Distributed Computational Guidance for High-Density Urban Air Mobility with Cooperative and Non-Cooperative Collision Avoidance

Joshua R. Bertram   Peng Wei
Iowa State University
Ames, IA 50011
{bertram1, pwei}@iastate.edu

Urban Air Mobility (UAM) will result in numerous aircraft which will need to respond to a dynamically changing airspace to safely reach their destinations. We use a highly efficient Markov Decision Process (MDP) based trajectory planner to demonstrate multi-agent distributed trajectory planning that can safely avoid cooperative and non-cooperative actors in the high-density free flight airspace. We demonstrate the algorithm in an urban setting where multiple aircraft all navigate to their designated landing site while avoiding cooperative and non-cooperative aircraft. We study the algorithm performance as the aircraft density increases to demonstrate scalability and the effect of cooperative versus non-cooperative actors in the environment.

I. Introduction

Urban Air Mobility (UAM) is a concept for future air transportation in which partially or fully autonomous air vehicles transport cargo and passengers through dense urban environments. For early adopters this technology offers the promise of bypassing ground-based freeways and hours-long commutes. As the technology matures, it will connect urban centers with outlying towns extending the reach of metropolitan areas.

As we contemplate this future, an important question to consider is how will this air traffic be managed? Will a structured airspace be required to enforce some order on the aircraft similar to today's air traffic management system? Or can improvements in technology allow a more dynamic, less structured airspace design to be used?

This paper offers a demonstration of an algorithm that can efficiently provide guidance to aircraft operating in high-density airspace containing many aircraft. The algorithm allows each aircraft to make its own decisions in a distributed manner using simple inputs as would be available from sensors such as radar, LIDAR or systems such as ADS-B. The computational burden is very light due to the algorithm's efficiency and is capable of running on embedded processors found in UAV and UAM aircraft.

The algorithm works in a *cooperative* setting, meaning that all aircraft in the airspace are using the algorithm from this paper, and also works in a *non-cooperative* setting, meaning that one or more aircraft in the airspace are not performing any avoidance. A cooperative setting is the most desirable case as both aircraft can participate in avoidance to ensure separation. However, it requires active participation from every aircraft and is vulnerable to fault conditions or rogue actors. If an aircraft's collision avoidance system fails due to a fault condition or sensor failure, it will not behave as expected or may not perform any avoidance at all. We demonstrate the algorithm's performance in both cases.

We also test the algorithms ability to scale to higher densities, while acknowledging that there is some upper limit to the number of aircraft that can be supported by a given airspace due to physical constraints. We perform simulations to evaluate the algorithm under different densities while monitoring for Near Mid-Air Collisions (NMACs). And finally we provide timing measurements of the performance of the algorithm.

II. Related Work

NASA, Uber and Airbus have been exploring the use of vertical takeoff and landing (VTOL) aircraft for Urban Air Mobility (UAM) [1–5]. In general, the UAM concept calls for UAM aircraft taking off from small-scale airports known as vertiports where VTOL aircraft depart and arrive.

An unstructured airspace approach known as “free flight" has been proposed to cope with the ongoing congestion of the current ATC system. It was shown in [6, 7] that free flight with airborne separation is able to handle a higher traffic density, and [8] found free flight can also bring fuel and time efficiency. In a free flight framework, each aircraft is responsible for separation assurance and conflict resolution. [9] shows that free flight is potentially feasible due to enabling technologies such as Global Positioning Systems (GPS), data link communications like Automatic Dependence Surveillance-Broadcast (ADS-B) [10], Traffic Alert and Collision Avoidance Systems (TCAS) [11], but would require powerful onboard computation.

In terms of algorithms used for trajectory planning and collision avoidance, there is extensive literature on the topic. In centralized methods, a central supervising controller resolves conflicts between aircraft. The state of each aircraft, obstacles, and trajectory constraints as well as the state of the terminal area are observable to the controller via sensors, radar, etc. The central controller precomputes trajectories for all aircraft before flight, typically by formulating the problem in and optimal control framework and solving the problem with varying methods; examples are: semidefinite programming [12], nonlinear programming [13, 14], mixed integer linear programming [15–18], mixed integer quadratic programming [19], sequential convex programming [20, 21], second-order cone programming [22], evolutionary techniques [23, 24], and particle swarm optimization [25]. One common thread among centralized approaches is that in order to pursue a global optimum, they must consider each aircraft and obstacle in the space. This necessarily leads to scalability issues with large numbers of aircraft and obstacles. Also, as new aircraft enter the scene, centralized algorithms typically need to recompute part or all of the problem in order to arrive at a new global optimum.

In contrast, decentralized methods scale better with respect to the number of aircraft and objects in the system, but typically cannot obtain globally optimal solutions. However, decentralized methods typically do not have a single point of failure, so may be considered more robust than a centralized approach [26]. Conflicts are resolved by each aircraft locally in decentralized approaches and the underlying method can be considered as cooperative or non-cooperative. Examples of cooperative methods using some form of communication are [27–30] which typically use the communication to make smaller optimal control problems that can be solved locally. Other approaches such as [31, 32] use time slices to compute collision free paths one agent at a time.

Computational geometry methods such as visibility graphs [33] and Voronoi diagrams [34] can also generate paths for aircraft, but typically do not take dynamic constraints into account. Within the robotics community several sampling based dynamics aware methods have been created: probabilistic roadmaps [35], RRT [36], RRT* [37], and RT-RRT* [38] (which may be applied in centralized and decentralized contexts.) These methods attempt to generate a connected graph or tree of collision-free states that are very near each other so that a path from a start to a goal position can be determined via a path through the graph or tree. The advantage of these methods is they can discover conflict-free paths through high dimensional spaces with many obstacles, but the graph/tree must either be pre-computed or in the case of RT-RRT* can be computed on-line by fixing the amount of computation allowed during any compute cycle. While none of these methods compute a truly optimal path with a finite number of samples, with enough samples the paths can be good enough for many problems and can further be smoothed after extraction from the graph/tree.

