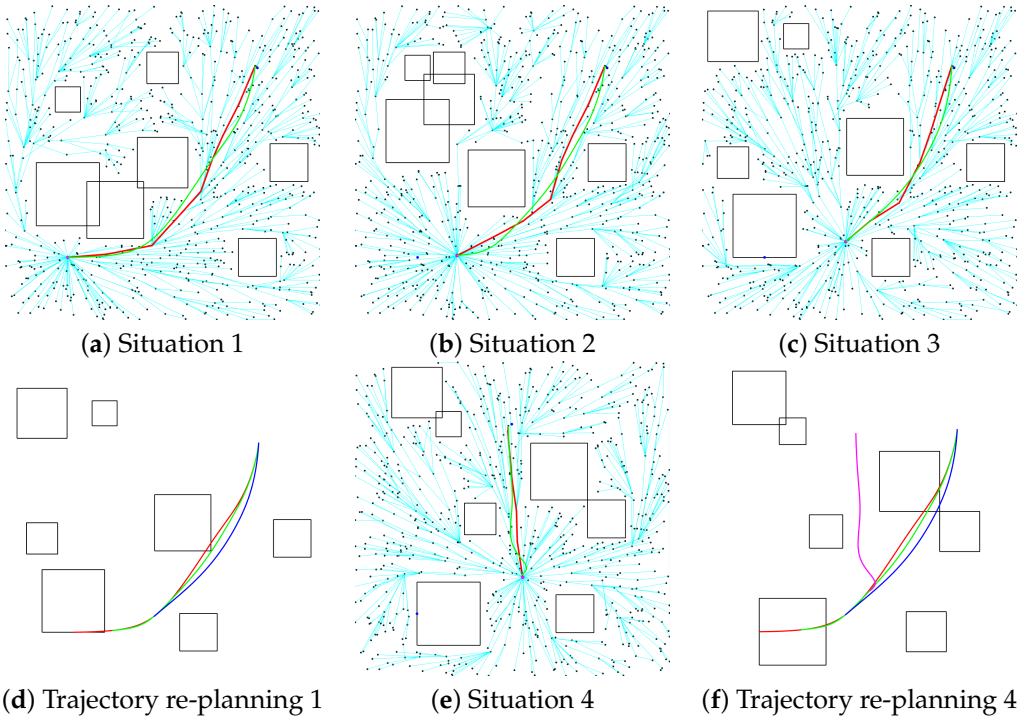


(**a**) Situation 1 (**b**) Situation 2 (**c**) Situation 3

Figure 5. Real-time motion re-planning in dynamics environment. Red lines refer to path and green curve refers to trajectory.

Table 1 presents simulation execution times on a computer with an AMD Ryzen 9 5900× CPU (from Advanced Micro Devices, Inc. Santa Clara, CA, USA), NVIDIA RTX 3090 GPU (from EVGA Corporation, Brea, CA, USA), and 32 GB RAM (from Corsair, Fremont, CA, USA). The benchmarks include the informed RRT* [36] and a method by Richter et al. [19] that uses a substitution technique to convert the trajectory generation problem into an unconstrained QP, enhancing efficiency in complex trajectory scenarios. For comparison, we also employed batch informed trees (BIT*s) [37] to generate waypoints, followed by our approach for trajectory generation. Our algorithm, built on a set of tree data containing 800 nodes and using LSTM or a GRU as the policy network, was tested with different configurations of potential parents and children. The application of the NVIDIA CUDA Deep Neural Network library (cuDNN) enabled GPU acceleration for LSTM and the GRU. In 20 randomly sampled environments similar to Figure 6, each algorithm was

applied separately, and the snaps and mean computation times were recorded. After normalizing the snaps with those of the benchmark algorithm, we calculated the mean normalized snap. Our proposed method showed significantly shorter computation times compared to the first benchmark algorithm, maintaining high-quality trajectory generation. The BIT*'s performance was similar in terms of processing time and result accuracy, but its biased sampling approach did not provide the comprehensive environment exploration required in UAM scenarios. In contrast, our method's capability to quickly generate trajectories to nearby vertical ports and select the most suitable one in emergencies makes it particularly advantageous for UAM applications compared to BIT*s.



(**a**) Situation 1 (**b**) Situation 2 (**c**) Situation 3

(**d**) Trajectory re-planning 1 (**e**) Situation 4 (**f**) Trajectory re-planning 4

Figure 6. Real-time on-the-fly re-planning in dynamics environment. Curves of various colours refer to trajectories generated under Situation 1, 2, 3, and 4.

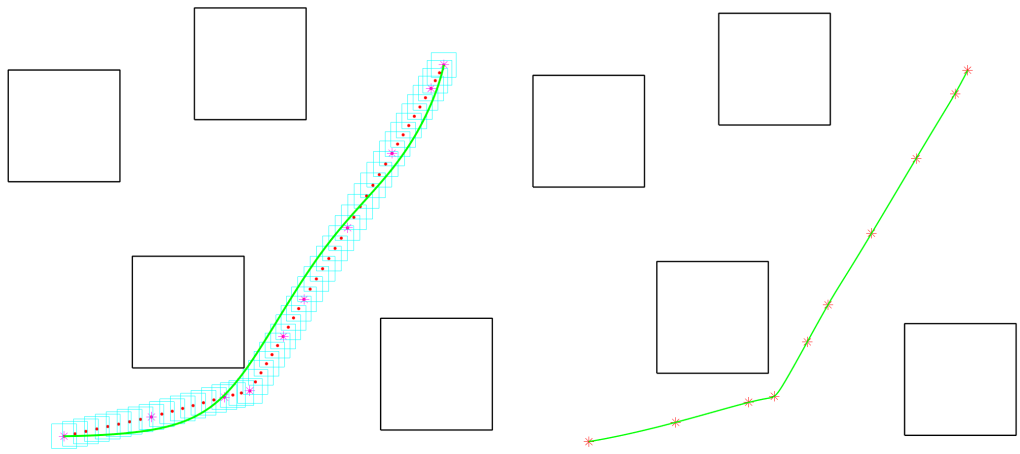
Table 1. Simulation results of decoupled motion planning approach

Algorithm	**Mean Computation Time (s)**	**Mean Normalized Snap**
Info. RRT* + poly	16.19	1.00
BIT* + poly with cor	2.51	1.07
Our method with LSTM, n_c = 20	5.09	1.06
Our method with LSTM, n_c = 16	4.09	1.09
Our method with LSTM, n_c = 12	2.92	1.14
Our method with LSTM, n_c = 8	1.91	1.25
Our method with GRU, n_c = 20	3.27	1.06
Our method with GRU, n_c = 16	2.69	1.08
Our method with GRU, n_c = 12	1.96	1.14
Our method with GRU, n_c = 8	1.52	1.26

Our method effectively circumvents the challenge of waypoint time allocation, a factor known to influence the processing duration of our complete method. Nonetheless, in

specific urban air mobility (UAM) applications, the precise traversal of certain waypoints is imperative, rendering time allocation an essential step in such scenarios. In the classical approach, this issue has been addressed by initializing with estimated segment times and subsequently refining them iteratively using a gradient descent technique according to snap. However, this method tends to be time-intensive, posing challenges for real-time application in UAM contexts. To overcome this limitation, we present an efficient technique designed to substantially expedite the time allocation process. Figure 7 shows trajectories generated by these two methods for the same path.

