Autonomous On-Demand Free Flight Operations in Urban Air Mobility using Monte Carlo Tree Search

Xuxi Yang
Aerospace Engineering Department
Iowa State University
Ames, IA, USA
xuxiyang@iastate.edu

Peng Wei
Aerospace Engineering Department
Iowa State University
Ames, IA, USA
pwei@iastate.edu

***Abstract*—Vertical takeoff and landing (VTOL) aircraft for personal air transportation or on-demand air taxi will bring fundamental changes to city infrastructures and daily commutes. NASA, Uber, and Airbus have been exploring the exciting concept of Urban Air Mobility (UAM). In order to enable safe and efficient autonomous on-demand free flight operations in this UAM concept, a computational guidance algorithm was designed and analyzed with collision avoidance capability. The approach is to formulate this problem as a Markov Decision Process and solve it using an online algorithm called Monte Carlo Tree Search. For the sake of illustration, a simplified numerical experiment was created to test the performance of this algorithm. Results show that this algorithm can help aircraft quickly reach the trip destination and avoid conflicts with other aircraft.**

Keywords: Urban Air Mobility, Autonomous Free Flight, On-Demand Air Transportation, Collision Avoidance, Monte Carlo Tree Search

I. INTRODUCTION

A. Motivation

NASA, Uber and Airbus have been exploring the exciting new concept of Urban Air Mobility (UAM) [1-5], where the vertical takeoff and landing (VTOL) aircraft may be either human piloted or autonomous for passenger transport in personal commute or on-demand air taxi. The UAM operations are expected to fundamentally change cities and people's lives to reduce commute time and stress. The vehicle technology and airspace operation concepts are critical for UAM realization. Through close collaborations and discussions with colleagues from NASA, Uber and Airbus, in this paper the authors investigate how to combine the power of onboard aircraft intelligence (vehicle technology) and the advantage of the free flight idea (airspace operation concept) to enable safe and efficient flight operations in on-demand urban air transportation.

The idea of this paper is inspired from the analogy where a lot of people walking in a crowded plaza (Times Square in New York or Tiananmen Square in Beijing) and they do not collide into each other. In a crowded plaza, a person can avoid collision with other people by observing and judging the position, velocity, direction, and even intention of other people. So for aircraft equipped with autonomous avionics system, it should be similar: one aircraft should be able to avoid other intruder aircraft by sensing their positions and velocities while flying toward its destination.

The concept of "Free Flight" was proposed primarily for future air transportation applications because it has the potential to cope with the ongoing congestion of the current ATC system. It was shown in previous work that Free Flight with Airborne Separation is able to handle a higher traffic density with the same or higher level of safety [6]. Besides, Free Flight can also bring fuel and time efficiency [7,8]. In a Free Flight framework, it is implied that aircraft will be responsible for their own separation assurance. The loss of an airway structure, however, may make the process of detecting and resolving conflicts between aircraft more complex. Accordingly, automated conflict detection and resolution tools will be required to aid pilots and/or ground controllers in ensuring traffic separation.

In this paper, a computational guidance algorithm with collision avoidance capability is proposed using Monte Carlo Tree Search, where the input of this algorithm is the position, velocity of other aircraft, and the position of the destination. Through onboard sensed information of other aircraft, the aircraft will perform online sequential decision making to select actions in real time with onboard avionics. The series of actions will generate a trajectory which can guide the aircraft to quickly reach its goal and avoid potential conflicts. The proposed algorithm provides a potential solution framework to enable autonomous on-demand free flight operations in urban air mobility.

B. Related Work

There have been many important contributions to the topic of collision avoidance algorithms for aircraft. Kuchar and Yang had a comprehensive overview of 33 different methods for conflict avoidance problem, including force field techniques, genetic algorithms, rule based methods, and optimization techniques [9]. Some later methods such as

mixed-integer programming performs well for small networks but difficult to adapt to stochastic dynamic models [10-12]. Formulating the collision avoidance problem as a Markov Decision Process (MDP) has been shown to improve safety and operational efficiency [13,14]. The MDP-based method can also be adapted to handle the cooperation among aircraft [15]. However, the MDP-based algorithm needs to discretize the state space and build the transition model, which can lead to loss of information and inaccuracy of the solution. In addition, whether the MDP model can accurately reflect the dynamics of the real world needs more investigation. In this paper, an online Monte Carlo Tree Search algorithm was used to solve the computational guidance and collision avoidance problem for autonomous on-demand free flight operations in urban air mobility, where there is no need to discretize the state space. And the results show that this algorithm has very promising performance.

