JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 40, No. 1, January 2017

Markov Decision Process-Based Distributed Conflict Resolution for Drone Air Traffic Management

Hao Yi Ong* and Mykel J. Kochenderfer†
Stanford University, Stanford, California 94305

DOI: 10.2514/1.G001822

Ensuring safety and providing timely conflict alerts to small unmanned aircraft, commonly known as drones, is important to their integration into civil airspace. This paper proposes a short-term conflict avoidance algorithm for an automated low-altitude, small unmanned aircraft traffic management system. The goal is to balance between aircraft safety and efficiency subject to uncertainty in the environment, aircraft, and pilot response. The proposed algorithm generates advisories for each aircraft to follow, and is based on decomposing a large multiagent Markov decision process and fusing their solutions. As a result, the method scales well and resolves conflicts efficiently. Further, this paper presents a massively distributed architecture used to implement the conflict avoidance algorithm on a practical scale in which simultaneous conflicts can be solved in parallel in tens of milliseconds. Our controller significantly outperforms two baseline algorithms based on uncoordinated and closest-threat heuristics. In terms of conflict probability, our simulations demonstrate that our method is an order of magnitude better than the latter and about 10% better than the former.

I. Introduction

FROM goods delivery to infrastructure surveillance, there are many important applications of small unmanned aircraft systems (UASs). A key enabler of these drone operations is flexible access to the civil airspace [1–3]. To facilitate safe and efficient operations at low altitude, NASA has been leading the exploration of concepts and prototypes for a largely automated UAS traffic management (UTM) system [4]. Although there are many important components in UTM, this paper focuses on automated conflict avoidance because it is critical to ensuring safety.

Designing a robust conflict avoidance system is challenging for various reasons. UTM is intended to handle many more aircraft in a more restrictive and crowded airspace than conventional air traffic management systems. There is uncertainty in the current state because of imperfect sensor measurements, and there is uncertainty in the future paths of the aircraft because of variable pilot response, vehicle performance, wind, and so on. Determining the appropriate trade-off between safety and efficiency is not straightforward, and the system must also coordinate complementary advisories among many aircraft.

There have been a number of important contributions to the topic of conflict avoidance. Kuchar and Yang offer a comprehensive survey of conflict avoidance methods up to the year 2000 [5]. More recent approaches to conflict avoidance problems include mixed-integer programming and sequential convex programming by Schouwenaars et al. [6], Mellinger et al. [7], and Augugliaro et al. [8]. These approaches tend to work well for small vehicle networks. Ong and Gerdes have explored distributed convex optimization techniques to develop an efficient and scalable algorithm called proximal message passing [9]. The limit of these mathematical programming-based approaches is the difficulty with incorporating complex probabilistic models on dynamics, pilot response, and vehicle performance. Other approaches that do not rely on mathematical optimization exploit specific structures in conflict resolution. For instance, Erzberger's algorithm chooses feasible trajectories for commercial aircraft based on a deterministic set of procedures [10,11].

Using a decision theoretic approach that naturally incorporates stochastic models, Airborne Collision Avoidance System X (ACAS X) uses a control policy derived from Markov decision process (MDP) solution techniques and has been shown to improve safety and operational efficiency over existing collision avoidance systems [12,13]. Our algorithm builds upon the concepts underlying ACAS X and introduces an explicit dynamics model of how all aircraft involved in a potential conflict interact under joint, coordinated advisories. Additionally, our algorithm capitalizes on system capabilities such as having large message bandwidths and a centralized system that can receive flight state information from all aircraft [14]. This is different from ACAS X, which is decentralized and has limited communication bandwidth.

The goal of this paper is to develop a robust and efficient method that generates advisories and resolves conflicts between unmanned aircraft. The contributions of this paper are twofold. First, we formulate the conflict resolution problem as a stochastic problem in the form of a large multiagent MDP (MMDP) and develop an efficient way to solve it. The method decomposes the MMDP into computationally tractable subproblems, solves these subproblems, and combines their solutions into a coordinated joint advisory using an iterative search scheme called alternating maximization [15]. Second, we present a distributed architecture used to implement the algorithm that resolves conflicts in real time at scale and demonstrate their effectiveness over two baseline methods.