In this work, we introduce a novel method for the time allocation of waypoints in minimum snap trajectory planning for UAM using a transformer-based model. The process is divided into three key steps: data preparation, model design, and training and validation. During the data preparation phase, trajectory data obtained through multiple iterations of the gradient-descent-based method mentioned earlier are collected. These data include key features such as coordinates, velocity, and acceleration limitations for each waypoint. The time differences between consecutive waypoints are calculated and utilized as the target variable for the model. Feature normalization and sequence transformation are applied to convert the data into a suitable format for processing by the transformer model, and the dataset is split into training, validation, and test sets. Our model design is similar to the transformer encoder architecture used in natural language processing. The input layer is designed to embed waypoint features into a fixed-length vector, incorporating positional encoding based on time to retain the continuity of the temporal sequence. The transformer architecture comprises multiple attention heads to capture long-distance dependencies within the sequence, a feedforward network within each transformer block, and layer normalization following each sub-layer to stabilize the training process. The output layer is configured to predict the time intervals between waypoints. It is a dense layer and uses softplus as the activation function. The training and validation of the model are conducted using a mean squared error (MSE) loss function, optimized using the AdamW optimizer with adjusted learning rates and other hyperparameters. Appropriate batch sizes and iteration counts are determined, and regularization techniques such as dropout are employed to mitigate overfitting. The model's performance is regularly monitored on a validation set, with early stopping implemented to prevent overtraining. Finally, we input the generated time allocation of each waypoint into the piecewise polynomial trajectory generation method, and then use OSQP to solve it to obtain the complete minimum snap trajectory. The response time for the complete process of generating the trajectory is slightly longer than the previous method but still short enough for UAM real-time application, and the trajectory can pass exactly through each preset waypoint. Algorithm 2 shows the details.



(**a**) Situation 1 (**b**) Situation 2

Figure 7. Minimum snap trajectory generated by two methods.

Algorithm 2: Transformer-based Time Allocation for Minimum Snap Trajectory

Input: Set of trajectories T, each trajectory $t \in T$ consists of waypoints $W_t = \{w_1, w_2, \ldots, w_n\}$

Output: Optimized time intervals for each waypoint in T

for *each trajectory* $t \in T$ **do**
- Collect trajectories generated by method using iteration for time allocation
- Extract features F_t from W_t
- Normalize features F_t^{norm}
- Convert F_t^{norm} into sequential format S_t
- Divide S_t into training set S_{train}, validation set S_{val}, and test set S_{test}
- Initialize Transformer Model M
- Define loss function $\mathcal{L}$ as Equation (13)
- Initialize Adam optimizer with learning rate α
- **for** *each epoch* **do**
  - **for** *each batch* $B \subseteq S_{train}$ **do**
    - Apply dropout to B with rate r
    - Predict time intervals $\hat{Y}_B$ using $M(B)$
    - Calculate loss $\mathcal{L}(Y_B, \hat{Y}_B)$
    - Update M using AdamW to minimize $\mathcal{L}$
  - **end**
  - Calculate average loss $\mathcal{L}_{val}$ on S_{val}
  - **if** $\mathcal{L}_{val}$ *has not improved* **then**
    - Early Stopping
    - break
  - **end**
- **end**
- Evaluate M on S_{test}

end

$$\mathcal{L}(Y, \hat{Y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{13}$$

where T is the set of trajectories, where each trajectory is a sequence of waypoints. t is a single trajectory in the set T. W_t is a set of waypoints in trajectory t, denoted as $W_t = \{w_1, w_2, \ldots, w_n\}$. w_i is the ith waypoint in a trajectory, represented by its spatial coordinates and other features like velocity and acceleration limitation. F_t is the feature set extracted from the waypoints in trajectory t. F_t^{norm} represents normalized features of F_t. S_t is the sequential format of the normalized features F_t^{norm}. $S_{train}, S_{val}, S_{test}$ are training, validation, and test sets derived from S_t. M is the transformer-based model used for time interval generation. $\mathcal{L}$ is the loss function, defined as the mean squared error (MSE) in the context of the algorithm. Y_B is time intervals generated by the classical method for a batch B. $\hat{Y}_B$ represents time intervals generated for a batch B by model M. α is the learning rate for the AdamW optimizer. r is the dropout rate used during training to mitigate overfitting. $\mathcal{L}_{val}$ represents the average loss calculated on the validation set S_{val}. $Y = \{y_1, y_2, \ldots, y_N\}$ represents the set of values generated by the classical method. $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N\}$ represents the set of predicted values by the model. N is the number of samples in the set. y_i is the value of the i-th sample in the set of values generated by the classical method. $\hat{y}_i$ is the predicted value for the i-th sample.

4. Approach Based on Generative Adversarial Imitation Learning

In this section, we propose a coupled planning algorithm that generates the time-optimal trajectory in an extremely short computation time via a generative adversarial imitation learning (GAIL) algorithm [11], meanwhile guaranteeing sufficient exploration of the environment. This method also performs motion planning and re-planning on the basis

of existing tree data, but the algorithm for producing tree data is kinodynamic RRT* [16]. There are two main reasons for why kinodynamic RRT* is expensive. First, a complete process requires a large number of iterations. Second, in kinodynamic RRT*, evaluating the connection cost between two states requires solving a two-point boundary value problem (TPBVP), which is usually complicated because the dynamic transition from one state to another is required to be considered. Unlike RRT*, when we want to generate a minimum time trajectory, the step size of kinodynamic RRT* is not the Euclidean distance but the time difference; in addition, the connection between the tree nodes generated by kinodynamic RRT* is not a straight line but a curve trajectory that conforms to kinematic and dynamic constraints [26]. When the goal of kinodynamic RRT* is the minimum time trajectory, using time as the step size can indeed provide some intuitive benefits for solving the TPBVP, especially compared to using Euclidean distance. This simplifies the TPBVP when the exact time difference between two points is known. Furthermore, since the goal of optimization is to minimize time, making the step size correspond to time is intuitive and can help to ensure that the generated path is consistent with the optimization goal. For the overall algorithm, whenever trying to connect a node in the tree to a new random sample point, or when trying to connect a node in the tree to another node in the 'choose parent' and 'rewire' phase, whether there are feasible trajectories satisfying dynamical constraints needs to be considered. This requires solving a TPBVP in almost every iteration, which leads to a very long processing time. Therefore, in order to make the processing time of our algorithm short enough to achieve real-time requirements, we need to bypass the large number of iterations and the challenging two-point boundary value problems.