The structure of the paper is as follows: in Section II, the background of MDP and MCTS will be introduced. In Section III, the description of the problem and its mathematical formulation of MDP are presented. Section IV presents the designed MCTS algorithm to solve this problem. The numerical experiment and results are shown in Section V. And Section VI is the conclusion.

II. BACKGROUND

A. Markov Decision Process (MDP)

The Markov Decision Process (MDP) includes the following components:

- The state space $\mathcal{S}$ which consists of all the possible states.
- The action space $\mathcal{A}$ which consists of all the actions that the agent can take.
- Transition function $\mathcal{T}(s_{t+1} \mid s_t, a_t)$ which describes the probability of arriving at state s_{t+1}, given the current state s_t and action a_t.
- The reward function $\mathcal{R}(s_t, a_t, s_{t+1})$ which decides how much reward the agent will collect after a transition s_t, a_t, s_{t+1}. The reward function may only depend on the current state s_t, which will be the case in this paper.
- A discount factor $\gamma \in [0,1]$ which decides the preference on immediate reward versus future rewards. Setting the discount factor less than 1 is also beneficial for the convergence of cumulative reward.

In a MDP problem, a policy π is a mapping from the state to one specific action (known as deterministic policy)

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

The goal of MDP is to find an optimal policy π*, which can maximize the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} R\ s_t, a_t\ \mid \pi\right]$$

B. Monte Carlo Tree Search (MCTS)

MCTS is a very popular online algorithm designed to solve sequential decision making problems because it is a "statistical anytime" algorithm for which more computing power generally leads to better performance, and it can be used with little or no domain knowledge to obtain a successful result on difficult problems [16]. The basic idea is to judge the value of an action by taking random samples in the decision space and building a search tree according to the results. The agent doesn't need to be trained to get good performance. However, the agent usually needs high computation resources to achieve a decent result.

The MCTS algorithm involves running many simulations from the current state while estimating the state-action value (Q value) of the states in the tree. And the state-action value $Q(s,a)$ is the expected sum of discounted reward resulting from choosing action a from state s. The following is an overview of the iteration steps of a typical MCTS algorithm called Upper Confidence Bound for Trees (UCT):

- Selection: when the process is in the tree, select the node j to maximize

  $$UCT = \overline{X}_j + 2C\sqrt{\frac{2\ln n}{n_j}}$$

  where $\overline{X}_j$ is the normalized state-action value at node j, and n is the number of times the current node has been visited, n_j is the number of times child j has been visited, C is a positive constant. The first term represents the exploitation based on the current knowledge, and the second term represents the exploration to gather more information for some actions.
- Expansion: one (or more) child nodes are added to expand the tree, according to the available actions.
- Simulation: a simulation is run from the new nodes according to the default policy to produce an outcome.
- Backpropagation: the simulation result is 'backed up' (i.e. backpropagated) through the selected nodes to update their statistics.

III. PROBLEM FORMULATION

A. Problem Statement

The goal of this paper is to control an aircraft through a series of actions so that the aircraft can quickly arrive at the destination while avoiding potential conflict with other intruder aircraft during the flight. Guiding the aircraft through a series of actions is a sequential decision making problem which can be formulated as an MDP problem. In this process, the action is decided directly from the state, which incorporates all the information (the position and velocity of intruder aircraft, the position of the destination or goal) for the agent to decide which action is optimal for the corresponding state.

In this paper, a simplified scenario is considered: all the intruders can only fly straight at a fixed velocity, and only one aircraft (the ownship) is equipped with MCTS algorithm and will try to reach its goal. The collisions between intruders are not considered in this paper.

When controlling the aircraft, only horizontal actions are considered in this paper, which means all the aircraft will be flying at the same height and this problem can be solved in 2 dimension. This assumption is reasonable because UTM limits its focus to a narrow altitude band with one proposal restricting altitudes to between 200 and 500 feet [17].

