Real-time on-the-fly Motion planning via updating tree data of RRT* using Neural network inference

Junlin Lou*, Burak Yuksek†, Gokhan Inalhan‡, Antonios Tsourdos§
School of Aerospace, Transport and Manufacturing, Cranfield University, United Kingdom, MK43 0AL

In this study, we consider the problem of motion planning for urban air mobility applications to generate minimal snap trajectory and trajectory that cost minimal time to reach 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 We proposed is a decoupled method that includes designing a policy network based on recurrent neural network for the reinforcement learning algorithm, and then combine 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 generative adversarial imitation learning algorithm for training recurrent neural network based policy network and generating time optimized trajectory. Simulation results show that our approaches have 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 re-planned trajectories maintain continuity for executed trajectory. To the best of our knowledge, we propose one of the first approaches enabling to perform on-the-fly update of final landing position, and to optimize path and trajectory in real-time while keeping explorations in the environment.

I. Introduction

Urban Air Mobility (UAM) is an air transport system concept operated in urban airspace, which has been proposed in recent years and has received extensive attention in the aerospace and transportation industries. The objective of UAM is to transport people and cargo more effectively by utilising specific vehicles such as electric vertical take-off and landing (eVTOL) aircraft and small unmanned air vehicle (sUAV). These vehicles are highly automated to navigate from takeoff phase to landing phase without a human operator aboard. However, such a transportation model is subject to numerous complicated issues in terms of environment, urban design factors, even factors of UAM platform itself including buildings [1], high density of flying vehicles [2], micro-weather patterns [3] [4], activities or sudden disasters in cities [1], and communication, navigation and surveillance quality and availability of the service [5][6]. These factors lead to static, dynamic, and uncertain flight-restricted areas, obstacles, and geofences[7] in urban airspace, which cause three main challenging issues for highly-automated aerial vehicles on their safety. Firstly, UAM vehicles need to perform real-time on-the-fly re-planning due to uncertainties in environment. Secondly, kinematics and dynamics of the system are nonlinear and subject to constraints on the aerial vehicle in accordance with the aerodynamic characteristics and power configuration, which results in the limitation of cruise velocity and acceleration of vehicle. Thirdly, it is required to generate a feasible trajectory rapidly when the landing vertiport changes due to the congestion of the original target vertiport or the vehicle has to conduct contingency operations such as emergency landing on-the-fly. In this paper, we address these challenges by combining approaches from sampling-based path planning [8], trajectory optimization, recurrent neural networks (RNN)[9], reinforcement learning (RL)[10] and generative adversarial imitation learning (GAIL)[11] in order to perform real-time on-the-fly motion re-planning in the face of uncertainty in airspace.

In this study, we proposed two approaches that are coupled and decoupled algorithms for path planning and trajectory optimization applications, respectively. Compared with past proposed classical methods that can achieve the same effect, our methods have significantly shorter computation time and therefore enable real-time performance in UAM operation. Vehicle with coupled algorithm is able to reach the destination as soon as possible under constraints of kinematics

*Ph.D. Candidate, junlin.lou@cranfield.ac.uk, Student Member of AIAA.
†Postdoctoral Research Fellow, burak.yuksek@cranfield.ac.uk, Professional Member of AIAA.
‡BAE Systems Chair, Professor of Autonomous Systems and Artificial Intelligence, inalhan@cranfield.ac.uk, Associate Fellow of AIAA.
§Professor, Head of the Centre of Autonomous and Cyber-Physical Systems, a.tsourdos@cranfield.ac.uk, Senior Member of AIAA.

and dynamics. Decoupled algorithm guarantees the optimality of straight path fragments and allows aerial vehicle with constraints to arrive at destination at scheduled time. Both approaches are based on the tree data generated by sampling-based path planning algorithm RRT* or its extended algorithms. RRT* features as Asymptotic Optimality and Probabilistic Completeness[12], which means adequate and intensive exploration of configuration space is necessary to return a feasible and optimal path. Since sufficient exploration time is a prerequisite for RRT*-based algorithms, it is almost impossible for them to plan paths in real-time. The benefit of utilizing tree data of RRT* is that the algorithm can update and optimize the flight path with previous explorations, so that the only requirement is to consider the approaches of updating tree data. Tree data generated by RRT* algorithm which contains a certain number of nodes, the attributes of each node include index of itself, coordinates, cost to tree root, index of its parent node and may also include velocities and accelerations. For current tree structure, all nodes have optimal paths from tree root. Tree root is defined as current position of vehicle or the position of departure vertiport. Our objective is real-time update of the tree data in dynamic environments, guaranteeing optimized attributes and connections (also known as index of parent) of nodes on the tree as well as feasibility and optimality of the flight path to destination. If the local graph-rewiring technique of RRT* are applied to update the connections, an amount of forward and backward propagation need to be performed and real-time update would be almost impossible. Especially when on-the-fly update is required, root moving needs to be executed and tree connections need to be optimized which results in high computational cost. Our approaches use RL policy or GAIL generator to guide the update of tree connection and attributes of nodes, which make the algorithms faster because expensive computations are replaced with simple neural network inference process. Moreover, another advantage of our approaches is that they can change destination of the aerial vehicle at any time and generate a feasible and optimal path from vehicle's current position to the new destination in real-time. This is due to intensive exploration resulting in tree nodes throughout the configuration space and tree data's feature that paths from the root to all nodes on the tree are optimized. To the best of our knowledge, this paper proposes one of the first approaches that is able to perform on-the-fly update of destination and optimize path and trajectory in real-time while keeping all explorations.

The rest of the paper is organized as follows; in Section II, we mention about related works that contributed to our research, as well as the shortcomings and limitations of some other approaches that achieve motion planning or re-planning. In Section III, we propose the decoupled planning algorithm that generates minimum snap trajectory in extremely short computation time via reinforcement learning and piecewise polynomial trajectory generation algorithm, meanwhile guaranteeing sufficient exploration of the environment. In Section IV, we present the coupled planning algorithm using generative adversarial imitation learning algorithm, which can directly generate time optimized trajectory with significantly short computation time while ensuring sufficient exploration of configuration space as well. In Section V, concluding remarks of this paper are given.