Each node in kinodynamic RRT* tree data has more attributes than that of RRT*. Each kinodynamic RRT* tree node contains the index of itself, the index of the parent, and the time cost to reach the node from the starting point, as well as the coordinate, velocity, and acceleration of each dimension. If still using the reinforcement-learning-based approach, complex tree data will lead to bloated reward functions, highly raising the difficulty of designing appropriate rewards and violating our intention of pursuing simpler but effective approaches. Hence, we utilize the approach based on GAIL, which has advantages in implementing and tuning hyper-parameters because it does not require the design of reward function compared to RL algorithms.

GAIL requires the agent to interact with the environment but cannot obtain rewards from the environment. In addition, GAIL needs an object to be imitated and, in practice, it needs to collect the decision-making records of the imitated object. In our method, this imitated object is kinodynamic RRT*. Equation (14) shows the decision-making record τ^{real} created by kinodynamic RRT*, where s_t^{real} is consistent every five time steps in two-dimensional motion planning and donates the node index and positions, and a_t^{real} represents the parent selection, single-dimensional velocity, and acceleration of the corresponding s_t^{real}.

$$\tau^{real} = [s_1^{real}, a_1^{real}, s_2^{real}, a_2^{real}, \cdots, s_t^{real}, a_t^{real}, \cdots, s_m^{real}, a_m^{real}] \tag{14}$$

GAIL consists of a generator and a discriminator. The generator is a policy network that makes decisions, and we will use PPO [27] to train the generator. The target of GAIL is to learn a policy network such that the discriminator cannot distinguish whether a decision is made by the policy network or the imitated object. The discriminator is a neural network that will be trained by gradient descent. Training makes the discriminator more accurate in determining where decisions are coming from, and this adversarial approach to mutual progress is the main idea of GAIL.

As mentioned previously, PPO will be used as our algorithm for training the policy network. Similar to the method in Section 3, we cannot use the entire tree data as input to the policy network, since tree data contain many nodes, and each node has several attributes, which makes tree data actually a very-high-dimensional vector that PPO is unable to process. Therefore, we still apply a recurrent neural network as the policy

network (known as the generator) of GAIL for handling this problem. Figure 8 shows the details of the RNN policy network.

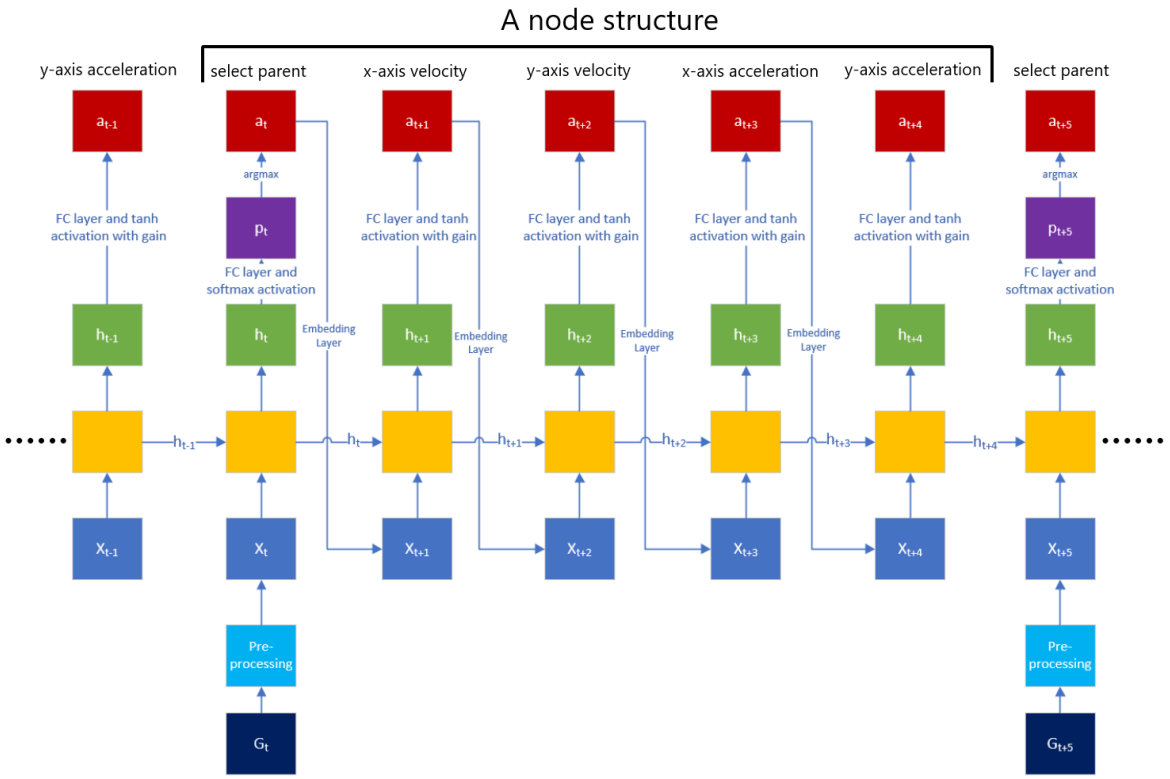


Figure 8. Using recurrent neural networks as policy network of GAIL.

Using the RNN policy network is the key to bypassing the large number of iterations and the challenging two-point boundary value problems. Solving the TPBVP is avoided by neural network inference. Furthermore, the tree is not built by iterations but by the sequential RNN tokens. The RNN policy network of GAIL still contains node structures whose number is equivalent to the number of nodes in the tree data. Each node structure can be regarded as a set of single-input multi-output RNN sequences. In two-dimensional motion planning, each node structure has five tokens and each token outputs an action.