The objectives for this specific MDP problem are two-fold: the first is to guide the aircraft to the goal state in a short time, and the second is to avoid any conflicts between the controlled aircraft and other intruder aircraft. Therefore, the reward function should be able to capture both two objectives.

Based on the above description, this problem will be mathematically formulated as an MDP problem in the next subsection.

B. MDP Formulation

1) State Space:

A state includes all the information the ownship need for its decision making: the position and velocity of the ownship and all the intruder aircraft, together with the goal position. Two numbers (x, y) are needed for the position of an aircraft. And another two numbers (v_x, v_y) are needed to describe the velocity of an aircraft. With this simplified assumption, the trajectory of each intruder will be fully determined by the above four numbers. For ownship, besides its position (x, y) and velocity (v_x, v_y), the heading angle is also included in the state, which can make determining the next state much easier. So if there are n intruders, 1 ownship, and 1 goal, it will need $4\times n+5\times 1+2$ numbers to describe the current state. In conclusion, the state space is a subset of $\mathbb{R}^{4n+7}$, and each state is a $4n+7$ dimensional vector, with each element in this vector taking a value in a different range.

Figure 1. an example state of the MDP problem.

For example, in Fig 1, the yellow aircraft is ownship, the red aircraft is the intruder, and the green star is the goal position for the ownship. If the position and velocity of intruder is (i_x, i_y) and (i_{vx}, i_{vy}), the position and velocity of ownship is (o_x, o_y) and (o_{vx}, o_{vy}), and the heading angle of ownship is o_ϕ, the position of goal state is (g_x, g_y), then the state will simply be $(i_x, i_y, i_{vx}, i_{vy}, o_x, o_y, o_{vx}, o_{vy}, o_\phi, g_x, g_y)$. If there are more than one intruders, the information of other intruders will also be included in the state.

2) Action Space:

At each time step (1 sec), the ownship can choose to turn left for 2°, turn right for 2°, or go straight. Specifically, the action space is

$$\mathcal{A} = \{-2^\circ/s, 0^\circ/s, 2^\circ/s\}$$

And the ownship maintains the action during this time step.

It's natural to consider extending the set of actions (conflict resolution advisories) to include change in speed and combinations of resolutions (e.g., turn left and speed up simultaneously). However, using MCTS algorithm to calculate the optimal action will be more time consuming with the extended action space since the tree size will be much larger.

3) Dynamical Model:

After the ownship chooses an action, a simple kinematic model will be used to compute state transition for ownship:

$$\dot{x} = v\cos\phi$$
$$\dot{y} = v\sin\phi$$
$$\dot{\phi} = a$$

where v is the speed of the intruder, ϕ is the heading angle, and a is the action that he ownship is currently taking, which takes the value options of -2°/s, 0°/s, 2°/s.

Here, the relation between the bank angle and the turning angle can be determined through the following equation:

$$\dot{\phi} = \frac{g \tan \psi}{v}$$

where ψ is the bank angle of ownship, and v is the speed of ownship. If the action is chosen to be 2°/s, and the speed of ownship is $20m/s$, then the bank angle of the aircraft will be 4°.

In addition, it is assumed that there is no noise when executing the chosen action, which means this process is deterministic. The simplicity of this dynamical model helps us reduce the computation time of MCTS algorithm and obtain a converged solution.

4) Terminal State:

For safety, the conflict of two aircraft is defined when the distance of two aircraft is less than a minimum separation distance $r^{min} = 320m$. And the terminal state of this MDP includes three different types of states:

- The distance from ownship to any intruder is less than r^{min} (referred to as a conflict state in the following);
- The ownship flies out of the map (referred to as a boundary state in the following);
- The ownship reaches the goal position (referred to as a goal state in the following).

The terminal state can be determined by examining the state information. For example, suppose the current state is

$$(i_x, i_y, i_{vx}, i_{vy}, o_x, o_y, o_{vx}, o_{vy}, o_\phi, g_x, g_y)$$

Then this state will be a conflict state if

$$\sqrt{(i_x - o_x)^2 + (i_y - o_y)^2} < r^{min}$$

If this state is not a conflict state, it will be a boundary state if the current position of ownship is out of the map. Finally, the goal state can be determined similar to the conflict state by calculating the distance from the ownship to the goal position.