II. Related Works

Our approaches draw inspiration from the vast body of previous research. Sampling based path planning algorithm rapidly exploring random tree [13] returns collision-free feasible path. Its variant RRT* [12] enables the cost of solution to converge towards the optimum. Kinodynamic RRT* [14] extends RRT* by applying kinematic and dynamic constraints and generates time-optimal trajectory. RL-RRT [15] is currently state-of-the-art kinodynamic planners in the aspect of efficiency, which introduces RL policy into the combination of RRT and a kinodynamic motion planner, being able to generate high quality time-optimal trajectory. In decoupled approach of motion planning, trajectory generation is on the basis of a sequence of waypoints which are generated by the path planner. The algorithm of Minimum snap trajectory for quadrotor[16] shows quadrotor dynamics with the four inputs is differentially flat. Therefore, it formulates trajectory as a polynomial function and optimizes it by utilising quadratic programming (QP). There is also an algorithm that generates more aggressive trajectories for quadrotor was proposed on the basis of time allocation algorithm for way-points, algorithm of ensuring collision-free trajectories, algorithm of state estimation for quadrotor, furthermore extends to trajectory generation for fixed-wing aircraft with Dubins-type dynamics [17]. Operator Splitting Quadratic Program (OSQP) [18] as the state-of-art QP solver enables polynomial trajectory to be optimized and generated in real-time [19].

An improved bidirectional RRT algorithm is utilized for path re-planning in dynamic environment in [20]. This algorithm is able to update paths rapidly with a tree pruning technique that only keeps a bit of nodes in the configuration space. The pruning technique leads to insufficient exploration of configuration space, and it can be seen from the simulation results that the quality of generated paths is poor. Algorithms for on-the-fly real-time planning using simplified tree data have also been proposed previously in [21]. The most important problem of this method is sacrificing the exploration information of configuration space to obtain real-time re-planning. It should be necessary to explore

urban airspace intensively in order to connect and improve flight path with better solutions while considering the flight safety since urban airspace is a complex and dynamic environment with uncertainties. A hybrid approach of virtual force and A* search algorithm [22] is proposed for re-planning and it has promising results for simple problems. However, this approach cannot avoid the explicit construction of configuration space, which can be prohibitive in complex planning problems, especially for operations in high dimensional space. Inverse reinforcement learning [23] (IRL), a classical imitation learning algorithm, is utilized for aerial vehicle path re-planning in dynamic environment [24] with the aid of the RRT*. Unfortunately, IRL approach has many flaws such as complex implementation and requirement of the reinforcement learning in an inner loop which makes it extremely expensive to run. When compared with the IRL, GAIL is simpler to implement and consumes fewer computing resources. More importantly, IRL learns a cost function, whereas GAIL learns a policy network, which means that IRL learns behavior, while GAIL learns policy to generate behavior. Such a difference enables GAIL to particularly outperform IRL in the face of complex planning problems.

III. An Reinforcement Learning based approach for generating minimal snap trajectory

In this section, we present decoupled re-planning algorithm using reinforcement learning and online minimum snap trajectory generation algorithm, which can re-plan the trajectory in significantly short computation time while ensuring sufficient exploration of the environment. The first obvious problem with reinforcement learning is that we cannot use the entire tree data as input to the policy network. Tree data contains massive nodes, and each node has several attributes. The entire tree data can be regarded as an extremely high-dimensional vector, but it is impossible for reinforcement learning to deal with such problems. Our solution is to use Recurrent Neural Networks as policy network of RL, which is also the solutions of several other problems. Fig. 1 shows the policy network.

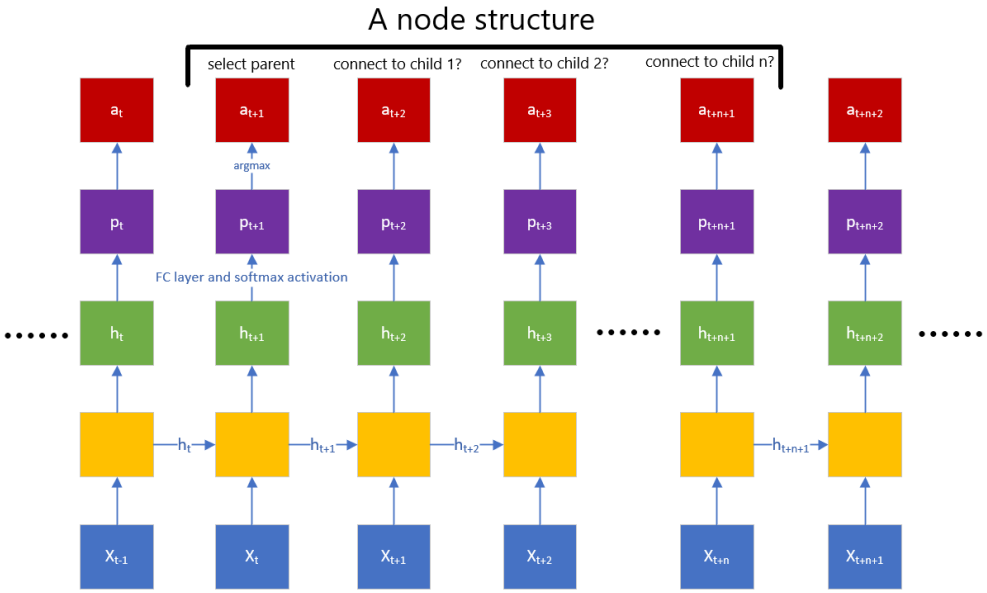


Fig. 1 Recurrent Neural Networks policy network of RL.

Proximal Policy Optimization (PPO) [25] is utilized in the training phase of the reinforcement learning framework. This algorithm originates from Trust Region Policy Optimization (TRPO)[26] algorithm. TRPO applies second-order trust region method to maximize the objective function, while PPO uses first-order method to simplify the implementation and perform at least as well as TRPO. Furthermore, for the approach purposed in this section, PPO has other three main advantages over other policy gradient learning algorithms. Firstly, it is less sensitive to learning rate, much easier to tune hyper-parameters and performs more stable learning process. Secondly, hyper-parameters of the algorithm have robust characteristics against variety of tasks[27]. Thirdly, PPO can achieve the same performance as other policy gradient method with less state, action and reward collected.

Asymptotic Optimality of RRT* stems from *ChooseParent* and *Rewire* operations. In the *ChooseParent* operation, it traverses the set of potential parent nodes that are inside a specific Euclidean distance of a child node, detecting collision,

updating the tree with most qualified parent node. Then, in the *Rewire* operation, it iterates through nodes within neighbourhood area around new node. The new node becomes a parent of some nodes if no collision and cumulative cost reduction happens.