The design of the first token in each node structure is very similar to that in Section 3: the input X_t is pre-processed G_t, which contains a node index and position, and its output a_t is an m_c-dimensional one-hot vector representing the selection of the parent node. The value of m_c is a manually tuned hyper-parameter that affects the speed of convergence of the reinforcement learning algorithm and the results of training. Before the start of training, the algorithm will iterate over the tree data and update tree data by adding m_c nearest nodes for each tree node. Closer nodes also represent shorter arrival times when there are zero initial velocity and acceleration. Equations (15), (17) and (18) display the pre-processing and input formation, probability distribution generation, and action determination of the first token.

$$X_t = E_{\text{index}} \cdot \text{one-hot}(F_t^{\text{index}}) + E_{\text{pos}} \cdot F_t^{\text{pos}} \tag{15}$$

$$h_t = \text{GRU}(X_t, h_{t-1}; \theta) \tag{16}$$

$$p_t = \text{softmax}(W_p \cdot h_t + b_p) \tag{17}$$

$$a_t = \text{one-hot}(\text{argmax}(p_t)) \tag{18}$$

where X_t is the input to the first token at time t; E_{index} is the embedding matrix for the one-hot encoding of the node index; $\text{GRU}(X_t, h_{t-1}; \theta)$ represents the GRU function taking the current input X_t, the previous hidden state h_{t-1}, and GRU parameters θ; $\text{one-hot}(F_t^{\text{index}})$ represents the one-hot encoding of the node index at time t; E_{pos} is the embedding matrix for the node position; F_t^{pos} stands for the position (coordinates) of the node at time t; p_t is the probability distribution output by the first token at time t; W_p denotes the weight matrix for generating the probability distribution; h_t and h_{t-1} represent the hidden state of the RNN from the current and previous time step; b_p is the bias vector associated with the probability distribution; a_t is the one-hot encoded action representing the selection of the parent node at time t; $\text{argmax}(p_t)$ denotes the function that returns the index of the maximum value in p_t, indicating the most probable parent node selection.

The output of the second token of each node is a single-dimensional velocity whose value depends on the position of the node itself and the choice of the parent node, and is influenced by other nodes. The hidden layer of the RNN contains information of the index and position of the current node, and historical information of some other nodes. The input of the second token X_{t+1} is the output of the first token (also known as the selection of the parent) after going through the embedding layer. The output of the third token of each node is also a single-dimensional velocity whose value depends on the position of the node itself, the choice of the parent node, and the obtained velocity of another dimension, and is influenced by other nodes. The input of the third token X_{t+2} is therefore the output of the second token after going through the embedding layer. The input of the fourth token is the output of the third token after going through the embedding layer since the output of the fourth token is a single-dimensional acceleration that also depends on velocity. The input of the fifth token is similar to that of previous tokens. In the second to fifth tokens of each node, we utilize gained tanh as the activation function to output the velocity and acceleration of each dimension. tanh is a center-symmetric function and monotonically increasing in a certain range, so the RNN policy network has equivalent performance regardless of whether the output velocity and acceleration of each dimension are positive or negative. Gain is determined in accordance with velocity and acceleration constraints. Equations (19) and (20) describe the process of the second to fifth token.

$$X_{t+i-1} = E_i \cdot a_{t+i-1} \quad \text{for } i = 2, 3, 4, 5 \tag{19}$$

$$a_{t+i-1} = gain \cdot \tanh((W_{v,a} \cdot h_{t+i-1} + b_{v,a})) \quad \text{for } i = 2, 3, 4, 5 \tag{20}$$

where X_{t+i-1} is the input for the i-th token, computed from the output of the previous token ($i = 2, 3, 4, 5$); E_i represents the embedding matrix specific to the i-th token; a_{t+i-1} denotes the output of the i-th token, which could be a single-dimensional velocity or acceleration; $W_{v,a}$ is the weight matrix for the velocity or acceleration outputs; h_{t+i-1} represents the hidden state of the RNN from the current time step for the i-th token; $b_{v,a}$ is the bias vector associated with the velocity or acceleration outputs; tanh is the hyperbolic tangent activation function, used here for its symmetric properties; $gain$ is a factor used to adjust the output within the constraints of velocity and acceleration.

It is worth mentioning that these four embedding layers between every two tokens in the structure of each node are different as they perform different tasks, whereas all node structures share the set of these four embedding layers because they separately execute the same tasks in each node. The output of the embedding layer will be concatenated with a corresponding type embedding vector [0][1][2][3][4] for 'select parent' and different dimensional velocities and accelerations before being input into X_{t+i}. Similar to the approach proposed in Section 3, we do not use a standard RNN, in fact, but use a GRU (Equation (16)) to avoid the problem of gradient vanishing and gradient explosion, where each token has two hidden layers. Equation (21) displays the decision-making record τ^{fake} created by the RNN policy network, where s_t^{fake} is also consistent every five time steps in two-dimensional motion planning and donates the node index and positions, a_t^{fake} represents the parent

selection, single-dimensional velocity, and acceleration of the corresponding s_t^{fake}, and m is the number of tokens in the RNN policy network.

$$\tau^{fake} = [s_1^{fake}, a_1^{fake}, s_2^{fake}, a_2^{fake}, \cdots, s_t^{fake}, a_t^{fake}, \cdots, s_m^{fake}, a_m^{fake}] \tag{21}$$

The discriminator of GAIL is a neural network whose structure is shown in Figure 9. The essence of the discriminator is actually a binary classifier. Its output value $D(s_t, a_t|\phi)$ represents the judgment of authenticity, where ϕ is the neural network parameter. The closer the output is to 1, the more true it is—that is, the action is produced by kinodynamic RRT*—and the closer the output is to 0, the more false it is; that is, it is generated by the policy network.

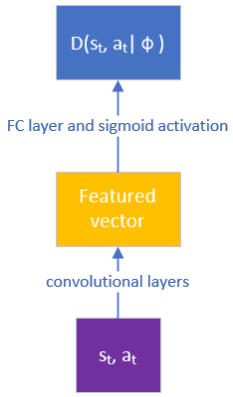


Figure 9. Discriminator of GAIL.