The rest of the paper is organized as follows. Section II provides a brief introduction to MDPs and a mathematical formulation of the conflict avoidance problem. Sections III and IV develop the decomposition and coordination approach used to solve the problem. Section V demonstrates the effectiveness of the algorithm through numerical experiments, and Sec. VI presents the distributed system design used to implement the algorithm. Concluding remarks are drawn in Sec. VII.

II. Short-Term Conflict Avoidance

We begin with a brief overview of MDPs and the problem definition.

A. Markov Decision Process

Since the 1950s, MDPs have been well studied and applied to a diverse set of problems [16–19]. In an MDP, an agent chooses action

Received 28 April 2016; revision received 11 July 2016; accepted for publication 15 July 2016; published online 10 October 2016.

*Graduate Research Assistant, Department of Mechanical Engineering; haoyi@stanford.edu. Student Member AIAA.

†Assistant Professor, Department of Aeronautics and Astronautics; mykel@stanford.edu. Senior Member AIAA.

a_t at time t after observing state s_t. The agent then receives reward r_t, and the state evolves probabilistically based on the current state–action pair. The explicit assumption that the next state only depends probabilistically on the current state–action pair is referred to as the Markov assumption. An MDP can be defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ and $\mathcal{A}$ are the sets of all possible states and actions, respectively, T is a probabilistic transition function, and R is a reward function. The probability of transitioning into state s' after taking action a from state s is denoted as $T(s, a, s')$. The immediate reward received for taking action a from state s is denoted as $R(s, a)$. The goal of solving MDPs is to find a policy that maximizes the expected accumulation of rewards for an agent.

Although solving MDPs is an effective method for determining actions for a single agent in stochastic environments, they can also be extended to cooperative multiagent domains. MMDPs extend MDPs and allow for sequential decision making in a cooperative multiagent system, and is similar to the case in which a centralized planner has access to the system state. Similar to an MDP, an MMDP can be defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R)$. The difference is that $\mathcal{S}$ and $\mathcal{A}$ are now the sets of all possible joint states and actions, respectively, and that T and R operate on elements from these sets.

To solve any MDP (or MMDP), we compute a policy $\pi^\star$ that, if followed, maximizes the expected discounted sum of immediate rewards from any given state. In general, it is not clear how to best represent the policy. If the state and action spaces are finite sets, the mapping from states to actions can be represented using a table. For problems with continuous state variables, as is the case with conflict avoidance, it is impossible to enumerate all possible state–action mappings. Several approximate dynamic programming techniques have been proposed to address this limitation [20], and we represent $\pi^\star$ with a grid-based discretization of $\mathcal{S}$ and $\mathcal{A}$.

The optimal policy can be defined in terms of the optimal state–action utility function $U^\star(s, a)$, which is the expected sum of rewards when starting in state s, taking action a, and then following actions dictated by $\pi^\star$. In the literature, the utility function is also called the value function. Mathematically, it obeys the Bellman recursion

$$U^\star(s, a) = R(s, a) + \sum_{s' \in \mathcal{S}} T(s, a, s') U^\star(s') \tag{1}$$

where we define $U^\star(s) = \max_{a \in \mathcal{A}} U^\star(s, a)$. The state–action utility function can be computed using a dynamic programming algorithm called value iteration. Beginning with an initial guess, this technique iteratively updates Eq. (1) until the estimate converges. The optimal policy, in turn, is given by

$$\pi^\star(s) = \operatorname*{argmax}_{a \in \mathcal{A}} U^\star(s, a) \tag{2}$$

Because the number of states scales exponentially with the number of agents, exact solutions may be computationally intractable for problems with more than just a few state variables or agents [18]. This issue is referred to as the curse of dimensionality [21]. Our formulation exploits the unique structure of our problem to provide a tractable, approximate solution.