The RNN policy network contains several dozens of node structures, sum of numbers of node structures in several policy network is equivalent to the number of nodes in the tree data. We normally put 20 node structures in a policy network. In training, agent learns to add connections for the environment that already exists some connections, such design avoids the problem of too long sequences. Each node structure has $n_c + 1$ tokens and each token outputs an action, where n_c donates the number of near nodes. This means that the n_c nodes closest to each node will be considered as its potential parents and potential children. The output of the first token of each node structure is an n_c-dimensional one-hot vector representing the selection of the parent node. The output of the second to $n_c + 1^{th}$ token is a 2-dimensional one-hot vector representing whether to select the corresponding node as a child node. It can be found that the dimensions of the output of these two decisions are different, but they need to be trained together. Being able to handle such problems easily is also one of the benefits of using RNN as a policy network. Besides, each node can only observe the current state of n-nearest nodes. In this situation, decision-making is difficult as agent can only partially observe the environment. For reinforcement learning problems with incomplete observations, past observations should be memorized, and all known information should be used to make decisions. This problem is solved with RNN policy networks as well. The value of n_c is a hyper-parameter that affects speed of convergence of reinforcement learning algorithm and results of training. Since our algorithm has no step of sampling in configuration space, the condition of Asymptotic Optimality cannot guide the choice of the value of n_c. After user definition of n_c, iterating over tree data will be executed so as to find n_c nearest nodes of each node, tree data will be updated by adding n_c nearest nodes for each node before training start.

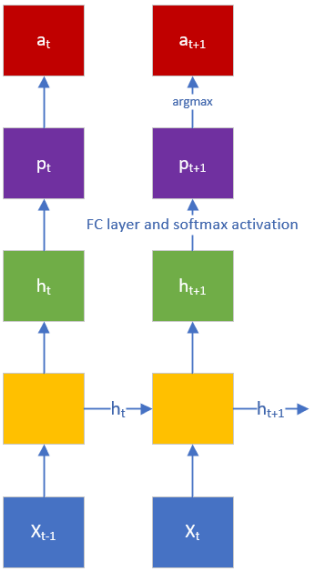


Fig. 2 Details of token of RNN policy network

Fig. 2 shows the details of token of RNN policy network. Part of the node information F_t undergoes data pre-processing and then X_t is obtained. The next step is inputting X_t and the feature vector h_t output by the hidden layer of the previous token into the hidden layer of the current token to obtain the feature vector h_{t+1}, so that the state s_t is $[F_t, h_t]$. Here, we only use one hidden layer in each token. After fully connected layer and softmax activation, a probability distribution p_{t+1} will be generated, and finally the argmax function will be applied on p_{t+1} to output an action a_{t+1}. Therefore, the policy network is denoted as $\pi(a_{t+1}|[F_t, h_t]; \theta)$.

Fig. 3 shows the details of data pre-processing. F_t is a three-dimension vector $[index, posx, posy]$ including index, x coordinate and y coordinate of the node. Obstacle information only needs to be input in the first token, and the RNN will convey it to subsequent tokens. 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

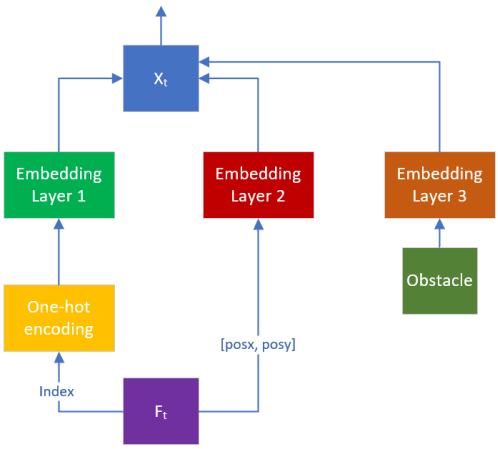


Fig. 3 Details of data pre-processing

data. This vector is too sparse so that 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, 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 the embedding layer 2 which amplify the location features of the nodes by increasing the dimension. Embedding layer 2 is a $l_d \times 2$ matrix that outputs a 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 are 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 next step.

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

Algorithm 1: Training of RNN policy network.

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 to learn policy of on-the-fly re-planning in 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.

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

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

- connecting each previously unconnected node
- each node that has less cost
- each connection that collides with obstacles
- euclidean distance from each node to the tree root
- a big punishment if parent is also connected as child

At the beginning of each episode, each node are orphan node and each node's cost to tree root is infinite. When the tree growing from the root is connected to an orphan node, the cost becomes a finite real numbers and a fixed reward is obtained. If subsequent updates of tree data reduce the cost from a node to tree root, a normalized reward will be obtained.

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

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

Afterwards, with Proximal Policy Optimization (PPO) algorithm[25], parameter of policy network will be updated once every $n+1$ steps via clipping surrogate objective showed 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 to control the clipping ratio. $\mathbb{E}[\cdot]$ denotes the expectation operator obtained by Monte Carlo approximation[28]. $A_{\theta old}[\cdot]$ is advantage function under 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 entropy operator displayed in equation (5). Entropy is used to measure the probability distribution output by softmax activation function in the policy network. Smaller entropy leads to more concentrated probability distribution. We found that entropy shrinks to a small value prematurely during training process, resulting in insufficient exploration in policy learning, therefore obtaining a defective policy. In face of this problem, we then introduce entropy regularization[29] to Equation (3).

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

λ donates adaptive entropy regularization coefficient showed in Equation (6).

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

where 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 has vanishing and exploding gradient problems[30] since its principle is based on simple iterative way. The standard RNN is therefore not appropriate for problem of long-term dependencies. As the policy network of the approach proposed in this section has long sequence, we introduce Long Short-Term Memory (LSTM)[31] and Gated Recurrent Unit (GRU)[32] in our actual simulation in order to overcome the drawbacks of the standard RNN. LSTM uses a conveyor belt to get longer memory than standard RNN and avoids the issues of exploding and vanishing gradients existing in RNN while the algorithm structure is far more complex. Each token of LSTM has four blocks named Forget Gate, Input Gate, New Values and Output Gate respectively. Each block also has a parameter matrix and these matrices have the same size, shape of $h_t \times$ shape of $(h_t + X_t)$. GRU has a simpler architecture, compared with LSTM, but has similar performance. There are three blocks with parameter matrix, which are Reset gate, Update gate and Candidate hidden state. Fewer parameters result in faster computation of GRU than that of LSTM as less memory is required.

The next step in this method is to generate a safe trajectory from the obtained waypoints and predefined total time. In our case, multi-copter with four rotor systems is regarded as the vehicle in simulation, so that we can follow the minimum snap formulation for quadrotor trajectory generation[16]. It shows piecewise polynomial trajectories can be utilized to specify flat outputs of coordinates of various dimensions and yaw angle x, y, z, ψ and their derivatives for smooth trajectory since vehicle similar to quadrotor possesses differential flatness characteristic. The cruise phase of the UAM vehicle will be limited to an altitude range in the urban airspace, as well as there will be insignificant flight altitude changes during this stage[1], so that we will ignore the planning of z coordinate. Besides, the planning for yaw angle will be left as future work.