The goal of training GAIL is to make the generator (also known as the RNN policy network) produce a record of decisions that are as good as those of the imitated object. At the end of training, the discriminator cannot distinguish between the generator's decision records and the imitated object's decision records. Therefore, while training the generator, we need to train the discriminator simultaneously, and only if the discriminator is good enough will the generator that can fool it obtain satisfactory results. When training the discriminator, we encourage it to make more accurate judgments. We want the discriminator to know that (s_t^{real}, a_t^{real}) is true, so $D(s_t^{real}, a_t^{real}|\phi)$ should be encouraged to be as large as possible. We want the discriminator to know that (s_t^{fake}, a_t^{fake}) is false, so $D(s_t^{fake}, a_t^{fake}|\phi)$ should be encouraged to be as small as possible. Equation (22) defines the loss function.

$$F(\tau^{real}, \tau^{fake}|\phi) = \frac{1}{m}\sum_{t=1}^{m} \ln\left[1 - D(s_t^{real}, a_t^{real}|\phi)\right] + \frac{1}{m}\sum_{t=1}^{m} \ln\left[D(s_t^{fake}, a_t^{fake}|\phi)\right. \tag{22}$$

We expect the loss function to be as small as possible so that we can use gradient descent to update parameters ϕ, which are shown in function (23).

$$\phi \leftarrow \phi - \beta \nabla_\phi F(\tau^{real}, \tau^{fake}|\phi) \tag{23}$$

where β donates the learning rate and ∇_ϕ is the gradient. The larger the output $D(s_t^{fake}, a_t^{fake}|\phi)$ of the discriminator, the more similar the decisions generated by the RNN policy network are to those generated by kinodynamic RRT*, and the more successful the imitation learning; therefore, we substitute the reward u_t with Equation (24).

$$u_t = \ln D(s_t^{fake}, a_t^{fake}|\phi) \tag{24}$$

Then, according to u_t, we can apply the PPO algorithm to train the RNN policy network π_θ of GAIL with Equations (25) and (26).

$$L^{clip}(\theta) = \mathbb{E}(min(r(\theta)A_{\theta old}([X_t, h_{t-1}], a_t)), clip(r(\theta), 1-\epsilon, 1+\epsilon)A_{\theta old}([X_t, h_{t-1}], a_t) \quad (25)$$

$$r_t(\theta) = \frac{\pi_\theta((a_t|[X_t, h_{t-1}]))}{\pi_{\theta old}((a_t|[X_t, h_{t-1}]))} \quad (26)$$

where ϵ is a hyper-parameter used to control the clipping ratio. $\mathbb{E}[\cdot]$ denotes the expectation operator. $A_{\theta old}[\cdot]$ is the advantage function under the old policy. $r(\theta)$ denotes the probability ratio.

Algorithm 3 shows the details of training process. The training performance of the RNN policy network acting as the generator of GAIL is strongly dependent on the hyper-parameters of the PPO algorithm. The discriminator that outputs rewards for the PPO algorithm is a neural network, which is also dependent on the selection of hyper-parameters. Therefore, optimizing hyper-parameters is essential since this approach is highly sensitive to hyper-parameters. Among the commonly used hyper-parameter optimization methods for machine learning, Bayesian hyper-parameter optimization methods have shown advantages in both accuracy and efficiency compared to grid search and random search [38] as this optimization problem has no explicit objective function expression.

Algorithm 3: Train RNN policy network using GAIL with PPO updates

Initialization:
Initialize RNN policy network π_θ and GAIL discriminator D_ϕ;
Initialize environment and decision-making algorithm (Kinodynamic RRT*);
Define loss functions for π_θ and D_ϕ;
Training Loop:
while *not converged* **do**
  Data Generation:
  Generate decision record τ^{real} using Kinodynamic RRT*;
  Generate decision record τ^{fake} using RNN policy network π_θ;
  Discriminator Update:
  for *each timestep t in* τ^{real} *and* τ^{fake} **do**
    Compute $D_\phi(s_t^{real}, a_t^{real})$ and $D_\phi(s_t^{fake}, a_t^{fake})$;
    Update ϕ by minimizing $F(\tau^{real}, \tau^{fake}|\phi)$;
  end
  Policy Network Update:
  Calculate reward $u_t = \ln D_\phi(s_t^{fake}, a_t^{fake})$;
  Estimate advantage function $A_\theta([X_t, h_{t-1}], a_t)$;
  Compute probability ratio $r_t(\theta)$;
  Update θ using clipped objective $L^{clip}(\theta)$;
  if *fine-tuning needed* **then**
    Adjust hyperparameters and learning rates;
  end
  Convergence Check:
  Evaluate training performance and convergence criteria;
end

The core in the Bayesian optimization method includes a surrogate model and an acquisition function. In our approach, the Gaussian process model (GP) is applied as a surrogate model. The Gaussian process is a joint distribution of a series of random variables that obey the normal distribution. Based on this model, the distribution of the objective function $f(x)$ can be estimated from the mean value $\mu(x)$, and the uncertainty of each position can be obtained from variance $\sigma(x)$, where x is a set of hyper-parameters

and $f(x)$ is the mean ratio of the time cost of trajectories generated by the RNN policy network and the benchmark algorithm kinodynamic RRT*, respectively, in the Monte Carlo simulation. In detail, we randomly generated 1000 sets of different starting points and target points, as well as obstacle positions, for the Monte Carlo simulation. The Euclidean distances between the starting points and the target points are greater than a threshold. Our proposed method and the baseline classical method will, respectively, generate a trajectory in each environment and then obtain the ratio of the time cost of the two trajectories, finally obtaining the mean ratio of the time cost of 1000 environments.

After constructing the surrogate model, the acquisition function is used to determine the next set of hyper-parameters, trading off exploration (sampling from high-uncertainty areas) and exploitation (sampling from high-value areas). The process will be iterated multiple times until it is close to the global optimum. The next set of hyper-parameters is determined as Equation (27).

$$x_{n+1} = \arg\max_{x} g(x|X) \tag{27}$$

where $g(x|X)$ is the acquisition function and X is the n observation points from $f(x)$ so far.

The expected improvement (EI) is a common choice as the acquisition function and it can be evaluated under the GP model as Equation (28) [39]:

$$EI(x) = \begin{cases} (\mu(x) - f(x^+ - \xi))\Phi(Z) + \sigma(x)\phi(Z), & \sigma(x) > 0 \\ 0, & \sigma(x) = 0 \end{cases} \tag{28}$$

where

$$Z = \begin{cases} \frac{\mu(x) - f(x^+ - \xi)}{\sigma(x)}, & \sigma(x) > 0 \\ 0, & \sigma(x) = 0 \end{cases} \tag{29}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and standard normal distribution function. The first term $(\mu(x) - f(x^+ - \xi))\Phi(Z)$ in Equation (28) is used for exploitation, the second term $\sigma(x)\phi(Z)$ is used for exploration, and the parameter ξ determines the proportion of exploration.