B. Mathematical Formulation

Our formulation focuses on horizontal conflict resolution with the aircraft assumed to be co-altitude. Here, we have n aircraft at risk of conflict between each other, and conflict is defined as the loss of minimum separation. Loss of minimum separation, in turn, is defined to be when two aircraft come closer than some threshold distance.

1. Resolution Advisories

At each time step, the system issues a joint advisory out of a finite set of bank angles corresponding to each aircraft's action in the multithreat scenario. The joint advisory can be written as $\phi = (\phi_1, \ldots, \phi_n)$. Although the time period for a single transition may be defined according to system needs, we define it to be 5 s. This period corresponds to the decision frequency of our system. These bank angles are directly related to the turn radii of the aircraft through Dubin's kinematic equations [22]. These joint advisories constitute the action set $\mathcal{A} = \{-20°, -10°, 0°, 10°, 20°, \text{COC}\}^n$. Here, positive angles correspond to left banks and negative angles to right banks. COC is a special clear-of-conflict status that allows the aircraft to continue its trajectory as usual. Conversely, all other actions correspond to corrective advisories to be issued during conflict. Although the resolution of the decision period and bank angles can be increased, shorter decision periods and finer bank angle discretization increase the problem complexity. Our simulation results also suggest that they worked well for our problem (see Sec. V).

It is natural to consider extending the set of resolution advisories to 1) vertical resolution, 2) speed resolution, and 3) combination resolutions (e.g., turn left and speed up simultaneously). In fact, vertical resolutions have been considered in ACAS X [12,19]. However, UTM limits its focus to a narrow altitude band, with one proposal restricting altitudes to between 200 and 500 ft. [23]. Although it is possible to incorporate speed resolution and combination resolutions into our model, considering all combinations of various speeds and bank angles would cause the solution search space to grow quickly. Nevertheless, modeling these maneuvers would be simple in terms of extending the advisory set and the dynamic model to account for them.

2. Dynamic Model

The system is described by the set of state variables of each aircraft. The ith plane's state is $s_i = (x_i, y_i, \psi_i, v_i, p_i)$, where x_i and y_i indicate the planar coordinates mapped from latitude and longitude, ψ_i indicates the heading, v_i indicates the aircraft groundspeed, and the Boolean variable p_i indicates the pilot responsiveness. In our formulation,

$$p_i = \begin{cases} 1, & \text{pilot is responsive} \\ 0, & \text{pilot is nonresponsive} \end{cases} \tag{3}$$

Responsiveness is defined as whether a pilot is currently maneuvering in compliance with the given resolution advisory, if one exists.

To account for uncertainty in the environment, aircraft dynamics, and pilot response, we model the aircraft banking response as Gaussian. We further assume that the aircraft dynamics is fast relative to the decision period of 5 s and thus allow the bank angle to change immediately when the pilot executes the advisory. This assumption is based on the observation that small UAS typically have extremely high power-to-weight ratios and accelerations can easily reach multiple Gs [24].

When an aircraft is not following a resolution advisory or nonresponsive, be it remotely or autonomously piloted, it follows a white-noise acceleration and banking model. When responsive, the pilot executes the advisory for the full decision period. In the model, the advisory response in the current time step of T is determined stochastically based on the new advisory using a Bernoulli process. As such, the delay until response follows a geometric distribution. For some mean time until response k, the response probability at each step in the process is $T/(T + k)$. Specifically,

1) the pilot always responds to a clear of conflict status "advisory";

2) when the pilot is not responding, the aircraft follows a white noise model;

3) once the pilot responds, it will continue to respond for the duration of the advisory; and

4) the average response delay for initial advisories is $k = 5$ s, which gives a response probability of 0.5 at each step (from ICAO-recommended practices for responding to resolution advisories [25]).

When the aircraft is issued an advisory and its pilot is responsive, the bank angle distribution is centered on the resolution advisory with a standard deviation of 4°. Otherwise, the aircraft banks randomly based on a zero-mean Gaussian with a standard deviation of 10°. The aircraft groundspeed is held constant between decision stages, and is modeled as a Gaussian distribution centered on the aircraft's reported speed with a standard deviation of 2 m/s. This simplification is

because of the observation that aircraft usually respond slowly to sudden speed change inputs for short durations. The variation in aircraft speed captures random effects like navigation errors, wind gusts, and other trajectory perturbations.