5) Reward Function:

The goal in this paper is to make an aircraft quickly reach its destination and avoid potential conflict. These two objectives can be captured in the reward function defined as follows:

$$R(s) = \begin{cases} 1, & \text{if } s \text{ is goal state} \\ 0, & \text{otherwise} \end{cases}$$

Reaching a conflict state or a boundary state before the goal state will terminate the whole process with a reward of 0, which the agent will try to avoid. Reaching a goal state will terminate this process with a reward of 1. Together with a discount factor less than 1, the agent will try to reach the goal state as soon as possible.

IV. Solution Method

A. UCT Algorithm

For the MDP formulated above, the most popular algorithm in the MCTS family, the Upper Confidence Bound for Trees (UCT), is used to solve this problem. UCT has some promising properties: it's very simple and efficient, and guaranteed to be within a constant factor of the best possible bound on the growth of regret. And it can balance exploration and exploitation very well.

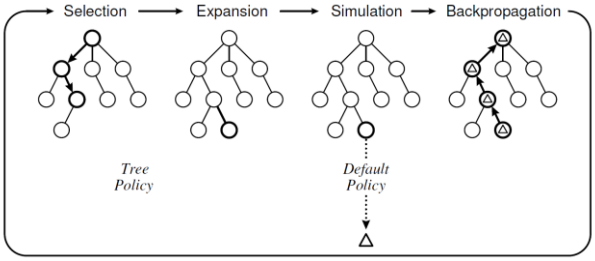

Figure 2. One iteration of general MCTS approach [16].

Before presenting the UCT algorithm for the above MDP formulated in Section III, several ideas and concepts need to be stated first:

In the MCTS algorithm, the nodes in the search tree will be the states in the state space of the MDP problem. In the remaining part of this paper, the state and the node will be used interchangeably. The child nodes of a node are all the possible next states (nodes) resulting from different actions from current state (node). Here, it should be noted that taking one action from the current state will only lead to one next state since this MDP problem is deterministic. So, each node will have at most 3 child nodes.

Informally, in Fig 2., an edge connecting the parent node and child node can be regarded as an action, which leads the parent node (current state) to the child node (next state).

After introducing the basic setting for the search tree of MCTS algorithm, it is now time to describe how to apply the UCT algorithm to solve the MDP problem.

The first step of the UCT algorithm is **selection**. Every time the ownship is in the tree, it selects a child node with maximum value:

$$UCT = \overline{X}_j + 2C\sqrt{\frac{2\ln n}{n_j}}$$

Here the first term $\overline{X}_j$ will be referred as exploitation term, which is directly from the formula

$$\overline{X}_j = Q(v) / N(v)$$

where the number $N(v)$ is the times this node has been visited before and the value $Q(v)$ is the total reward of all playouts that passed through this node (so that $Q(v)/N(v)$ is an approximation of the node's state-action value). The second term $2C\sqrt{2\ln n / n_j}$ is referred to as exploration, a higher C value will emphasize exploration and a lower C value will encourage exploitation. It should be noted that the value of C should depend on the value scale of $\overline{X}_j$. Since the value of $C = 1/\sqrt{2}$ was shown by Kocsis and Szepesvari [18] to satisfy the Hoeffding inequality with rewards in the range $[0,1]$, it is reasonable to set $C = 1/\sqrt{2}$ in this paper.

In the process of selection, if more than one child nodes have the same maximal value, the tie is broken by choosing the action $a = 0°$, as in general, it is usually not desirable for the ownship to take unnecessary action. This often happens when the ownship is very close to the goal position, where all three actions will have same mean reward 1. In this case, the ownship doesn't need to waste fuel by turning.

Meanwhile, it is generally understood that $n_j = 0$ yields a UCT value of ∞, so that if a node is never visited previously, it will be assigned to the largest possible value, to ensure that every child will be considered at least once before any expansion. This is the strategy used in this paper.

The second step for UCT algorithm is **expansion**, which happens when the ownship is at a new node which it has never visited before. This step is simply adding this new node to the current tree under its parent node (the previous state), and setting its visiting number to 1 and cumulative reward to 0.

While adding a new child node to the current node, the action which leads the current node to this new node should also be stored associated with this new node, since this action information will be used when selecting the best children.