Given the start-time of trajectory and all end-times of segments (τ_0, τ_1, τ_2,$\cdots$,τ_M), one dimension out of x, y of the M-segment, N^{th} order piecewise polynomial trajectory[33] 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 j^{th} order polynomial coefficient of the i^{th} segment. As Minimum snap trajectory is required, we phrase the problem as the generic minimization of the 4^{th} derivative of $f(\tau)$. In that case, the trajectory optimization problem for flat polynomial output is described as equation (8)

$$\min \int_{\tau_0}^{\tau_M} \left(\frac{d^4 f(\tau)}{d\tau^4} \right)^2 d\tau \tag{8}$$

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[16][17], $\mathbf{Q}_i$ 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 $4 \le q_r \le N, q_r \in \mathbb{N}$ is the row index starting from number 0 as well as $4 \le q_c \le N, q_c \in \mathbb{N}$ is the column index starting from number 0.

The trajectory has predefined start point, way-points and end point, as well as 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 ($\mathbf{A}_{lq}\mathbf{p} \le \mathbf{b}_{lq}$) constraints. As a result, we form a quadratic programming problem (QP) for trajectory generation, shown in Equation (11).

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

In our scenario, the total time of the trajectory is fixed, we need to assign a time amount $(\tau_i - \tau_{i-1})$ to each segment, and these time allocation will factor into the construction of the cost matrix. It is necessary to apply a specific algorithm to optimize the time allocation, as it directly affects the quality of final trajectory generated. The classic method is to

begin with an initial guess of segment times and then iteratively refine those times using gradient descent [16] [17]. This kind of method is relatively time-consuming as well as has a big challenge to achieve real-time. Additionally, some trajectory segments that are overly convex in shape may intersect obstacles, even though the path on which the way-points follow is collision-free. To address these two issues, inspired by the work of Chen *et al.*[33], we introduce corridor as constraint into the QP problem.

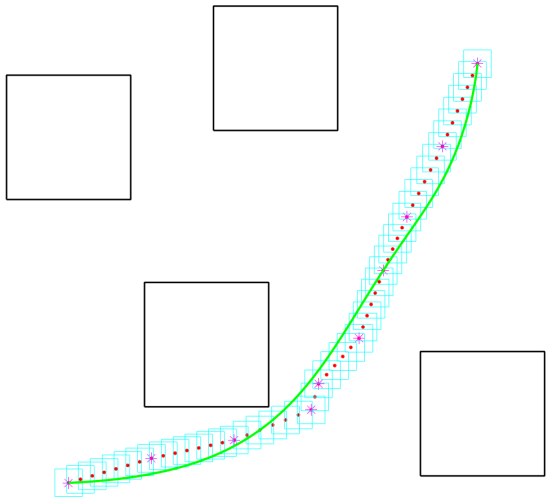


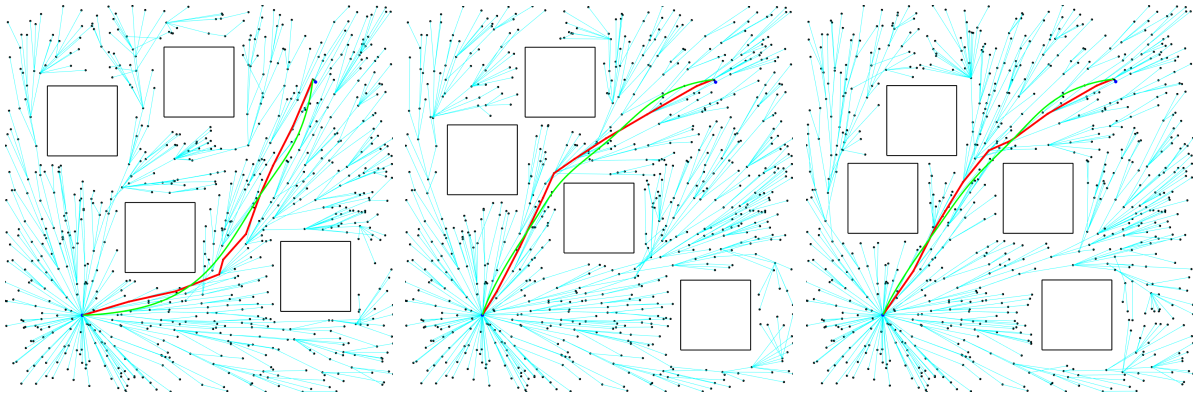
Fig. 4 Trajectory generation with corridor for collision avoidance and implicit time optimization

As shown in Figure 4, we uniformly add some sub-waypoints in the path, then define a inequality constraint about their position. Meanwhile, we change the strong constraint about position of waypoints into weak constraint, which allows the trajectory to pass around the waypoints instead of having to pass through the way-points. With the weak constraint defined in Equation (12), the problem that vehicle has to pass a certain position at predefined time is transformed to arriving inside a square area at the predefined time. Such an algorithm makes implicit time optimization through adjustment of segment transition points. Although this method actually increases the number of segments, it can generate a trajectory in approximately 0.1 seconds with the state-of-art QP solver OSQP [18], since iterations are not required in computing. Moreover, this approach simultaneously confine the trajectory to a corridor formed by several square areas.

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

where p_ϕ and τ_ϕ are positions and allocated time for particular waypoints or sub waypoints, 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 path planning algorithm and the flight interval stipulated by the local traffic regulations.