Collision avoidance can be solved with Model Predictive Control [39, 40] with high computational costs, and potential fields [41, 42] which are fast but do not provide guarantees of collision avoidance. Machine learning methods have also been used [43–46]. The methods perform well but require a great deal of time to train beforehand. The Monte Carlo Tree Search method used in [47] solves the problem efficiently by using a fixed window of computation. Collision avoidance can also be solved geometrically [48–51] and the computation time only grows linearly as the number of aircraft increases. DAIDALUS (Detect and Avoid Alerting Logic for Unmanned Systems) [52] is another geometric approach developed by NASA which provides heading and altitude guidance to avoid collisions to remote pilots. ACAS-X is a collision avoidance system also based off Markov Decision Processes that provides 1-on-1 collision avoidance [53].

Within the framework of the literature, the method in this paper phrases the problem as a Markov Decision Process (MDP). MDPs are known to be intractable for large problems, but a recently discovered method for efficiently computing them was described in [54]. We harness this implementation and demonstrate its efficient performance in a UAM context with multiple UAM vehicles navigating between multiple vertiports.

III. Background

Markov Decision Processes (MDPs) are a framework for sequential decision making with broad applications to finance, robotics, operations research and many other domains [55]. MDPs are formulated as the tuple (s_t, a_t, r_t, t) where $s_t \in S$

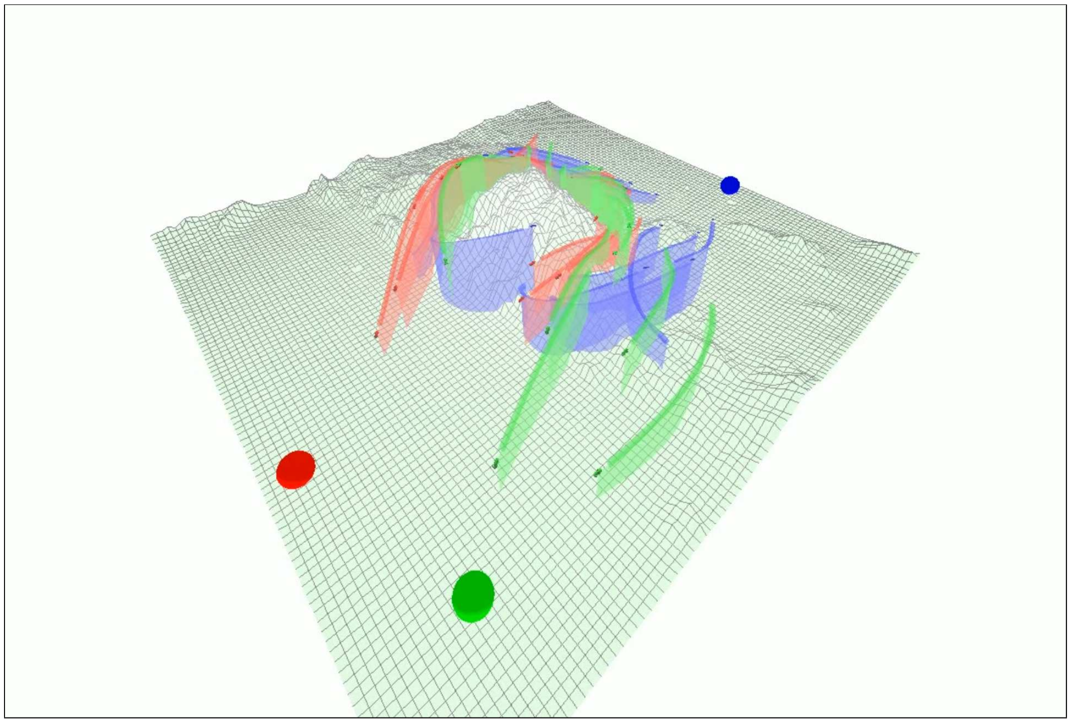

Fig. 1 Multiple aircraft with multiple goals. Colored spheres represent the goals. Airplanes and their history trails are colored according to their goal. Negative rewards are placed around terrain features so that UAVs avoid collision with terrain.

is the state at a given time t, $a_t \in A$ is the action taken by the agent at time t as a result of the decision process, r_t is the reward received by the agent as a result of taking the action a_t from s_t and arriving at s_{t+1}, and $T(s_t, a, s_{t+1})$ is a transition function that describes the dynamics of the environment and capture the probability $p(s_{t+1}|s_t, a_t)$ of transitioning to a state s_{t+1} given the action a_t taken from state s_t.

A policy π can be defined that maps each state $s \in S$ to an action $a \in A$. From a given policy $\pi \in \Pi$ a value function $V^{\pi}(s)$ can be computed that computes the expected return that will be obtained within the environment by following the policy π.

The solution of an MDP is termed the optimal policy π^*, which defines the optimal action $a^* \in A$ that can be taken from each state $s \in S$ to maximize the expected return. From this optimal policy π^* the optimal value function $V^*(s)$ can be computed which describes the maximum expected value that can be obtained from each state $s \in S$. And from the optimal value function $V^*(s)$, the optimal policy π^* can also easily be recovered.

IV. Method

We formulate the problem as a Markov Decision Process and use the algorithm described in [54] to efficiently solve the MDP. The algorithm in [54] was shown to efficiently perform collision avoidance while navigating to a goal in a 2D discretized state space. We extend [54] by moving to a 3D state space which is fully continuous and model the aircraft in 3D using a Dubin's aircraft model which is constrained to operate in a manner similar to a UAM air taxi in forward flight mode.

A. Dynamic Model

[56] proposes an extension of the Dubin's car model to aircraft using an optimal control framework relying on Pontraygin's Maximum Principle. [57] extends the model using a more realistic formulation from an aerospace engineering literature.

Table 1 Limits on aircraft performance to approximate an air taxi.

V_{min} (Kts)	V_{max} (Kts)	$\dot{\psi}_{\min}$ (deg/s)	$\dot{\psi}_{\max}$ (deg/s)	$\alpha_{\min}$ (deg)	$\alpha_{\max}$ (deg)	$\phi_{\min}$ (deg)	$\phi_{\max}$ (deg)	$\gamma_{\min}$ (deg)	$\gamma_{\max}$ (deg)
47	133	-30	30	-5	20	-20	20	-20	20

Note that the model assumes an airspeed controller that maintains a commanded airspeed through the duration of flight. A pseudo-6DOF formulation is proposed in [58] and [59].

The model used here is based off a pseudo-6DOF formulation, but uses performance limits that make the aircraft perform similar to a Dubin's aircraft with gentle climbs and bank angles suitable for a UAM passenger aircraft.