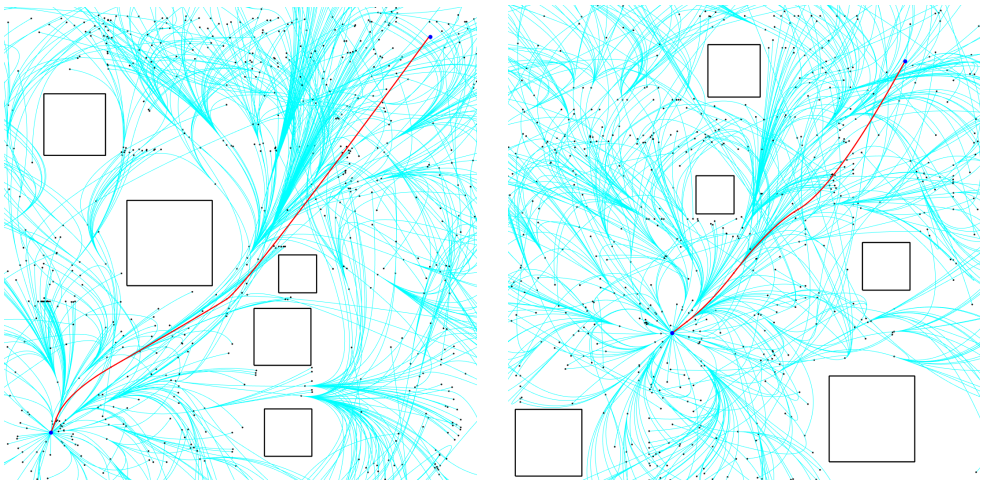
Additionally, when the vehicle that needs to perform motion planning changes—that is, the speed and acceleration limits of the vehicle change—we do not need to entirely retrain an RNN policy network but perform fine-tuning on the basis of the RNN policy network trained previously. Fine-tuning will perform the following five steps:

- Changing values of the gain of the gained tanh activation function;
- Setting independent learning rates for the two hidden layers of the RNN policy network;
- Fine-tuning by GAIL with fewer episodes;
- Applying a Bayesian hyper-parameter optimization method;
- Using the greedy soup receipt of the model soups method [40].

Model soups is a method used to average the weights of multiple models fine-tuned with different hyper-parameter configurations, resulting in improving accuracy and robustness without incurring any additional inference or memory costs. The greedy soup is constructed by sequentially adding each model as a potential ingredient in the soup, and only keeping the model in the soup if the performance on a held-out validation set evaluated by Monte Carlo simulation results improves. Before running this procedure, we sort the models in decreasing order of validation set accuracy so the greedy soup can be no worse than the best individual model on the held-out validation set [40].

Figure 10 shows the results produced by the approach based on GAIL. It can be found that our algorithm successfully generates a collision-free tree with curve connections and

obtains a safe trajectory. Figure 10a shows planning before flying and Figure 10b shows re-planning on the fly. In addition, a GRU is utilized as the RNN policy network, and cuDNN acceleration is applied. Based on tree data of 1200 nodes, an extremely short computation time is required to generate the trajectory using the trained policy network, which is only around three seconds on our device, while it takes more than two minutes to complete 1200 iterations using kinodynamic RRT*.



(**a**) Situation 1 (**b**) Situation 2

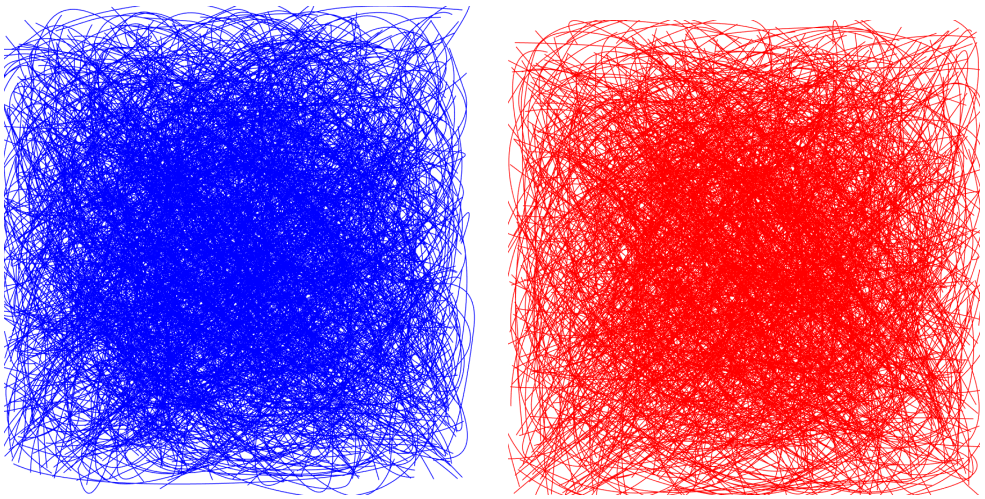
Figure 10. GAIL-based motion planning in environment 1 containing 5 dynamic obstacles.

Table 2 displays the average processing time and average normalized vehicle travel time of the benchmark method and the three methods used for comparison under 50 different situations of environment 1, as well as the average processing time and mean normalized vehicle travel time of our approach's Monte Carlo simulations. Kinodynamic RRT* is our benchmark algorithm. This algorithm needs a large number of iterations and requires solving a TPBVP in almost every iteration, which leads to a very long processing time. Learning-based kinodynamic RRT* [41] replaces solving the TPBVP in the 'choose parent' phase and 'rewire' phase with neural network inference, thus significantly accelerating the algorithm. However, the processing time of this algorithm is still not short enough as a large number of iterations is still required. The stable sparse RRT (SST) [42] algorithm achieves optimality guarantees without requiring optimal boundary value problem solutions, and only requires forwarding dynamically propagating random actions from a selected node, but such a technique is prone to 'wandering' through the state space and taking a long time to identify a solution [43]. Since our approach bypasses the large number of iterations and the challenging two-point boundary value problems, the processing time of our approach is significantly short, and the average vehicle travel time is also close to the baseline algorithm. Moreover, our approach produces significantly better trajectories than the 'directly imitate trajectory using kinodynamic RRT*' method, which reflects the necessity and superiority of the RNN policy network.

Figure 11a shows the Monte Carlo simulation results produced by the approach based on GAIL. After applying a Bayesian hyper-parameter optimization method, it can be found that the properties of the resulting trees, including the velocity, acceleration, and time that it takes to reach the node, are close to the results of kinodynamic RRT*. Figure 11b displays the Monte Carlo simulation results for fine-tuning based on the original generated RNN policy network. In this scene, the velocity limit is increased by half and the acceleration limit is doubled. As shown in Table 2, with Bayesian hyper-parameter optimization and greedy soup, the performance of the fine-tuned policy network for a new vehicle is close to that of the original policy network.

Table 2. Monte Carlo simulation results of coupled motion planning approach and results of algorithms for comparison and benchmark algorithm in environment 1.