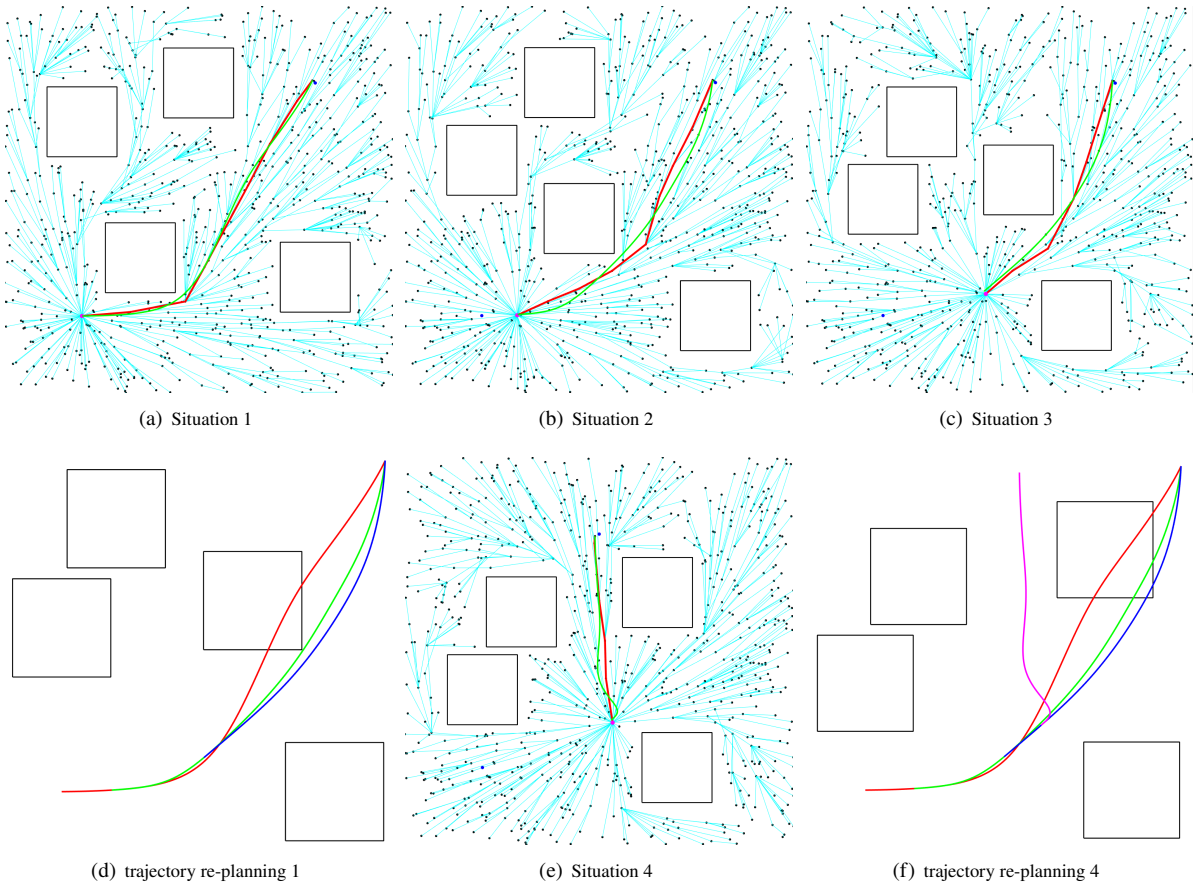
Figure 5 displays solving real-time motion re-planning problem with fixed start point in dynamics environment utilizing the decoupled approach proposed in this section. It can be found that, as long as using the tree data with sufficient exploration, even without sampling of the configuration space, our approach can plan or re-plan path with high quality in dynamic environment via updating and optimizing the connections between existing tree nodes. Then a collision-free trajectory that has optimized snap is simultaneously generated in accordance with path. Figure 6 shows the results that vehicle is on the fly. The tree node closet to the position where on-the-fly vehicle needs to execute re-planning will be defined as new tree root (as known as start point of path planning). The path re-planning method proposed in this section has a significant merit that enables user to directly change start point without extra processing such as root moving algorithm[21]. After way-points obtained, we determine the initial state of trajectory as the current position, velocity and acceleration of aerial vehicle rather than start point of path, then generate trajectory. Figure 6(a), 6(b), and 6(c) illustrate the simulation results that our vehicle performs real-time motion re-planning by the approach proposed in this section during flight when the original trajectory became infeasible due to uncertain and dynamic



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

Fig. 5 Real-time motion re-planning in dynamics environment

obstacles. The red straight line segments are the path generated by RNN policy network and the green curves are minimal snap trajectories. Figure 6(e) shows that if the final landing point needs to be changed for some reason such as emergency situations, this method can still rapidly re-plan a safe and optimized trajectory. Figure 6(d) and 6(f) demonstrate that our approach guarantees the continuity of trajectory undergoing re-planning.



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

(d) trajectory re-planning 1 (e) Situation 4 (f) trajectory re-planning 4

Fig. 6 Real-time on-the-fly re-planning in dynamics environment

Table 1 shows results of simulation executing times on computer with AMD Ryzen 9 5900x CPU, NVIDIA RTX

3090 GPU and 32GB RAM. The benchmark used is the combination of Informed RRT*[34] and Richter *et al.*'s method using technique of substitution to convert trajectory generation problem into an unconstrained QP in order to avoid the issues of ill-conditioning and increase the efficiency when there are many way-points and segments in a trajectory[17]. In various configuration of simulation, our algorithm on the basis of a set of tree data containing 800 nodes uses LSTM or GRU as policy network as well as has 20, 16, 12 or 8 potential parents and children. NVIDIA CUDA Deep Neural Network library (cuDNN) is applied in order to GPU-accelerate the execution of LSTM and GRU. We randomly sampled 20 environments similar to Figure 6, applying each algorithm separately on each environment, obtaining snaps and mean computation time of each algorithm and finally calculating mean normalized snap after normalizing snaps with that of benchmark algorithm. It can be found that our proposed method has significantly short computation time when compared to the benchmark algorithm, while the quality of the obtained trajectories is not much lower than that generated by the benchmark algorithm.

Table 1 Simulation results of decoupled motion planning approach

Algorithm	**Mean Compute Time (s)**	**Mean Normalized Snap**
Info. RRT* + poly	16.19	1.00
Our method with LSTM, n_c=20	5.07	1.09
Our method with LSTM, n_c=16	4.13	1.10
Our method with LSTM, n_c=12	2.99	1.17
Our method with LSTM, n_c=8	1.82	1.29
Our method with GRU, n_c=20	3.41	1.07
Our method with GRU, n_c=16	2.65	1.09
Our method with GRU, n_c=12	1.93	1.17
Our method with GRU, n_c=8	1.27	1.32

IV. Approach based on Generative adversarial imitation learning

In this section, we propose a coupled planning algorithm that generates the time-optimal trajectory in extremely short computation time via 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*[14]. Unlike RRT*, the step size of Kinodynamic RRT* is not Euclidean distance but time difference, as well as 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. In addition, 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 parent, the time cost to reach the node from starting point, as well as the coordinate, velocity, and acceleration of each dimension. If we still use 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. If we still use 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 get 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 13 shows the decision-making record τ^{real} created by Kinodynamic RRT*. Where s_t^{real} are consistent every five time steps in two-dimensional motion planning and donates node index and positions, a_t^{real} represents the parent selection, single-dimensional velocity and acceleration of the corresponding s_t^{real}.

$$\tau^{real} = [s_1^{real}, a_1^{real}, s_2^{real}, a_2^{real}, \cdots, s_t^{real}, a_t^{real}, \cdots, s_m^{real}, a_m^{real}] \tag{13}$$

The GAIL consists of a generator and a discriminator. The generator is a policy network that makes decisions and we will use PPO[25] 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 discriminator more accurate to determine 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 III, we cannot use the entire tree data as input to the policy network, since tree data contains 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 recurrent neural network as the policy network (as known as generator) of GAIL in terms of handling this problem, figure 7 shows the details of the RNN policy network.