- n_x: Throttle acceleration directed out the nose of the aircraft in g's
- V: Airspeed in meters/second.
- γ: Flight path angle in radians.
- x, y, z: position in NED coordinates in meters where altitude $h = -z$
- ϕ: Roll angle in radians
- ψ: Horizontal azimuth angle in radians
- α: Angle of attack in radians with respect to the flight path vector

The inputs to the model are: (1) the thrust n_x, (2) the rate of change of angle of attack $\dot{\alpha}$ and (3) the rate of change of the roll angle $\dot{\phi}$.

The equations of motion for the aircraft are:

$$\dot{V} = g\left[n_x \cos\alpha - \sin\gamma\right], \tag{1}$$

$$\dot{\gamma} = \frac{g}{V}\left[n_f \cos\phi - \cos\gamma\right], \tag{2}$$

$$\dot{\psi} = g\left[\frac{n_f \sin\phi}{V\cos\gamma}\right], \tag{3}$$

where the acceleration exerted out the top of the aircraft n_f in gs is defined as:

$$n_f = n_x \sin\alpha + L, \tag{4}$$

with a lift acceleration of $L = 0.9$. Here, 1 "g" is a unit of acceleration equivalent to 9.8 m/s^2. L was chosen to provide some amount of lift while in flight to counteract gravity and provide a stable flight condition with a low positive α angle of attack in the pseudo-6dof model.

The kinematic equations are:

$$\dot{x} = V\cos\gamma\cos\psi \tag{5}$$

$$\dot{y} = V\cos\gamma\sin\psi \tag{6}$$

$$\dot{z} = V\sin\gamma. \tag{7}$$

While this model is not aerodynamically comprehensive, it is sufficient to describe aircraft motion suitable for examining the algorithm behavior without loss of generality. The algorithm can integrate with any aircraft dynamics model that allows the trajectory to be computed forward in time.

Performance limits are used to make the aircraft dynamics perform like a Dubin's aircraft similar to what may be expected from a UAM air taxi, as shown in Table 1.

1. Trajectory Forward Projection

At each time step, each aircraft has a set of possible actions it can take and each of those actions will lead to different future states that the aircraft may reach. When we use the aircraft dynamics to compute the result of taking an action for a fixed amount of time, this is known as forward projection. When successive timesteps over a time period t are used to show the future trajectory resulting from an action, this is known as trajectory forward projection with window t.

If the actions $a \in A$ were fully continuous, we could describe the result of the forward projection of all actions as the reachability set, $R \subset S$ from our initial state $s \in S$. However, for computational efficiency we use a discrete set of actions $\hat{a} \in A$. When we use forward projection on this discrete set of actions, we obtain an approximation of the reachabililty set $\hat{R} \subset R$. If the discrete actions and timesteps are chosen wisely, they can however be a reasonable and useful approximation of the reachability set.

For this problem, we use a discrete time step of 0.1 seconds and a window of 3.0 seconds. For the trajectory forward projection, we keep the action fixed over the window and generate one trajectory for each action $\hat{a}$. Once the forward projection is completed, the resulting states from the trajectories form an approximation of the reachability set $\hat{R}$. The value of each of these states $s \in \hat{R}$ are computed within the solution of the MDP, which represents the points within the value function that are reachable within the trajectory forward projection window. If we compute the state with maximum value and then determine the action which results in that state, we then know the most valuable action to take from our current state. This action is then applied for the simulation's 0.1 second time step resulting in a 10 Hz rate action selection.

Thus the agent follows the optimal policy of the MDP at each time step by determining which future reachable state is most valuable, and then takes the action in the next time step that will lead it towards that state.

B. MDP Formulation

1. State Space

We define the environment where the aircraft operates within a 25 km by 25 km by 25 km volume which is treated as a continuous state space. The state includes all the information each aircraft needs for its decision making: the full aircraft state, the position and velocity of every other aircraft. Each aircraft is aware of its own state produced by the aircraft dynamics model. For each aircraft, the state is formed by concatenating the following:

- ζ the aircraft state: position x, y, z, the heading angle ψ, the flight path angle γ, and the speed V.
- for each other aircraft $f_j, \forall j \in J$: the position $f_{j,x}, f_{j,y}, f_{j,z}$ and velocity $f_{j,v_x}, f_{j,v_y}, f_{j,v_z}$,

$$s_o = [\zeta, f_1, \cdots, f_j] \tag{8}$$

where j represents the number of other aircraft.

2. Action Space

Inputs to the dynamics model are (1) the thrust n_x, (2) the rate of change of angle of attack $\dot{\alpha}$ and (3) the rate of change of the roll angle $\dot{\phi}$.

The action space is a discrete set of actions that are distributed logarithmicly through the range of each input. The logarithmic distribution is designed to have a fine granularity of inputs near the zero point of the input and a coarse granularity of inputs near the extreme ends of the input range. The philosophy here is that more control is needed when trying to make small corrections to correct for small perturbations or refinements of the trajectory, and that large control inputs are required for gross course corrections.

The action space is then defined as follows for both pitch inputs and roll inputs: 15 discrete inputs, 7 of which are positive, 7 of which are negative, and the value 0. The 7 positive and 7 negative inputs are mirror values of each other and are based off a linear spacing of coefficients κ from .0001 to .13:

$$\kappa = \{.0001, .02175, .0434, .06505, .0867, .10835, .13\} \tag{9}$$

The logarithmically spaced values in radians are then computed as follows:

$$\begin{aligned} k_r &= 10^{\kappa} - 1 \qquad (10) \\ &= \{0.00023, 0.051, 0.105, 0.162, 0.221, 0.283, 0.349\} \qquad (11) \end{aligned}$$