As mentioned, the formulation uses Dubin's kinematic model to compute state transitions:

$$\dot{x} = v\cos\psi, \qquad \dot{y} = v\sin\psi, \qquad \dot{\psi} = \frac{g\tan\phi}{v} \tag{4}$$

where v is the speed and g is the gravitational acceleration. The relative simplicity of this dynamic model reduces the computation time required to solve the problem and the risk of overfitting a more complicated model.

3. Reward Function

The focus of our conflict avoidance system is to balance the dual objectives of maintaining safety while minimizing disruption. These objectives are captured in a reward function composed of a sum of different components. For safety, a conflict is defined when aircraft come within a minimum separation distance $r^{\min} = 500$ m in the horizontal plane. The horizontal separation definition of 500 m is obtained by scaling the ICAO standard of 5 nmi defined for traditional aircraft down to a reasonable distance for small drones [26,27]. At any time t, the separation between two aircraft is denoted $r(t)$. The minimum separation between the two aircraft during each decision period T starting at time t is

$$r_t^{\text{sep}} = \min\{r(\tau)|\tau \in [t, t+T)\} \tag{5}$$

The reward function penalizes any decision that brings any pair of aircraft to $r_t^{\text{sep}} < r^{\min}$ by a constant value. The function further penalizes the minimum separation of any two aircraft by an exponentially decaying graph centered at 0 m to maximize separation even in cases in which conflict is unavoidable. To limit disruption, advisories are penalized. The COC action corresponds to a 0° bank angle during reward computation. The reward function for a decision period starting at time t is written as

$$R_t(s,\phi) = -\lambda_1 I_{\text{sep}}(s,\phi) - \lambda_2 \exp(-r_t^{\text{sep}}(s,\phi)/r^{\min}) \tag{6}$$

$$-\lambda_3\|\phi\|_2^2 - \lambda_4 \mathbf{1}^T I_{\text{coc}}(\phi) \tag{7}$$

where I_{sep} and I_{coc} are the indicator functions:

$$I_{\text{sep}}(s,\phi) = \begin{cases} 1, & r_t^{\text{sep}}(s,\phi) < r^{\min} \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

$$(I_{\text{coc}}(\phi))_i = \begin{cases} 1, & \phi_i \neq \text{COC} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

For this algorithm, the weight parameters are $\lambda_1 = 1000$, $\lambda_2 = 10$, and $\lambda_3 = 0.02$. We balance between corrective resolution advisory alerts and safety (probability of conflict) by tuning λ_4, and we set $\lambda_4 = 10$ unless otherwise stated. Whereas this reward function captures only basic preferences, a better function can be found through expert preference elicitation methods [28,29].

Although we consider only distance-based separation in our work, other types of separation parameters can be accommodated using penalty functions as done above. For instance, time-based parameters could be incorporated by extrapolating the current aircraft trajectory and computing how long it will take to breach a distance separation constraint, if a breach can occur. The reward function will then assign a negative value to smaller separation times. Future work could consider the merits of various separation parameters, both quantitatively in terms of conflict probability and qualitatively in terms of trajectory shapes and timeliness of advisory alerts.

III. MMDP Decomposition

As formulated in Sec. II, computing the optimal solution is impractical for more than a few aircraft. We therefore decompose the problem into pairwise encounter MDPs and fuse the state–action utilities together [13,30]. The source code that implements these models and algorithms can be found at https://github.com/sisl/ConflictAvoidanceDASC [31].

A. Pairwise Conflict Avoidance

Instead of tackling the problem as a full MMDP, our approach first decomposes it into a set of

$$\binom{n}{2} = \frac{1}{2}n(n-1)$$

pairwise encounters.