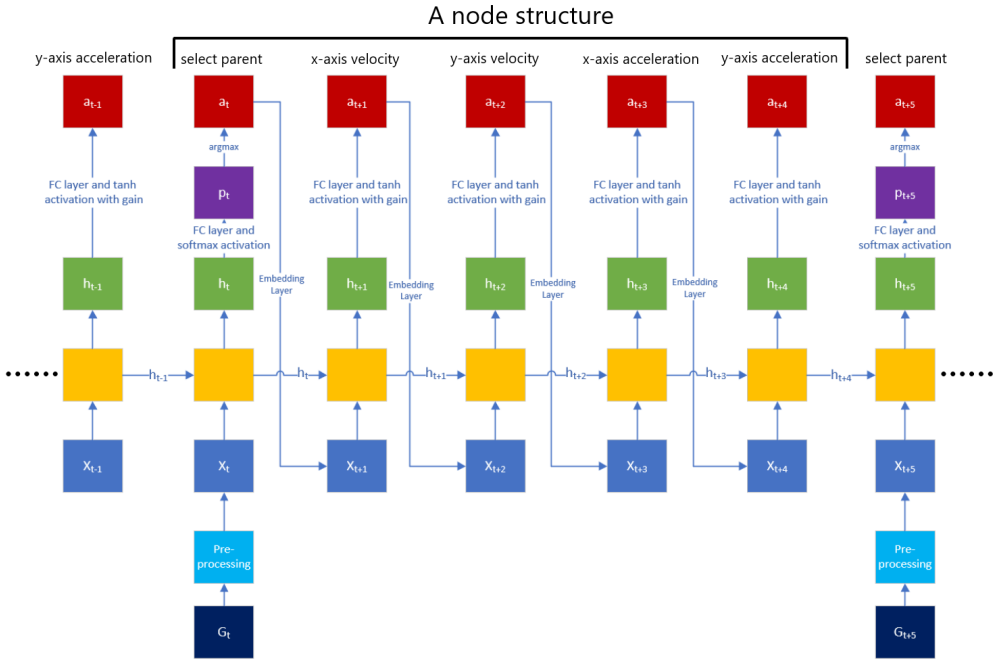


Fig. 7 Using Recurrent Neural Networks as policy network of GAIL.

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 sequence. In two dimensional motion planning, each node structure has 5 tokens and each token outputs an action. The design of the first token in each node structure is very similar to that in section III, the input X_t is pre-processed G_t that contains 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 speed of convergence of reinforcement learning algorithm and results of training. Before training start, 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 zero initial velocity and acceleration. The output of the second token of each node is a single-dimensional velocity whose value depends on the position of node itself and the choice of the parent node, as well as is influenced by other nodes. The hidden layer of RNN contains information of index and position of 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 (as known as the selection of 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 node itself, the choice of the parent node, and the obtained velocity of another dimension, as well as 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 which 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 that 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. 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. Similar to approach proposed in Section III, we do not use standard RNN in fact, but use GRU to avoid the problem of gradient vanishing and gradient explosion, each token has two hidden layers. Equation 14 displays the decision-making record τ^{fake} created by the RNN policy network. Where s_t^{fake} are also consistent every five time steps in two-dimensional motion planning and donates 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.

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

The discriminator of GAIL is a neural network whose structure is shown in Figure 8. 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 generated by the policy network.

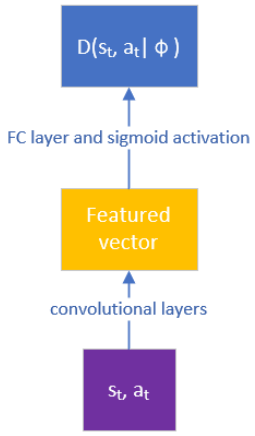


Fig. 8 Discriminator of GAIL

The goal of training GAIL is to make the generator (as 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, the generator that can fool it will get 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 15 defines the loss function.

$$F(\tau^{real}, \tau^{fake}|\phi) = \frac{1}{m}\sum_{t=1}^{m} \ln\left[1 - D(s_t^{real}, a_t^{real}|\phi)\right] + \frac{1}{m}\sum_{t=1}^{m} \ln\left[D(s_t^{fake}, a_t^{fake}|\phi)\right. \tag{15}$$

We expect the loss function to be as small as possible, so that we can use gradient descent to update parameters ϕ, which is shown in function 16.

$$\phi \leftarrow \phi - \beta\nabla_\phi F(\tau^{real}, \tau^{fake}|\phi) \tag{16}$$

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 to those generated by Kinodynamic RRT*, and the more successful the imitation learning, therefore, we substitute the reward u_t with Equation 17.

$$u_t = \ln D(s_t^{fake}, a_t^{fake}|\phi) \tag{17}$$

Then according to u_t, we can apply the PPO algorithm to train the RNN policy network π_θ of GAIL with equation 18 and 19.

$$L^{clip}(\theta) = \mathbb{E}(min(r(\theta)A_{\theta old}([X_t, h_{t-1}], a_t)), clip(r(\theta), 1-\epsilon, 1+\epsilon)A_{\theta old}([X_t, h_{t-1}], a_t) \tag{18}$$

$$r_t(\theta) = \frac{\pi_\theta((a_t|[X_t, h_{t-1}]))}{\pi_{\theta old}((a_t|[X_t, h_{t-1}]))} \tag{19}$$

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

The training performance of the RNN policy network acting as the generator of GAIL is strongly dependent on the hyper-parameters of PPO algorithm. The discriminator that outputs rewards for 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[35] as this optimization problem has no explicit objective function expression.