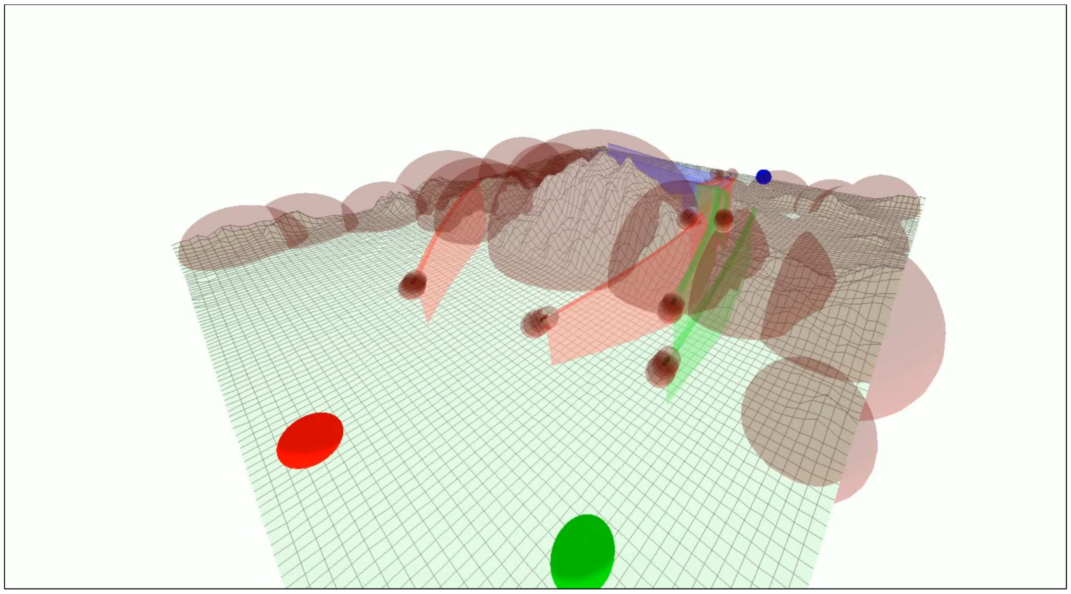
Fig. 2 Visualization of negative rewards. Each aircraft has negative rewards positioned in front of it for collision avoidance with other aircraft. Terrain features are covered by negative rewards which were manually placed and sized for terrain avoidance. A similar technique can be used for weather.

Or in degrees:

$$k_d = \{0.013, 2.943, 6.022, 9.258, 12.660, 16.236, 19.994\} \tag{12}$$

Making the final distribution in degrees:

$$\begin{aligned} K_d &= \{-k_d, 0, k_d\} && (13)\\ &= \{-19.99, -16.24, -12.66, -9.26, -6.02, -2.94, -0.01, 0, 0.01, 2.94, 6.02, 9.26, 12.66, 16.24, 19.99\}, && (14)\end{aligned}$$

with the pitch and roll inputs both equal to these logarithmic distributions:

$$\begin{aligned} \dot{\alpha} &= K_d && (15)\\ \dot{\phi} &= K_d && (16)\\ & && (17)\end{aligned}$$

n_x is a simple linear distribution: $\{-2, -1, 0, 1, 2, 3, 4\}$.
The joint action space is then:

$$A = \{\dot{\alpha}, \dot{\phi}, n_x\}. \tag{18}$$

3. Reward Function

The primary mechanism to control the behavior of an agent in a Markov Decision Process (MDP) is through the Reward Function. By providing positive and negative rewards to the agent, it is able to determine which actions lead to positive reward and the solution of an MDP maximizes the expectation of future reward.

Following the approach used in [54], we will treat each negative reward as a “risk well", which is a region of negative reward (i.e., a penalty) which is more intense at the center and decays outward until a fixed radius is reached, where after no penalty is applied.

We present our reward function in terms of the behaviors we wish to obtain in Table 2. In this table, $\hat{p}$ represents the current position of an aircraft and $\hat{v}$ represents that aircraft's current linear velocity. In some cases we project the aircraft's position forward in time with an expression $\hat{p} + \hat{v}t$ and then define a range of time as in $\forall t \in \{0, 1, 2\}$ to

Table 2 Rewards created for each aircraft

For each other aircraft:

Magnitude	Decay factor	Location	Radius	Timesteps	Comment
-1000	.97	$\hat{p} + \hat{v}t$	$300 + 10t$	$\forall t \in \{-5, 0, 5, 10, 15\}$	Collision avoidance, 5 rewards

For each terrain feature:

Magnitude	Decay factor	Location	Radius	Timesteps	Comment
-1000	.99	manually placed	manually selected	N/A	Terrain avoidance

For aircraft's goal:

Magnitude	Decay factor	Location	Radius	Timesteps	Comment
200	.999	manually placed	∞	N/A	Vertiport attraction

indicate that we create a reward at the location of the aircraft at each timestep in the future indicated by the range of t. See Figure 2 for a visualization of the rewards.

All aircraft also receive a penalty below a certain altitude which prevents the aircraft from plummeting into the terrain. We define a terrain "floor" h_{floor} which is the 5th percentile of all terrain vertices. The NASA SRTM terrain data [60] contains some locations of bad data, so the 5th percentile is used to eliminate any outliers when determining a good average minimum height. For any state with an altitude of h from the hard deck up to an altitude of $h_{\text{penalty}} = h_{\text{floor}} + 200$, a penalty is applied $r_{\text{penalty}} = -(10000 - h)$ which is a very strong negative reward that will override any other positive rewards in the game.

To avoid collision with terrain, for this paper negative rewards were manually placed in the terrain such that they covered the terrain features with a hemisphere of negative reward, providing a repulsive force that prevents collision. (In future research, this could be made more automatic for true terrain or building avoidance.)

C. Algorithm

The algorithm in this paper is based off the algorithm in [54], which defines the concept of a *peak* in the value function that results from a reward in the space. Peaks are located in the state space at the locations of positive rewards. The algorithm takes as input a number of pre-allocated data structures that described the peaks:

- peakLocations: Locations of each peak within the state space.
- propLimits: A limit on how far the value function of each peak should be allowed to propagate. For our positive rewards in the state space, this is ∞. For negative rewards, this is a fixed radius that represents the distance at which another aircraft poses no risk.
- propLimits: A limit on how far the value function of each peak should be allowed to propagate. For our positive rewards in the state space, this is ∞. For negative rewards, this is a fixed radius that represents the distance at which another aircraft or terrain feature poses no risk.
- discountFactors: A parameter for each reward which allows the reward to decay outward as distance from the center increases.
- rwdMagnitudes: The magnitude of each of the rewards.
- states: The states at which the value function should be computed for this MDP. This allows flexibility of computing only neighboring states needed for following the optimal policy, or large batches for visualizing the value function for debug.