1. Resolution Advisories

As in the original MMDP formulation, the system can issue one advisory out of a finite set of bank angles corresponding to turn radii for each aircraft at each time step. These joint advisories constitute the action set $\mathcal{A} = \{-20°, -10°, 0°, 10°, 20°, \text{COC}\}^2$, which follows the definition used in the MMDP.

2. Dynamic Model

One may use the multiagent formulation presented in Sec. II with only two aircraft. But a better approach would be to replace the absolute state variables of each aircraft with a single set of relative state variables. The use of relative state variables reduces the cardinality of the state space and thus computational complexity. The speeds, however, are still absolute with respect to the world.

The pilot response and dynamic models of the aircraft is the same as in the original MMDP. Because we use relative states, however, appropriate modifications to the equations of motion are required to take rotating reference frames into account. To compute the relative state transitions, Eq. (4) is used to update the absolute aircraft states from the initial configuration. We compute the intermediate states $x^{\text{abs}} = x_j - x_i$ and $y^{\text{abs}} = y_j - y_i$. The relative states in $s = (x^{\text{rel}}, y^{\text{rel}}, \psi^{\text{rel}}, v^{\text{ownship}}, v^{\text{intruder}}, p^{\text{ownship}}, p^{\text{intruder}})$ for aircraft j with respect to i can then be computed as follows:

$$\psi^{\text{rel}} = \psi_j - \psi_i \tag{10}$$

$$x^{\text{rel}} = x^{\text{abs}}\cos\psi^{\text{rel}} + y^{\text{abs}}\sin\psi^{\text{rel}} \tag{11}$$

$$y^{\text{rel}} = -x^{\text{abs}}\sin\psi^{\text{rel}} + y^{\text{abs}}\cos\psi^{\text{rel}} \tag{12}$$

3. Reward Function

The reward function for the pairwise encounter is the same as the MMDP formulation for two agents.

B. Offline Solution

1. Discretization Scheme

The optimal policy specifies the target bank angle to reach and hold for the 5 s time step between decisions. States are mapped from the continuous to the discrete domain using multilinear interpolation, and transition probabilities are estimated using sigma-point sampling [32]. We use a set of weighted values for bank angle ϕ and speeds v_1 and v_2 for sigma-point sampling. For the transition between states, the system has a $1/3$ probability of following the nominal bank angle and speeds. The remaining $2/3$ weight is uniformly distributed across every other combination of $(\sigma_\phi, \sigma_{v_1}, \sigma_{v_2})$, where $\sigma_\phi \in \{-4°, 0°, 4°\}$ for a corrective resolution advisory, $\sigma_\phi \in \{-10°, 0°, 10°\}$ for a clear-of-conflict status, and $\sigma_{v_{1|2}} \in \{-2\text{ m/s}, 0\text{ m/s}, 2\text{ m/s}\}$. These values come from the Gaussian distributions over the bank angle response and aircraft speed in Sec. II.

Table 1 State space discretization scheme

Variable	Minimum	Maximum	No. of values
x^{rel}, y^{rel}	−3000 m	3000 m	51
ψ^{rel}	0°	360°	37
v^{ownship}, v^{intruder}	10 m/s	20 m/s	5
p^{ownship}, p^{intruder}	— —	— —	2

These techniques were effective in providing a discrete approximation to an MDP with a continuous state space in [12,13]. The state space for the relative variable formulation is uniformly discretized using a multidimensional grid. Table 1 shows the result of the discretization, which has approximately 9.6 million discrete states: 51×51 grid for 37 discrete relative headings, with 5×5 different speed combinations for the aircraft pair and 2×2 possible pilot response state combinations. Even though the discretization can be made finer at the expense of additional computation and storage, this level of discretization worked well in numerical experiments.

2. *Implementation*

Dynamic programming is used to compute the utility function $U^\star$. This technique is called value iteration, and solves the discretized problem optimally. The basic idea of this algorithm is to iteratively solve the Bellman recursion (1) from an initial guess. One modification is that the original transition function is replaced by $\tilde{T}$, the approximate transition function resulting from sigma-point sampling and multilinear interpolation. In the case of partial state observability, a heuristic called QMDP can be used to produce a solution that performs well in many real-world scenarios [33,34].