The core in the Bayesian optimization method includes a Surrogate Model and a Acquisition Function. In our approach, Gaussian process model (GP) is applied as 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 time cost of trajectories respectively generated by RNN policy network and the benchmark algorithm kinodynamic RRT* in Monte Carlo simulation. In detail, We randomly generated 1000 sets of different starting points and target points, as well as obstacle positions, for 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 time cost of the two trajectories, finally obtain the mean ratio of 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 20.

$$x_{n+1} = \arg\max_x g(x|X) \tag{20}$$

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 21 [36]

$$EI(x) = \begin{cases} (\mu(x) - f(x^+ - \xi))\Phi(Z) + \sigma(x)\phi(Z), & \sigma(x) > 0 \\ 0, & \sigma(x) = 0 \end{cases} \tag{21}$$

where

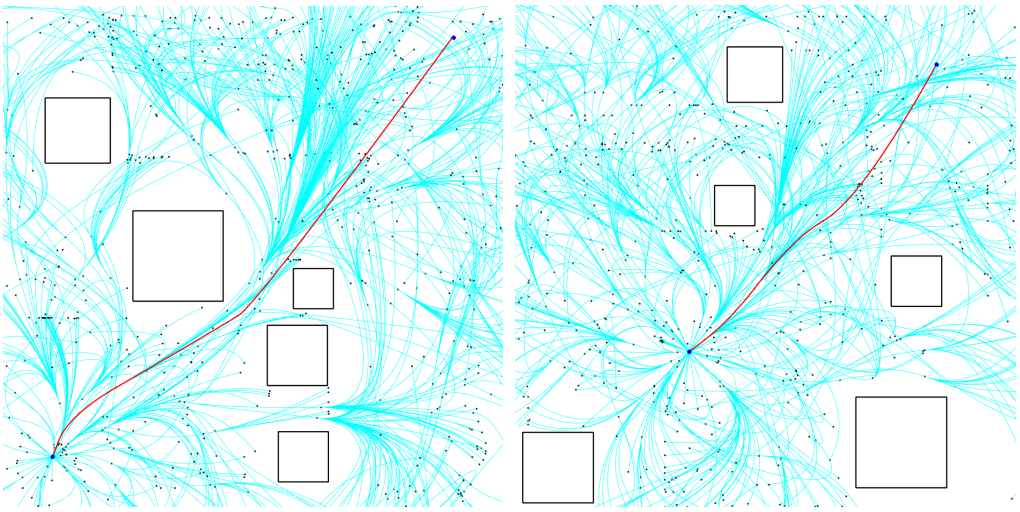
$$Z = \begin{cases} \frac{\mu(x) - f(x^+ - \xi)}{\sigma(x)}, & \sigma(x) > 0 \\ 0, & \sigma(x) = 0 \end{cases} \tag{22}$$

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 21 is used for exploitation, the second term $\sigma(x)\phi(Z)$ is used for exploration, the parameter ξ determines the proportion of exploration.

Additionally, when the vehicle that needs to do 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 do fine-tuning on the basis of the RNN policy network trained previously. Fine-tuning will perform the following five steps;

- changing values of gain of gained tanh activation function
- setting independent learning rates for the two hidden layers of the RNN policy network
- fine-tuning by GAIL with less episodes
- applying Bayesian hyper-parameter optimization method
- using Greedy Soup receipt of Model Soups method[37]

Model soups is a method 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 performance on a held out validation set evaluated by Monte Carlo simulation result improves. Before running this procedure we sort the models in decreasing order of validation set accuracy, and so the greedy soup can be no worse than the best individual model on the held-out validation set[37].



(a) Situation 1 (b) Situation 2

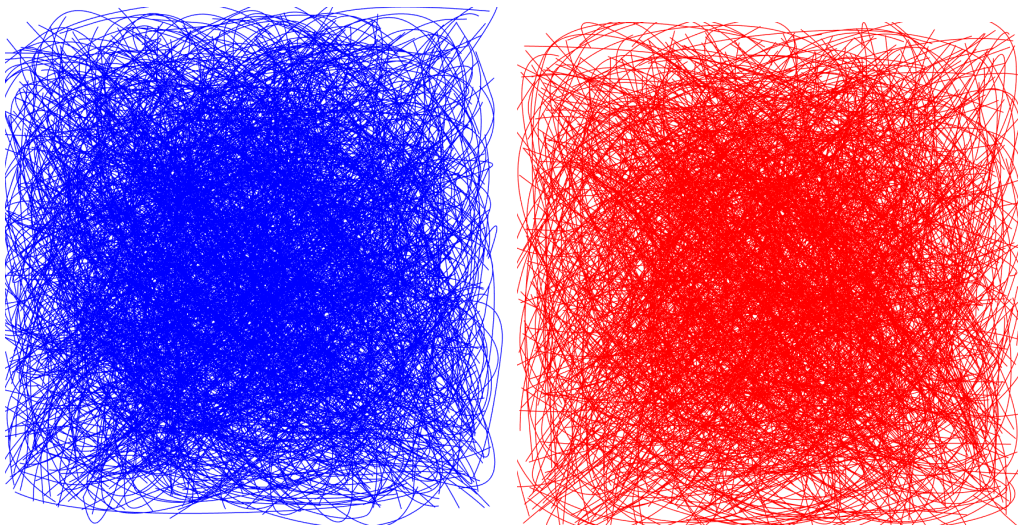
Fig. 9 GAIL based motion planning

Figure 9 shows the results produced by 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. Besides, GRU is utilized as the RNN policy network, as well as applying cuDNN acceleration. Based on a tree data of 1200 nodes, extremely short computation time is required to generate the trajectory using the trained policy network, only around three seconds on our device, while it takes more than two minutes to complete 1200 iterations using Kinodynamic RRT*.

Figure 10(a) shows the Monte Carlo simulation results produced by approach based on GAIL. After applying a Bayesian hyper-parameter optimization method, it can be found that the properties of the resulting trees, including velocity, acceleration, and the time that takes to reach the node, are close to the results of Kinodynamic RRT*. Figure 10(b) displays the Monte Carlo simulation results for fine-tuning based on original generated RNN policy network. In this scene, velocity limit is increased by half and acceleration limit is doubled. As shown in 2, with Bayesian hyper-parameter optimization and Greedy soup, the performance of fine-tuned policy network for new vehicle is close to that of original policy network.

Table 2 Monte Carlo simulation results of coupled motion planning approach

Algorithm	Mean Compute Time (s)	Mean Normalized Time to reach the goal position
Kinodynamic RRT*	121	1.00
Original policy network by our method	3.24	1.09
Fine-tuned policy network by our method	3.31	1.10



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

Fig. 10 Monte Carlo simulation for GAIL based motion planning and Fine-tuning

V. Conclusion and Future work

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 uncertainty of the environment, kinematic and dynamic constraints of the vehicle and possible emergency situations. Both of our methods utilise the tree data generated by RRT* based algorithm, which ensures ample exploration of the configuration space. Moreover, we apply neural network inference instead of 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 recurrent neural network for the reinforcement learning algorithm in terms of addressing three issues. These three issues are; a) input vector has significantly high dimension, b) input and output vectors of consecutive steps have different dimensions, and c) agent is unable to observe global environment. Afterwards, we combine an online trajectory generation algorithm to obtain the minimal snap trajectory for the vehicle. Simulation results demonstrate that this method can generate high-quality trajectories in extremely short computation time. Furthermore, it enables on-the-fly motion re-planning, the re-planned trajectory maintains continuity for executed trajectory.