We reimplement the algorithm proposed in [54] to vectorize the algorithm for more efficient operation. A vectorized algorithm is one which is designed to perform identical computations at each step over a vector of data inputs and is often accelerated by SIMD (Same Instruction Multiple Data) units available in modern CPUs. Libraries are available that assist in mapping code to these accelerators. In our case, within python a library known as *Numpy* provides this

Algorithm 1 Distributed UAM using FastMDP algorithm

```
procedure DistributedUam(aircraftState, worldState)
    $\mathbf{S}_0 \leftarrow$ randomized initial aircraft states
    $\mathbf{A} \leftarrow$ aircraft actions (precomputed)
    $\mathbf{L} \leftarrow$ aircraft limits (precomputed)
    $\mathbf{S}_{t+1} \leftarrow$ allocated space
    while aircraft remain do
        for each aircraft do
            $s_t \leftarrow \mathbf{S}_t[aircraft]$
            // Build peaks per Table 2
            $\mathbf{P}^+ \leftarrow$ build pos rewards
            $\mathbf{P}^- \leftarrow$ build neg rewards in Standard Positive Form
            $\mathbf{P}^* \leftarrow$ build neg rewards for terrain in Standard Positive Form
            // Perform forward projection per Section IV.A.1
            $\mathbf{\Delta_1} \leftarrow fwdProject(s_t, \mathbf{A}, \mathbf{L}, 0.1\ s)$
            $\mathbf{\Delta_{10}} \leftarrow fwdProject(s_t, \mathbf{A}, \mathbf{L}, 1.0\ s)$
            // Compute the value at each reachable state
            $\mathbf{V}^* \leftarrow$ allocate space for each reachable state
            for $s_j \in \mathbf{\Delta_{10}}$ do
                // First for positive peaks
                for $p_i \in \mathbf{P}^+$ do
                    $d_p \leftarrow \left\| s_j - \mathbf{location}(p_i) \right\|_2$        ▹ distance
                    $r_p \leftarrow \mathbf{reward}(p_i)$
                    $\gamma_p \leftarrow \mathbf{discount}(p_i)$
                    $\mathbf{V}^+(p_i) \leftarrow |r_p| \cdot \gamma_p^{d_p}$
                end for
                $V^+_{max} \leftarrow \max_{p_i} \mathbf{V}^+$
                // Next for negative peaks (in Standard Positive Form) including terrain
                for $n_i \in \{\mathbf{P}^-, \mathbf{P}^*\}$ do
                    $d_n \leftarrow \left\| s_j - \mathbf{location}(n_i) \right\|_2$        ▹ distance
                    $\rho_n \leftarrow negDist_i < \mathbf{radius}(n_i)$        ▹ within radius
                    $r_n \leftarrow \mathbf{reward}(n_i)$
                    $\gamma_n \leftarrow \mathbf{discount}(n_i)$
                    $\mathbf{V}^-(p_i) \leftarrow int(\rho_n) \cdot |r_n| \cdot \gamma_n^{d_n}$
                end for
                $V^-_{max} \leftarrow \max_{p_i} \mathbf{V}^-$
                // Hard deck penalty
                if $\mathbf{altitude}(s_t) < penaltyAlt$ then
                    $V_{deck} \leftarrow 1000 - \mathbf{altitude}(s_t)$
                else
                    $V_{deck} \leftarrow 0$
                end if
                $\mathbf{V}^*[s_t] \leftarrow V^+_{max} - V^-_{max} - V_{deck}$
            end for
            // Identify the most valuable action
            $i_{max} \leftarrow \arg\max_s(\mathbf{V}^*)$
            // For illustration, the corresponding value
            $maxValue \leftarrow \mathbf{V}^*[i_{max}]$
            // And the next state when taking the action
            $s_{t+1} \leftarrow \mathbf{\Delta_1}[i_{max}]$
            $\mathbf{S}_{t+1}[aircraft] \leftarrow s_{t+1}$
        end for
        // Now that all aircraft have selected an action, apply it
        $\mathbf{S} \leftarrow \mathbf{S}_{t+1}$
    end while
end procedure
```

capability. Note that for true real-time embedded performance, the code would be ported to a language such as C or C++, but this is a reasonable approximation as *Numpy* relies on C and C++ libraries to perform its acceleration.

V. Experiment Setup

We demonstrate this trajectory planner in a 3D aircraft simulation showing a view of the aircraft, the goals, and the terrain as shown in Figure 1. The simulation covers a 25km by 25km by 25km volume which contains a configurable number of aircraft and goals. The terrain is derived from NASA SRTM radar data [60] in the Lake Tahoe, California area. The height has been exaggerated by a factor of 6 to represent a UAM operating area with terrain features which must be avoided with vertical maneuvering. While in this case the terrain features correspond to mountains, they could instead represent skyscrapers or population centers. In the simulations and videos, this forces the simulated UAM aircraft into more conflicts than would be observed with a flat environment. The location of the goals were manually selected to lie within flat, relatively featureless areas of the terrain and represent vertiports.

As described above, all aircraft are driven by the algorithm and are assigned a goal. The color of the aircraft corresponds to the color of their goal. At each time step (0.1 seconds), the simulation updates the state for each aircraft. Each aircraft creates and solves its own MDP using the highly efficient algorithm presented in this paper, and then uses the solution of the MDP to select its next action. The actions of all aircraft are performed simultaneously in the simulation at the beginning of the next time step, simulation then advances by one time step (.1 seconds), whereupon the process repeats. Note here that because the environment changes, a new MDP is calculated at each time step, which is made possible by the performance of the algorithm in [54].

We measure performance in two ways. First, we characterize the time it takes for the algorithm to compute the solution of the MDP. Second, we examine the algorithms ability to reach the goal while avoiding collisions. We define a Near Mid-Air Collision (NMAC) as an aircraft coming within 100 meters of another aircraft during flight. As we vary experimental parameters, we track the number of goals obtained and the number of NMACs that occur over the simulation duration.

VI. Results

Results of simulation comparing cooperative and non-cooperative experiments are shown in Figure 3. In these simulations, there are three color-coded teams of aircraft (red, green, blue) with each team's color according to which vertiport it is navigating to (rendered as a sphere with matching color) as shown in Figure 1. The number of aircraft in the simulation $\{3, 15, 30, \ldots\}$ results from the number of aircraft per team $\{1, 5, 10, \ldots\}$.