Next step is **simulation**. After a new node is added to the tree, its value should be determined by running a simulation to a terminal state following a random policy. After a random action is selected, the next state should be determined to advance the simulation. Since the state space includes all the information (position and velocity of all the aircraft and goal position), this process can be regarded as Markovian, which means the next state can be determined directly from the current state and the chosen action. Specifically, the next state will be determined as follows:

For the intruder aircraft, its velocity remains unchanged since it's assumed that intruder aircraft can only fly straight at fixed velocity. The position of next state is the sum of current position and current velocity (assuming the time step is 1 sec).

For the goal position, it remains unchanged if the ownship doesn't arrive at the goal state. If the ownship arrives at the goal state, this state will be a terminal state with reward of 1.

For the ownship, the next state will be harder to determine since it also depends on the action chosen. At first the heading angle of ownship will change corresponding to the action chosen. Then new velocity (v_x', v_y') can be decided from the new direction:

$$v_x' = v\cos\phi'$$
$$v_y' = v\sin\phi'$$

where v is the fixed speed (20 m/s) of ownship. After getting the new velocity, adding it to the current position of the ownship will yield the position of ownship at the next state.

A simple example is given to illustrate how the state is updated. Assume the current state is

$$(i_x, i_y, i_{vx}, i_{vy}, o_x, o_y, o_{vx}, o_{vy}, o_\phi, g_x, g_y)$$

and the chosen action is turning left, which means $a = +2°/s$, then the next state

$$(i_x', i_y', i_{vx}', i_{vy}', o_x', o_y', o_{vx}', o_{vy}', o_\phi', g_x', g_y')$$

can be determined as follows:

$$i_{vx}' = i_{vx}$$
$$i_{vy}' = i_{vy}$$
$$i_x' = i_x + i_{vx}'$$
$$i_y' = i_y + i_{vy}'$$
$$o_\phi' = o_\phi + a \times \Delta t$$
$$o_{vx}' = v\cos o_\phi'$$
$$o_{vy}' = v\sin o_\phi'$$
$$o_x' = o_x + o_{vx}'$$
$$o_y' = o_y + o_{vy}'$$
$$g_x' = g_x$$
$$g_y' = g_y$$

where $\Delta t = 1s$ is the fixed time step, $v = 20m/s$ is the fixed speed for ownship.

Note that this process is not strictly following the kinematic model discussed before. But this approximation can perform well, especially when the time step is very small.

In addition, in the simulation step, the ownship needs to simulate the whole process to a terminal state, after which it

can get an evaluation of this generated trajectory. In this way, after simulating enough times, actions leading to the goal state will have a higher value and actions leading to conflicts won't be preferred.

The final step of MCTS is **backpropagation**. After simulating the whole process to one terminal state, the final reward should be given to the newly added nodes, and this score should be backed up to all the parents of the newly added nodes.

One iteration of the above four steps is called one simulation. If the computation budget allows (e.g., the decision needs to be made in 100ms), enough simulations will be repeated, which can yield very accurate approximation of the value of different nodes. And the number of simulations can be also decided before the MCTS algorithm runs. When the simulation stops and the decision needs to be made, the most promising node can be easily selected by solely performing exploitation (set $C = 0$).

B. *Estimated Value Function*

The only remaining concern now is that the ownship may not have enough time to run this algorithm, since the algorithm presented in this paper is an online algorithm, which means it is vital for the ownship to think quickly to make decisions. Since simulating this process to a terminal state usually needs many steps, simulating this process to a fixed depth d is beneficial to reduce computation time in the simulation step. If the algorithm simulates the process to the fixed depth d, which is a non-conflict state, the ownship should be able to estimate the value of this state. Intuitively, if this non-terminal state is closer the goal state, this state should be a better state if there is no other information, so the following estimated value function is used for the non-terminal states, so that the ownship can judge the goodness of any non-terminal state:

$$\tilde{V}(s) = 1 - \frac{d(o,g)}{\max d(o,g)} \text{ if } s \text{ is non-terminal state}$$

where $d(o,g)$ denotes the distance from ownship to goal position. And $\max d(o,g)$ is the maximum distance from ownship to the goal state, which is just the diagonal distance of the map. In this way, if there is no conflict with intruder or the border (which has reward 0), the ownship will get a positive reward, depending on how far the ownship is from the goal state.

The above procedure is summarized in Algorithm 1.