We also propose a coupled method that generates 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, we use Bayesian hyperparameter optimization to solve the problem that the results of this method are 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 velocitys and accelerations, which avoids the need to retrain a policy network, as well as the performance of the fine-tuned model is close to the original model.

References

[1] Bauranov, A., and Rakas, J., "Designing airspace for urban air mobility: A review of concepts and approaches," *Progress in Aerospace Sciences*, Vol. 125, 2021, p. 100726.

[2] Yu, X., and Zhang, Y., "Sense and avoid technologies with applications to unmanned aircraft systems: Review and prospects," *Progress in Aerospace Sciences*, Vol. 74, 2015, pp. 152–166.

[3] Murray, C. W., Ireland, M., and Anderson, D., "On the response of an autonomous quadrotor operating in a turbulent urban environment," *AUVSI's unmanned systems conference*, 2014.

[4] Logan, M. J., Bird, E., Hernandez, L., and Menard, M., "Operational Considerations of Small UAS in Urban Canyons," *AIAA Scitech 2020 Forum*, 2020, p. 1483.

[5] Pang, B., Ng, E. M., and Low, K. H., "UAV trajectory estimation and deviation analysis for contingency management in urban environments," *AIAA Aviation 2020 Forum*, 2020, p. 2919.

[6] for Aeronautics, R. T. C., *Minimum Aviation System Performance Standards for Automatic Dependent Surveillance Broadcast (ADS-S).*, RTCA, Incorporated, 2002.

[7] Dill, E. T., Young, S. D., and Hayhurst, K. J., "SAFEGUARD: An assured safety net technology for UAS," *2016 IEEE/AIAA 35th digital avionics systems conference (DASC)*, IEEE, 2016, pp. 1–10.

[8] LaValle, S. M., *Planning algorithms*, Cambridge university press, 2006.

[9] Medsker, L. R., and Jain, L., "Recurrent neural networks," *Design and Applications*, Vol. 5, 2001, pp. 64–67.

[10] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An introduction*, MIT press, 2018.

[11] Ho, J., and Ermon, S., "Generative adversarial imitation learning," *Advances in neural information processing systems*, Vol. 29, 2016.

[12] Karaman, S., and Frazzoli, E., "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, Vol. 30, No. 7, 2011, pp. 846–894.

[13] LaValle, S. M., et al., "Rapidly-exploring random trees: A new tool for path planning," 1998.

[14] Webb, D. J., and Van Den Berg, J., "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," *2013 IEEE international conference on robotics and automation*, IEEE, 2013, pp. 5054–5061.

[15] Chiang, H.-T. L., Hsu, J., Fiser, M., Tapia, L., and Faust, A., "RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies," *IEEE Robotics and Automation Letters*, Vol. 4, No. 4, 2019, pp. 4298–4305.

[16] Mellinger, D., and Kumar, V., "Minimum snap trajectory generation and control for quadrotors," *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 2520–2525.

[17] Bry, A., Richter, C., Bachrach, A., and Roy, N., "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," *The International Journal of Robotics Research*, Vol. 34, No. 7, 2015, pp. 969–1002.

[18] Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S., "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, Vol. 12, No. 4, 2020, pp. 637–672.

[19] Burke, D., Chapman, A., and Shames, I., "Generating minimum-snap quadrotor trajectories really fast," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 1487–1492.

[20] Meng, L., Qing, S., and Jun, Z. Q., "UAV path re-planning based on improved bidirectional RRT algorithm in dynamic environment," *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, IEEE, 2017, pp. 658–661.

[21] Lou, J., Yuksek, B., Inalhan, G., and Tsourdos, A., "An RRT* Based Method for Dynamic Mission Balancing for Urban Air Mobility Under Uncertain Operational Conditions," *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, IEEE, 2021, pp. 1–10.

[22] Dong, Z., Chen, Z., Zhou, R., and Zhang, R., "A hybrid approach of virtual force and A* search algorithm for UAV path re-planning," *2011 6th IEEE Conference on Industrial Electronics and Applications*, IEEE, 2011, pp. 1140–1145.

[23] Ng, A. Y., Russell, S. J., et al., "Algorithms for inverse reinforcement learning." *Icml*, Vol. 1, 2000, p. 2.

[24] Sadhu, A. K., Shukla, S., Sortee, S., Ludhiyani, M., and Dasgupta, R., "Simultaneous Learning and Planning using Rapidly Exploring Random Tree* and Reinforcement Learning," *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2021, pp. 71–80.

[25] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[26] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P., "Trust region policy optimization," *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[27] Yuksek, B., Umut Demirezen, M., Inalhan, G., and Tsourdos, A., "Cooperative Planning for an Unmanned Combat Aerial Vehicle Fleet Using Reinforcement Learning," *Journal of Aerospace Information Systems*, Vol. 18, No. 10, 2021, pp. 739–750.

[28] Hammersley, J., *Monte carlo methods*, Springer Science & Business Media, 2013.

[29] Chow, Y., Nachum, O., and Ghavamzadeh, M., "Path consistency learning in tsallis entropy regularized mdps," *International conference on machine learning*, PMLR, 2018, pp. 979–988.

[30] Pascanu, R., Mikolov, T., and Bengio, Y., "On the difficulty of training recurrent neural networks," *International conference on machine learning*, PMLR, 2013, pp. 1310–1318.

[31] Hochreiter, S., and Schmidhuber, J., "Long short-term memory," *Neural computation*, Vol. 9, No. 8, 1997, pp. 1735–1780.

[32] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[33] Chen, J., Su, K., and Shen, S., "Real-time safe trajectory generation for quadrotor flight in cluttered environments," *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, IEEE, 2015, pp. 1678–1685.

[34] Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D., "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 2997–3004.

[35] Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., and Deng, S.-H., "Hyperparameter optimization for machine learning models based on Bayesian optimization," *Journal of Electronic Science and Technology*, Vol. 17, No. 1, 2019, pp. 26–40.

[36] Zhang, D., Ma, G., Deng, Z., Wang, Q., Zhang, G., and Zhou, W., "A self-adaptive gradient-based particle swarm optimization algorithm with dynamic population topology," *Applied Soft Computing*, Vol. 130, 2022, p. 109660.

[37] 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," *International Conference on Machine Learning*, PMLR, 2022, pp. 23965–23998.