Over the cooperative simulation runs, no collisions between any aircraft occurred. For non-cooperative simulations, one-third of the aircraft were set to be non-cooperative intruders. These intruders are still run by the algorithm, but they were not presented with any rewards related to any other aircraft (including other intruders). While they can still successfully navigate to the goal while avoiding terrain, they are blind to other aircraft. Thus, any collisions or NMACs between two intruders are ignored during the non-cooperative simulation. As expected, NMACs increased with an increase in the number of blind intruders. However, despite the large numbers of intruders the remaining two-thirds of the aircraft (red and blue aircraft) managed to avoid collisions with themselves or the intruders remakably well.

Timing measurements were performed on a laptop class PC with an Intel i9-8950HK CPU running at 2.90 GHz with 32 GB RAM. For the tests the number of aircraft were increased and the time to execute the algorithm per aircraft is reported. Table 3 shows the results showing a linear increase in performance (meaning $O(n)$ performance) with increased number of aircraft. Note that our target execution time for a 10 Hz frame rate (100 ms period) is exceeded at 90 aircraft (30 aircraft per team) indicating that further optimization is needed to scale to larger numbers of aircraft. An alternate version of the algorithm is in development and early results show that it will dramatically improve the performance beyond what is reported here.

We show sample videos of the simulation to provide intuition on the complexity of the task and performance of the algorithm. Two videos are provided that show 30 aircraft, one of which shows the negative rewards for terrain in order to provide insight into how the algorithm functions. A third video is provided with 150 aircraft to provide insight into the density of the airspace. Note that in all three videos, the aircraft are rendered with a wingspan of 200 meters for improved visibility, though in simulation the actual wingspan is 5 meters.

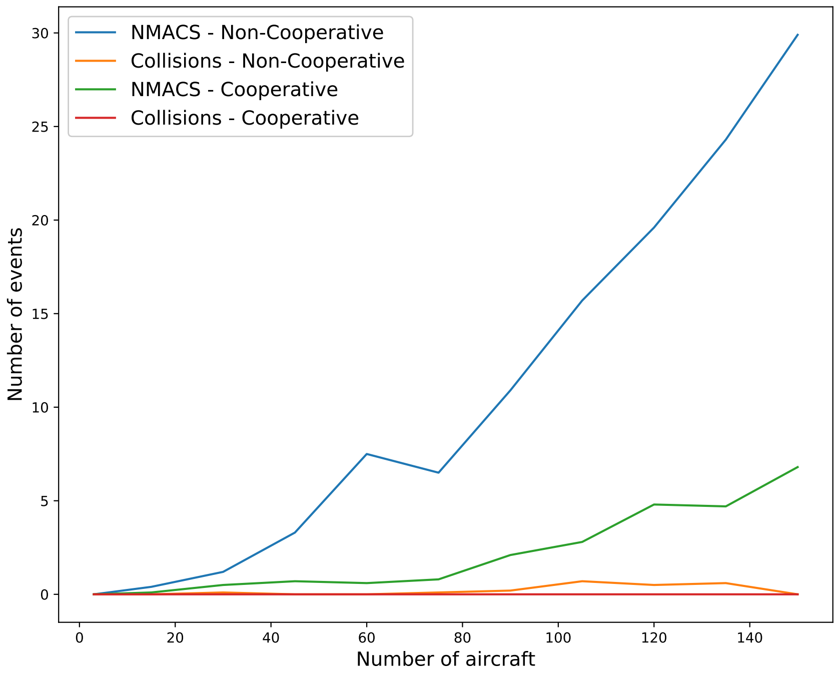

Fig. 3 Comparing results of cooperative flight versus non-cooperative flight. As expected, NMACS and Collisions increase when non-cooperative agents are present in the airspace. Despite the presence of multiple intruders, the number of collisions has been kept to a very low value. Values shown here are averaged over 10 Monte Carlo runs with random initial positions of aircraft. In the uncooperative measurement, $1/3$ of the aircraft are uncooperative and do not attempt to avoid any other aircraft, acting as blind intruders. Any collisions or NMACS between any two intruders are ignored as they may blindly fly into each other.

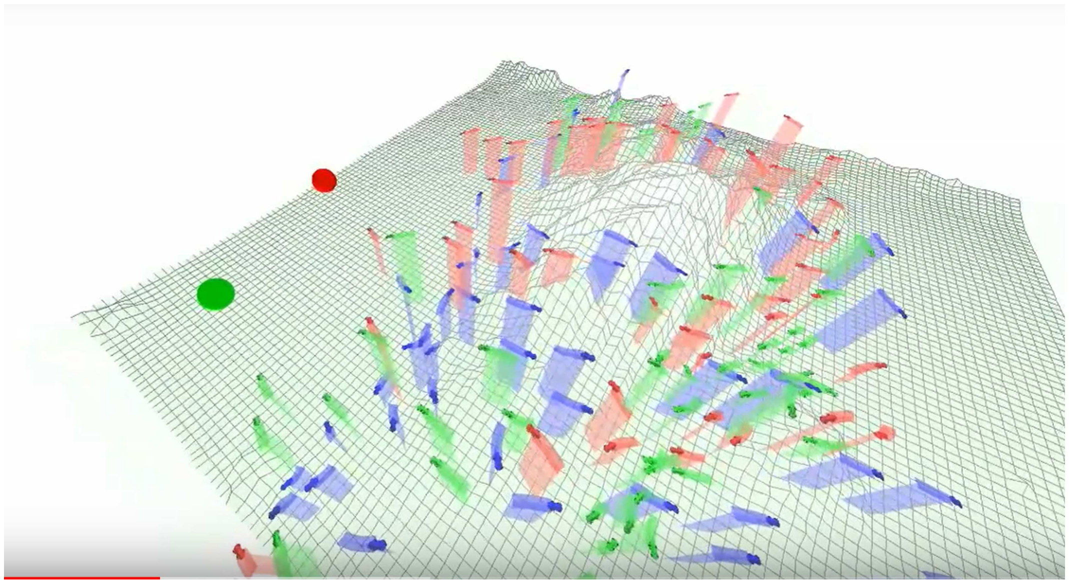

Fig. 4 Example of 150 agents to illustrate the difficulty of avoiding NMACs and collisions. Agents are rendered with 200 meter wingspan for improved visibility, though in simulation actual wingspan of air taxi is 5 meters.

Table 3 Timing measurements of algorithm performance for each frame as the number of aircraft increases.