Algorithm 1 MCTS-UCT algorithm

```
function UctSearch(s_0)
    create root node v_0 with state s_0
    while within computational budget do
        v_l ← TreePolicy(v_0)
        Δ ← DefaultPolicy(s(v_l))
        Backup(v_l, Δ)
    return a(BestChild(v_0, 0))

function TreePolicy(v)
    while v is nonterminal and d(s(v)) ≤ d do
        if v not fully expanded then
            return Expand(v)
        else
            v ← BestChild(v, C)
    return v

function Expand(v)
    choose a ∈ untried actions from A(s(v))
    s(v') = Proceed(s(v), a)
    add the new child v' to v
    return v'

function BestChild(v, c)
    return argmax_{v' ∈ children of v}  Q(v')/N(v') + c sqrt(2 ln N(v) / N(v'))

function DefaultPolicy(s)
    while s is nonterminal and d(s) ≤ d do
        choose a ∈ A(s) uniformly at random
        s ← Proceed(s, a)
    return reward for state s

function Backup(v)
    while v is not null do
        N(v) ← N(v) + 1
        Q(v) ← Q(v) + Δ
        v ← parent of v

function Proceed(s, a)
    s' ← next state from current s, a
    d(s') ← d(s) + 1
    return s'
```

V. Numerical Experiments

A. *Simulator*

To test the performance of the proposed algorithm, a simulator was built where multiple aircraft can fly freely in the two-dimensional map. The map has 11km length and 8km width.

At the beginning, the position of ownship is randomly generated, the speed of the ownship is fixed at 20 m/s, the flight direction is uniformly generated between 0° and 360°. Then a set of intruder aircraft are generated with speed uniformly distributed between 10 m/s and 20 m/s and with direction uniformly distributed between 0° and 360°. The goal position is also uniformly generated in the map. Based on the assumption before, the intruder aircraft can only fly at fixed velocity with fixed direction. Whenever any intruder flies out of the map, a new intruder is randomly generated with the same parameters described before. Whenever there is a collision between ownship and any intruder, the intruder will disappear, and a new intruder is generated, and currently the conflicts between intruders are not addressed in this paper. In this way, the number of intruders will always be fixed. In

addition, the intruder shouldn't appear too close to ownship, in which case the ownship might not be able to avoid this intruder no matter what action it takes. When the ownship reaches the goal, a new goal will be generated. Finally, if the ownship flies out of the map, a new ownship will be generated. The number of goals the ownship has reached and the number of conflicts it has with other intruders will be recorded during the process.

B. Experiment setting

In the MCTS algorithm, there are two parameters which are important to the performance of this algorithm: the number of simulations n each time a decision needs to be made and the search depth d, which is discussed in Section IV. In the first experiment, the number of intruders is fixed to be 20, and the number of simulations is ranged from 100 to 1000 with step length 100, and the search depth is chosen from three numbers: 2, 3, and 4. In 10,800 time steps, the total number of conflicts with intruders, and the number that reach goal positions are tracked and compared. These statistics will be referred to as the algorithm performance. In addition, the computation time of this algorithm used for each decision made is also compared. The performance and running time will be considered to pick the best parameters n and d. Then in the second experiment, with the chosen parameter, the number of intruders will be ranged from 10 to 80 to test the ability of this algorithm.

C. Result

In Fig 3 and Fig 4, the performance of different parameters is compared. Fig 3 shows that the search depth $d=2$ performs much worse than larger search depths. Fig 4 shows the more simulations that are run, the better the algorithm performs, since the goal is reached more often. These conclusions are consistent with the MCTS algorithm (more simulations and deeper search depth can yield better solution).

Although there were some conflicts between the ownship and the intruders, there were no cases of near midair collisions (NMACs) in this simulations. Here, the NMAC is defined to be when two aircraft are closer than 30m.