Algorithm	Mean Computation Time (s)	Mean Normalized Time to Reach the Goal Position
Kinodynamic RRT*	121	1.00
Extract trajectory from SST	18.99	1.04
Directly imitate trajectory using kinodynamic RRT*	<1	1.47
Learning-based kinodynamic RRT*	21.82	1.02
Original policy network using our method	3.24	1.09
Fine-tuned policy network using our method	3.31	1.10



(**a**) Original RNN policy network (**b**) Fine-tuned RNN policy network

Figure 11. Monte Carlo simulation for GAIL-based motion planning and fine-tuning.

Figure 12 shows a trajectory generated by our approach based on GAIL in environment 2, which includes five dynamic obstacles and three static obstacles. Table 3 displays the average processing time and average normalized vehicle travel time of the benchmark method, three methods used for comparison, and our approaches under 50 different situations of environment 2. It can be found that our methods have relatively better performance in environments with more obstacles and fewer effective samples.

Table 3. Monte Carlo simulation results of coupled motion planning approach and results of algorithms for comparison and benchmark algorithm in environment 2.

Algorithm	Mean Computation Time (s)	Mean Normalized Time to Reach the Goal Position
Kinodynamic RRT*	109	1.00
Extract trajectory from SST	17.39	1.04
Directly imitate trajectory using kinodynamic RRT*	<1	1.41
Learning-based kinodynamic RRT*	23.09	1.03
Original policy network using our method	3.11	1.06
Fine-tuned policy network using our method	3.16	1.06

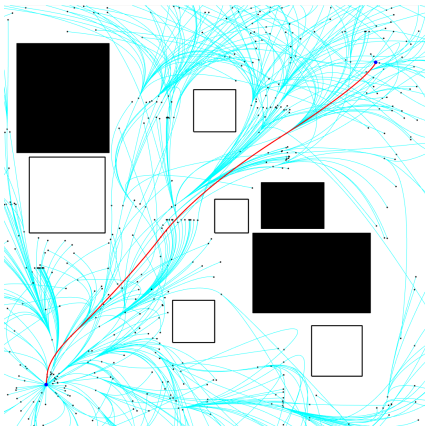


Figure 12. GAI- based motion planning in environment 2 containing 5 dynamic obstacles and 3 static obstacles.

5. Conclusions

In this paper, we presented two novel approaches in the face of challenges of motion planning and re-planning in UAM operations in the presence of the uncertainty of the environment, kinematic and dynamic constraints of the vehicle, and possible emergency situations. Both of our methods utilize the tree data generated by the RRT*-based algorithm, which ensures ample exploration of the configuration space. Moreover, we apply neural network inference instead of a math-based algorithm to update the tree data, which makes our approaches suitable for real-time application in UAM operations due to their short computation times.

The first approach is a decoupled approach in which we design a policy network based on a recurrent neural network for the reinforcement learning algorithm in terms of addressing three issues. These three issues are that (a) the input vector has a significantly high dimension, (b) the input and output vectors of consecutive steps have different dimensions, and (c) the agent is unable to observe the global environment. Afterward, we combine an online trajectory generation algorithm to obtain the minimal snap trajectory for the vehicle. The simulation results demonstrate that this method can generate high-quality trajectories in an extremely short computation time. Furthermore, it enables on-the-fly motion re-planning, where the re-planned trajectory maintains continuity for the executed trajectory.

We also propose a coupled method that generates a time-optimized trajectory. We design a single-input multiple-output RNN policy network for this method, and utilize GAIL to train the policy network so as to generate similar decisions to kinodynamic RRT*. The results show that this approach can update and generate collision-free global tree data in a very short time, and we use Bayesian hyperparameter optimization to solve the problem of the results of this method being extremely dependent on the hyper-parameter configuration. In addition, we design a model soup-based fine-tuning method for the problem of changing to vehicles with different velocities and accelerations, which avoids the need to retrain a policy network; in addition, the performance of the fine-tuned model is close to the original model.

Author Contributions: Conceptualization, G.I.; methodology, J.L. and B.Y.; software, J.L.; validation, J.L. and B.Y.; formal analysis, J.L., B.Y., G.I. and A.T.; resources, G.I. and A.T.; data curation, J.L.; writing—original draft preparation, J.L.; writing—review and editing, B.Y., G.I. and A.T.; visualization, J.L.; supervision, G.I. and A.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