Number of aircraft	Timing (ms)
3	34.9
15	43.9
30	58.1
45	67.4
60	77.1
75	95.2
90	116.4
105	123.6
120	137.7
135	149.8
150	162.6

Table 4 Links to videos

Number of aircraft	URL	Comment
30	`https://youtu.be/vX1BUC9bLFU`	Rewards not rendered
30	`https://youtu.be/7NXnl5cFWnM`	Rewards are rendered
150	`https://youtu.be/_B9Ath-3gQI`	Rewards not rendered

VII. Conclusion

In this paper we have demonstrated a computationally efficient distributed computational guidance algorithm suitable for high-density Urban Air Mobility applications. The algorithm can work in both cooperative and non-cooperative collision avoidance contexts. We compare operation of collision avoidance in both scenarios and show that the algorithm successfully avoids collisions even with large numbers of non-cooperative agents in the environment. We show that performance is suitable for embedded real-time applications, but to scale to large numbers of aircraft additional optimizations are required. Future research is already under way to further optimize the algorithm and early results show a dramatic improvement which will permit faster frame rates while scaling to larger numbers of aircraft and other obstacles.

In other future research, a method to automatically construct negative rewards from terrain should be pursued and alternative ways of capturing the terrain efficiently should be explored. Risk wells seem very appropriate for small obstacles such as other aircraft, but a new approach will be needed to efficiently represent terrain.

References

[1] Gipson, L., "NASA Embraces Urban Air Mobility, Calls for Market Study," `https://www.nasa.gov/aero/nasa-embraces-urban-air-mobility`, 2017. Accessed: 2018-01-19.

[2] "Uber Elevate | The Future of Urban Air Transport," `https://www.uber.com/info/elevate/`, 2017. Accessed: 2018-08-13.

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

[4] "Urban Air Mobility," `http://publicaffairs.airbus.com/default/public-affairs/int/en/our-topics/Urban-Air-Mobility.html`, 2018. Accessed: 2018-08-13.

[5] "Future of Urban Mobility," http://www.airbus.com/newsroom/news/en/2016/12/My-Kind-Of-Flyover.html, 2017. Accessed: 2018-08-13.

[6] Hoekstra, J. M., van Gent, R. N., and Ruigrok, R. C., "Designing for safety: the 'free flight'air traffic management concept," *Reliability Engineering & System Safety*, Vol. 75, No. 2, 2002, pp. 215–232.

[7] Bilimoria, K. D., Grabbe, S. R., Sheth, K. S., and Lee, H. Q., "Performance evaluation of airborne separation assurance for free flight," *Air Traffic Control Quarterly*, Vol. 11, No. 2, 2003, pp. 85–102.

[8] Clari, M. S. V., Ruigrok, R. C., Hoekstra, J. M., and Visser, H. G., "Cost-benefit study of free flight with airborne separation assurance," *Air Traffic Control Quarterly*, Vol. 9, No. 4, 2001, pp. 287–309.

[9] Tomlin, C., Pappas, G. J., and Sastry, S., "Conflict resolution for air traffic management: A study in multiagent hybrid systems," *IEEE Transactions on automatic control*, Vol. 43, No. 4, 1998, pp. 509–521.

[10] Kahne, S., and Frolow, I., "Air traffic management: Evolution with technology," *IEEE Control Systems*, Vol. 16, No. 4, 1996, pp. 12–21.

[11] Harman, W. H., "TCAS- A system for preventing midair collisions," *The Lincoln Laboratory Journal*, Vol. 2, No. 3, 1989, pp. 437–457.

[12] Frazzoli, E., Mao, Z.-H., Oh, J.-H., and Feron, E., "Resolution of conflicts involving many aircraft via semidefinite programming," *Journal of Guidance, Control, and Dynamics*, Vol. 24, No. 1, 2001, pp. 79–86.

[13] Raghunathan, A. U., Gopal, V., Subramanian, D., Biegler, L. T., and Samad, T., "Dynamic optimization strategies for three-dimensional conflict resolution of multiple aircraft," *Journal of guidance, control, and dynamics*, Vol. 27, No. 4, 2004, pp. 586–594.

[14] Enright, P. J., and Conway, B. A., "Discrete approximations to optimal trajectories using direct transcription and nonlinear programming," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 4, 1992, pp. 994–1002.

[15] Schouwenaars, T., De Moor, B., Feron, E., and How, J., "Mixed integer programming for multi-vehicle path planning," *Control Conference (ECC), 2001 European*, IEEE, 2001, pp. 2603–2608.

[16] Richards, A., and How, J. P., "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," *American Control Conference, 2002. Proceedings of the 2002*, Vol. 3, IEEE, 2002, pp. 1936–1941.

[17] Pallottino, L., Feron, E. M., and Bicchi, A., "Conflict resolution problems for air traffic management systems solved with mixed integer programming," *IEEE transactions on intelligent transportation systems*, Vol. 3, No. 1, 2002, pp. 3–11.

[18] Vela, A., Solak, S., Singhose, W., and Clarke, J.-P., "A mixed integer program for flight-level assignment and speed control for conflict resolution," *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, IEEE, 2009, pp. 5219–5226.

[19] Mellinger, D., Kushleyev, A., and Kumar, V., "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, 2012, pp. 477–483.

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

[21] Morgan, D., Chung, S.-J., and Hadaegh, F. Y., "Model predictive control of swarms of spacecraft using sequential convex programming," *Journal of Guidance, Control, and Dynamics*, Vol. 37, No. 6, 2014, pp. 1725–1740.

[22] Acikmese, B., and Ploen, S. R., "Convex programming approach to powered descent guidance for mars landing," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 5, 2007, pp. 1353–1366.

[23] Delahaye, D., Peyronne, C., Mongeau, M., and Puechmorel, S., "Aircraft conflict resolution by genetic algorithm and B-spline approximation," *EIWAC 2010, 2nd ENRI International Workshop on ATM/CNS*, 2010, pp. 71–78.

[24] Cobano, J. A., Conde, R., Alejo, D., and Ollero, A., "Path planning based on genetic algorithms and the monte-carlo method to avoid aerial vehicle collisions under uncertainties," *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 4429–4434.

[25] Pontani, M., and Conway, B. A., “Particle swarm optimization applied to space trajectories,” *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 5, 2010, pp. 1429–1441.