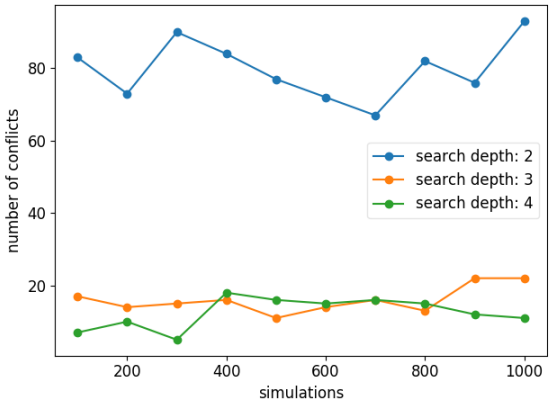

Figure 3. Number of conflicts with intruders in 10,800 time steps. Conflict is defined as the distance between two aircraft is less than 320m.

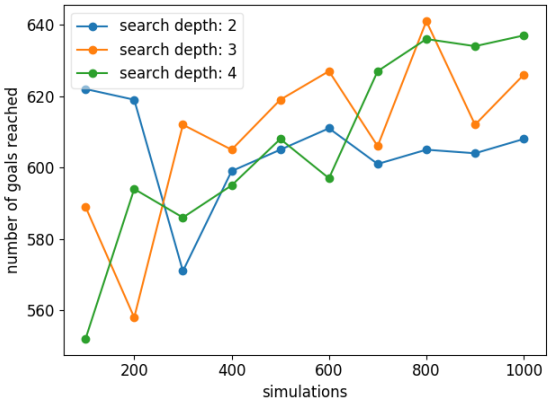

Figure 4. Number of goals reached in 10,800 time steps.

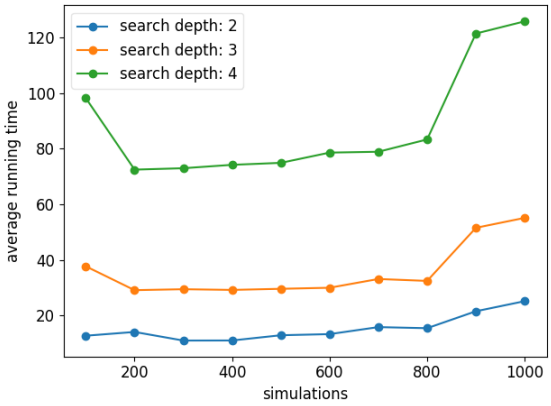

Figure 5. The running time for different parameter.

For the running time, the simulator was run for 1,000 time steps for each chosen parameter, and then the average value was taken to approximate the running time for each decision. The results are plotted in Fig 5, from which it can be seen that search depth is main factor for slowing down this algorithm. And the running time under 100ms is acceptable for a real time decision making system, given that the decision are made every 1 second.

After comparing the results in Fig 3, 4, and 5, $n=800$ and $d=3$ were chosen for the second experiment with different numbers of intruders. In this second experiment, the numbers of NMACs, conflicts and goals reached are compared in 10,800 time steps.

From Fig 6, it can be shown that with the increase of number of intruders, the number of conflicts is increasing exponentially, and the number of goals reached is decreasing, since there are more intruders blocking the way to the goal state and the ownship needs more time to avoid them. The

number of NMACs is very small, which means this algorithm is very effective at avoiding potential collisions when the intruder size is within a reasonable range.

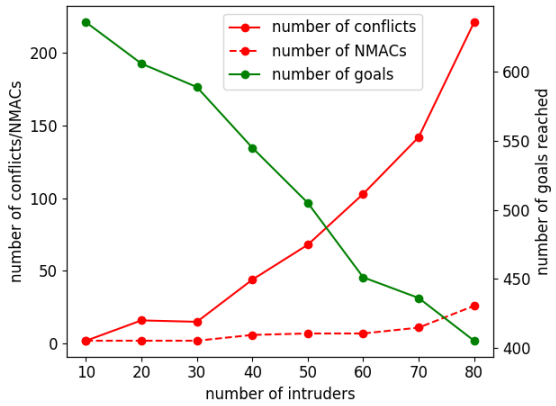

Figure 6. The performance of MCTS (with $d=3, n=800$) by increasing the number of intruder aircraft. NMAC is defined as the distance between two aircraft is less than 30m.

VI. Conclusion

A computational guidance algorithm with collision avoidance capability for autonomous on-demand free flight operations in urban air mobility is proposed in this paper. The problem is formulated as a Markov Decision Process (MDP) with the action of the turning angle of the aircraft. The problem is then solved by using Monte Carlo Tree Search (MCTS) algorithm. Numerical experiments show that this proposed algorithm has promising performance to help an aircraft reach its destination and avoid potential conflicts even in relatively dense air traffic scenarios. This proposed algorithm provides a potential solution framework to enable autonomous on-demand free flight operations in urban air mobility.