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

References

1. Bauranov, A.; Rakas, J. Designing airspace for urban air mobility: A review of concepts and approaches. *Prog. Aerosp. Sci.* **2021**, *125*, 100726. [CrossRef]
2. Yu, X.; Zhang, Y. Sense and avoid technologies with applications to unmanned aircraft systems: Review and prospects. *Prog. Aerosp. Sci.* **2015**, *74*, 152–166. [CrossRef]
3. Murray, C.W.; Ireland, M.; Anderson, D. On the response of an autonomous quadrotor operating in a turbulent urban environment. In Proceedings of the AUVSI's Unmanned Systems Conference, Orlando, FL, USA, 12–15 May 2014.
4. Logan, M.J.; Bird, E.; Hernandez, L.; Menard, M.; Moore, A.; Balachandran, S.; Young, S.D.; Dill, E.T.; Glaab, L.J.; Munoz, C.; et al. Operational Considerations of Small UAS in Urban Canyons. In Proceedings of the AIAA Scitech 2020 Forum, Orlando, FL, USA, 6–10 January 2020; p. 1483.
5. Pang, B.; Ng, E.M.; Low, K.H. UAV Trajectory Estimation and Deviation Analysis for Contingency Management in Urban Environments. In Proceedings of the AIAA Aviation 2020 Forum, Virtual, 15–19 June 2020; p. 2919.
6. Radio Technical Commission for Aeronautics. *Minimum Aviation System Performance Standards for Automatic Dependent Surveillance Broadcast (ADS-S)*; RTCA, Incorporated: Washington, DC, USA, 2002.
7. Dill, E.T.; Young, S.D.; Hayhurst, K.J. SAFEGUARD: An assured safety net technology for UAS. In Proceedings of the 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), Sacramento, CA, USA, 25–29 September 2016; pp. 1–10.
8. LaValle, S.M. *Planning Algorithms*; Cambridge University Press: Cambridge, UK, 2006.
9. Medsker, L.R.; Jain, L.C. Recurrent neural networks. *Des. Appl.* **2001**, *5*, 64–67.
10. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
11. Ho, J.; Ermon, S. Generative adversarial imitation learning. *arXiv* **2016**. [CrossRef]
12. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *arXiv* **2017**. [CrossRef]
13. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [CrossRef]
14. Lou, J.; Yuksek, B.; Inalhan, G.; Tsourdos, A. An RRT* Based Method for Dynamic Mission Balancing for Urban Air Mobility Under Uncertain Operational Conditions. In Proceedings of the 2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC), San Antonio, TX, USA, 3–7 October 2021; pp. 1–10.
15. LaValle, S.M. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*; Iowa State University: Ames, IA, USA, 1998.
16. Webb, D.J.; Van Den Berg, J. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany , 6–10 May 2013; pp. 5054–5061.
17. Chiang, H.T.L.; Hsu, J.; Fiser, M.; Tapia, L.; Faust, A. RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies. *IEEE Robot. Autom. Lett.* **2019**, *4*, 4298–4305. [CrossRef]
18. Mellinger, D.; Kumar, V. Minimum snap trajectory generation and control for quadrotors. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 2520–2525.
19. Bry, A.; Richter, C.; Bachrach, A.; Roy, N. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *Int. J. Robot. Res.* **2015**, *34*, 969–1002. [CrossRef]
20. Stellato, B.; Banjac, G.; Goulart, P.; Bemporad, A.; Boyd, S. OSQP: An operator splitting solver for quadratic programs. *Math. Program. Comput.* **2020**, *12*, 637–672. [CrossRef]
21. Burke, D.; Chapman, A.; Shames, I. Generating minimum-snap quadrotor trajectories really fast. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020; pp. 1487–1492.
22. Meng, L.; Qing, S.; Jun, Z.Q. UAV path re-planning based on improved bidirectional RRT algorithm in dynamic environment. In Proceedings of the 2017 3rd International Conference on Control, Automation and Robotics (ICCAR), Nagoya, Japan, 24–27 April 2017; pp. 658–661.
23. Dong, Z.; Chen, Z.; Zhou, R.; Zhang, R. A hybrid approach of virtual force and A* search algorithm for UAV path re-planning. In Proceedings of the 2011 6th IEEE Conference on Industrial Electronics and Applications, Beijing, China, 21–23 June 2011; pp. 1140–1145.
24. Ng, A.Y.; Russell, S. Algorithms for inverse reinforcement learning. *Icml* **2000**, *1*, 2.
25. Sadhu, A.K.; Shukla, S.; Sortee, S.; Ludhiyani, M.; Dasgupta, R. Simultaneous Learning and Planning using Rapidly Exploring Random Tree* and Reinforcement Learning. In Proceedings of the 2021 International Conference on Unmanned Aircraft Systems (ICUAS), Athens, Greece, 15–18 June 2021; pp. 71–80.
26. Lou, J.; Yuksek, B.; Inalhan, G.; Tsourdos, A. Real-time on-the-fly Motion planning via updating tree data of RRT* using Neural network inference. In Proceedings of the AIAA SCITECH 2023 Forum, Washington, DC, USA, 23–27 January 2023; p. 0786.
27. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
28. Hammersley, J. *Monte Carlo Methods*; Springer Science and Business Media: Berlin/Heidelberg, Germany, 2013.
29. Chow, Y.; Nachum, O.; Ghavamzadeh, M. Path consistency learning in tsallis entropy regularized mdps. In Proceedings of the International Conference on Machine Learning PMLR, Baltimore, MD, USA, 17–23 July 2018; pp. 979–988.

30. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning PMLR, Lille, France, 6–11 July 2015; pp. 1889–1897.
31. Yuksek, B.; Demirezen, M.U.; Inalhan, G.; Tsourdos, A. Cooperative Planning for an Unmanned Combat Aerial Vehicle Fleet Using Reinforcement Learning. *J. Aerosp. Inf. Syst.* **2021**, *18*, 739–750. [CrossRef]
32. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
33. Chen, J.; Su, K.; Shen, S. Real-time safe trajectory generation for quadrotor flight in cluttered environments. In Proceedings of the 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), Zhuhai, China, 6–9 December 2015; pp. 1678–1685.
34. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the International Conference on Machine Learning PMLR, Atlanta, GA, USA, 17–19 June 2013; pp. 1310–1318.
35. Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
36. Gammell, J.D.; Srinivasa, S.S.; Barfoot, T.D. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, 14–18 September 2014; pp. 2997–3004.
37. Gammell, J.D.; Barfoot, T.D.; Srinivasa, S.S. Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search. *Int. J. Robot. Res.* **2020**, *39*, 543–567. [CrossRef]
38. Wu, J.; Chen, X.Y.; Zhang, H.; Xiong, L.D.; Lei, H.; Deng, S.H. Hyperparameter optimization for machine learning models based on Bayesian optimization. *J. Electron. Sci. Technol.* **2019**, *17*, 26–40.
39. Zhang, D.; Ma, G.; Deng, Z.; Wang, Q.; Zhang, G.; Zhou, W. A self-adaptive gradient-based particle swarm optimization algorithm with dynamic population topology. *Appl. Soft Comput.* **2022**, *130*, 109660. [CrossRef]
40. Wortsman, M.; Ilharco, G.; Gadre, S.Y.; Roelofs, R.; Gontijo-Lopes, R.; Morcos, A.S.; Namkoong, H.; Farhadi, A.; Carmon, Y.; Kornblith, S.; et al. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In Proceedings of the International Conference on Machine Learning PMLR, Baltimore, MD, USA, 17–23 July 2022; pp. 23965–23998.
41. Zheng, D.; Tsiotras, P. Sampling-based kinodynamic motion planning using a neural network controller. In Proceedings of the AIAA Scitech 2021 Forum, Virtual, 11–15 January 2021; p. 1754.
42. Li, Y.; Littlefield, Z.; Bekris, K.E. Asymptotically optimal sampling-based kinodynamic planning. *Int. J. Robot. Res.* **2016**, *35*, 528–564. [CrossRef]
43. Allen, R.; Pavone, M. A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance. In Proceedings of the AIAA Guidance, Navigation, and Control Conference, San Diego, CA, USA, 4–8 January 2016; p. 1374.

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



https://dspace.org/

Real-time on-the-fly motion planning for urban air mobility via updating tree data of sampling-based algorithms using neural network inference

Lou, Junlin; Yuksek, Burak; Inalhan, Gokhan; Tsourdos, Antonios

2024-01-01
MDPI
https://doi.org/10.3390/aerospace11010099
Lou J, Yuksek B, Inalhan G, Tsourdos A. (2024) Real-time on-the-fly motion planning for urban air mobility via updating tree data of sampling-based algorithms using neural network inference. Aerospace, Volume 11, Issue 1, January 2024, Article number 99

Downloaded from a DSpace repository. https://dspace.org/