[26] Pallottino, L., Scordio, V. G., Frazzoli, E., and Bicchi, A., “Probabilistic verification of a decentralized policy for conflict resolution in multi-agent systems,” *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, IEEE, 2006, pp. 2448–2453.

[27] Wollkind, S., Valasek, J., and Ioerger, T., “Automated conflict resolution for air traffic management using cooperative multiagent negotiation,” *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004, p. 4992.

[28] Purwin, O., D’Andrea, R., and Lee, J.-W., “Theory and implementation of path planning by negotiation for decentralized agents,” *Robotics and Autonomous Systems*, Vol. 56, No. 5, 2008, pp. 422–436.

[29] Desaraju, V. R., and How, J. P., “Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees,” *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 4956–4961.

[30] Inalhan, G., Stipanovic, D. M., and Tomlin, C. J., “Decentralized optimization, with application to multiple aircraft coordination,” *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, Vol. 1, IEEE, 2002, pp. 1147–1155.

[31] Schouwenaars, T., How, J., and Feron, E., “Decentralized cooperative trajectory planning of multiple aircraft with hard safety guarantees,” *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004, p. 5141.

[32] Richards, A., and How, J., “Decentralized model predictive control of cooperating UAVs,” *43rd IEEE Conference on Decision and Control*, Vol. 4, Citeseer, 2004, pp. 4286–4291.

[33] Hoffmann, G., Rajnarayan, D. G., Waslander, S. L., Dostal, D., Jang, J. S., and Tomlin, C. J., “The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC),” *The 23rd Digital Avionics Systems Conference (IEEE Cat. No. 04CH37576)*, Vol. 2, IEEE, 2004, pp. 12–E.

[34] Howlet, J. K., Schulein, G., and Mansur, M. H., “A practical approach to obstacle field route planning for unmanned rotorcraft,” 2004.

[35] Kavraki, L., Svestka, P., and Overmars, M. H., *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, Vol. 1994, Unknown Publisher, 1994.

[36] LaValle, S. M., “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[37] 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.

[38] Naderi, K., Rajamäki, J., and Hämäläinen, P., “RT-RRT*: A real-time path planning algorithm based on RRT,” *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, ACM, 2015, pp. 113–118.

[39] Shim, D. H., and Sastry, S., “An evasive maneuvering algorithm for UAVs in see-and-avoid situations,” *American Control Conference, 2007. ACC’07*, IEEE, 2007, pp. 3886–3891.

[40] Shim, D. H., Kim, H. J., and Sastry, S., “Decentralized nonlinear model predictive control of multiple flying robots,” *Decision and control, 2003. Proceedings. 42nd IEEE conference on*, Vol. 4, IEEE, 2003, pp. 3621–3626.

[41] Sigurd, K., and How, J., “UAV trajectory design using total field collision avoidance,” *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003, p. 5728.

[42] Langelaan, J., and Rock, S., “Towards autonomous UAV flight in forests,” *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2005, p. 5870.

[43] Kahn, G., Zhang, T., Levine, S., and Abbeel, P., “Plato: Policy learning using adaptive trajectory optimization,” *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 3342–3349.

[44] Zhang, T., Kahn, G., Levine, S., and Abbeel, P., “Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search,” *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE, 2016, pp. 528–535.

[45] Ong, H. Y., and Kochenderfer, M. J., “Markov Decision Process-Based Distributed Conflict Resolution for Drone Air Traffic Management,” *Journal of Guidance, Control, and Dynamics*, 2016, pp. 69–80.

[46] Chen, Y. F., Liu, M., Everett, M., and How, J. P., "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 285–292.

[47] Yang, X., and Wei, P., "Autonomous On-Demand Free Flight Operations in Urban Air Mobility using Monte Carlo Tree Search," *International Conference on Research in Air Transportation (ICRAT), Barcelona, Spain*, 2018.

[48] Han, S.-C., Bang, H., and Yoo, C.-S., "Proportional navigation-based collision avoidance for UAVs," *International Journal of Control, Automation and Systems*, Vol. 7, No. 4, 2009, pp. 553–565.

[49] Park, J.-W., Oh, H.-D., and Tahk, M.-J., "UAV collision avoidance based on geometric approach," *SICE Annual Conference, 2008*, IEEE, 2008, pp. 2122–2126.

[50] Krozel, J., Peters, M., and Bilimoria, K., "A decentralized control strategy for distributed air/ground traffic separation," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2000, p. 4062.

[51] Van Den Berg, J., Guy, S. J., Lin, M., and Manocha, D., "Reciprocal n-body collision avoidance," *Robotics research*, Springer, 2011, pp. 3–19.

[52] Muñoz, C., Narkawicz, A., Hagen, G., Upchurch, J., Dutle, A., Consiglio, M., and Chamberlain, J., "DAIDALUS: detect and avoid alerting logic for unmanned systems," 2015.

[53] Kochenderfer, M. J., Chryssanthacopoulos, J. P., Kaelbling, L. P., and Lozano-Pérez, T., "Model-based optimization of airborne collision avoidance logic," Tech. rep., MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, 2010.

[54] Bertram, J. R., Yang, X., Brittain, M., and Wei, P., "Online Flight Planner with Dynamic Obstacles for Urban Air Mobility," *2019 Aviation Technology, Integration, and Operations Conference*, 2019.

[55] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An introduction*, Vol. 1, MIT press Cambridge, 1998.

[56] Chitsaz, H., and LaValle, S. M., "Time-optimal paths for a Dubins airplane," *2007 46th IEEE conference on decision and control*, IEEE, 2007, pp. 2379–2384.

[57] Owen, M., Beard, R. W., and McLain, T. W., "Implementing dubins airplane paths on fixed-wing uavs," *Handbook of Unmanned Aerial Vehicles*, 2015, pp. 1677–1701.

[58] Park, H., Lee, B.-Y., Tahk, M.-J., and Yoo, D.-W., "Differential game based air combat maneuver generation using scoring function matrix," *International Journal of Aeronautical and Space Sciences*, Vol. 17, No. 2, 2016, pp. 204–213.

[59] Huynh, H., Costes, P., and Aumasson, C., "Numerical optimization of air combat maneuvers," *Guidance, Navigation and Control Conference*, 1987, p. 2392.

[60] Rodriguez, E., Morris, C., Belz, J., Chapin, E., Martin, J., Daffer, W., and Hensley, S., "An assessment of the SRTM topographic products," 2005.