The contribution of this research is integrating the power of onboard aircraft intelligence (vehicle autonomy technology) and the advantage of the free flight concept for airspace operations to enable safe and efficient flight operations in on-demand urban air transportation.

Meanwhile, it should also be noted that the proposed algorithm is only tested under a simplified scenario, where all the intruders will execute their flight intention in a deterministic way. Future work will test the algorithm in the environment with uncertainties in the aircraft dynamics and intention execution.

Ackowledgement

This work has benefited from discussions with Mykel Kochenderfer at Stanford and Karthik Balakrishnan and Richard Golding at Airbus A^3. The authors thank Mykel Kochenderfer for inspiring the research idea and the A^3 Altiscope team for their guidance and support throughout this work.

References

[1] Gipson, Lillian. 2017. “NASA Embraces Urban Air Mobility, Calls for Market Study.” NASA. November 7. Accessed January 19. https://www.nasa.gov/aero/nasa-embraces-urban-air-mobility.

[2] “Uber Elevate | The Future Of Urban Air Transport.” 2018. Accessed January 19. https://www.uber.com/info/elevate/

[3] Holden, J., and N. Goel. "Fast-Forwarding to a Future of On-Demand Urban Air Transportation." San Francisco, CA (2016).

[4] “Urban air mobility.” 2017. Airbus. June 20. http://airbus-xo.com/urban-air-mobility/.

[5] “Future of urban mobility.” 2018. Airbus. Accessed January 19. http://www.airbus.com/newsroom/news/en/2016/12/My-Kind-Of-Flyover.html.

[6] Hoekstra, Jacco M., Ronald NHW van Gent, and Rob CJ Ruigrok. "Designing for safety: the ‘free flight’ air traffic management concept." Reliability Engineering & System Safety 75, no. 2 (2002): 215-232.

[7] Clari, Mario SV Valenti, Rob CJ Ruigrok, Jacco M. Hoekstra, and Hendrikus G. Visser. "Cost-benefit study of free flight with airborne separation assurance." Air Traffic Control Quarterly 9, no. 4 (2001): 287-309.

[8] Krozel, Jimmy, and Mark Peters. "Conflict detection and resolution for free flight." Air Traffic Control Quarterly 5, no. 3 (1997): 181-212.

[9] Kuchar, James K., and Lee C. Yang. "A review of conflict detection and resolution modeling methods." IEEE Transactions on intelligent transportation systems 1, no. 4 (2000): 179-189.

[10] Schouwenaars, Tom, Mario Valenti, Eric Feron, and Jonathan How. "Implementation and flight test results of MILP-based UAV guidance." In Aerospace Conference, 2005 IEEE, pp. 1-13. IEEE, 2005.

[11] Mellinger, Daniel, Alex Kushleyev, and Vijay Kumar. "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams." In Robotics and Automation (ICRA), 2012 IEEE International Conference on, pp. 477-483. IEEE, 2012.

[12] Augugliaro, Federico, Angela P. Schoellig, and Raffaello D'Andrea. "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach." In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 1917-1922. IEEE, 2012.

[13] Kochenderfer, Mykel J., and J. P. Chryssanthacopoulos. "Robust airborne collision avoidance through dynamic programming." Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371 (2011).

[14] Chryssanthacopoulos, James P., and Mykel J. Kochenderfer. "Decomposition methods for optimized collision avoidance with multiple threats." Journal of Guidance, Control and Dynamics 35, no. 2 (2012): 398-405.

[15] Ong, Hao Yi, and Mykel J. Kochenderfer. "Markov Decision Process-Based Distributed Conflict Resolution for Drone Air Traffic Management." Journal of Guidance, Control, and Dynamics (2016).

[16] Browne, Cameron B., Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games 4, no. 1 (2012): 1-43.

[17] “Unmanned Aerial System Traffic Management Fact Sheet,” 2015.

[18] Kocsis, Levente, Csaba Szepesvári, and Jan Willemson. "Improved monte-carlo search." Univ. Tartu, Estonia, Tech. Rep 1 (2006).