The solver was implemented in Julia, a high-level dynamic programming language for technical computing [35]. The solution was generated using twenty 2.30 GHz Intel Xeon processor cores with 125 GB RAM. The procedure took approximately 7 h and 60 iterations to complete, with most of the work done in computing state transition probabilities. After this offline computation, the policy is consulted during operation in the form of a $U^\star(s, a)$ look-up table to select the best advisory to issue at each time step.

Despite the expensive one-off computation, the policy computation at runtime is extremely efficient. With a 64-bit floating point representation for $U^\star(s, a)$ values, the lookup table would take up 77 megabytes, which would fit comfortably in a random-access memory device of a modern compute cluster. The computation to extract the best action from the table using Eq. (2) would thus only take microseconds.

3. *Policy Visualization*

Figure 1 shows the policy for two encounters with the ownship aircraft state $(x, y, \psi, v) = (0\text{ m}, 0\text{ m}, 0°, 10\text{ m/s})$. Both aircraft are responsive. In Fig. 1a, the intruder is traveling toward the ownship aircraft with 180° relative heading at 10 m/s, and in Fig. 1b, the intruder is traveling toward the ownship aircraft with 120° relative heading at 10 m/s. The figures show the action that would be issued for both aircraft with the ownship's position fixed at (0, 0) and the intruder's coordinates are varied over (x, y) with the indicated heading. Positive and negative values on the color squares legend on top of the figures indicate left and right banks, respectively. The clear-of-conflict status actions are colored pure white. Note that the negative bank angle with the largest magnitude is pale gray — distinct from the clear-of-conflict status actions.

The heat map visualization suggests that the solution matches intuition. For instance, consider the scenario in which the intruder aircraft is at $(x, y) = (1000\text{ m}, 500\text{ m})$ heading straight toward the ownship aircraft. Here, Fig. 1a shows that the optimal joint policy is for the ownship and intruder aircraft to both bank right. This combination of actions ensures that both aircraft turn away from one another. It is also important to note that the algorithm issues advisories between a set decision time period from a fixed set of advisories, which results in some surprising behavior. For instance, an interesting feature is the slight "indent" in the left inset of Fig. 1a at roughly $(x, y) = (1500\text{ m}, 0\text{ m})$, which indicates that the ownship aircraft should wait to determine which way the intruder might turn before responding with a bank action. In Fig. 1b the encounter scenario is identical except the intruder is flying at a relative heading of 120°. The nonzero banking region is rotated to prevent the intruder from flying into the exclusion region of the ownship aircraft from the intruder's left. For both heading angles, once both aircraft experience loss of minimum separation at less than 500 m the system immediately advises both aircraft to bank more aggressively to resolve the conflict quickly.

Figure 2 shows the policy for three encounters with the same setup as in Fig. 1a, except with different ownship and intruder pilot responsiveness. Again, the solution matches intuition: whenever a pilot is in potential conflict and nonresponsive, the solution suggests a more aggressive bank action much earlier to compensate for the potential response delay. We observe this in Fig. 2a. Here, the intruder is alerted with the most aggressive maneuvers at more than 1700 m separation, whereas the ownship is only alerted starting at roughly 1200 m separation. The other plots reveal the same behavior.

IV. Multiagent Coordination

To coordinate multiple aircraft in conflict, this section uses MMDP decomposition and utility fusion to produce an approximate utility function for the global MMDP. As the search space for the approximate utility function can still be large (on the order of $|\{-20°, -10°, 0°, 10°, 20°, \text{COC}\}|^n = 6^n$ for n aircraft), we derive an algorithm that solves the traffic management problem using an iterative search heuristic.

A. Utility Fusion

To obtain a global MMDP solution from the pairwise subproblems, we need a measure of how good each individual pairwise solutions are when combined with other pairwise encounters. Utility fusion combines state–action values from subproblems to obtain a proxy to the full problem's state–action value function [12,13,30]. It computes the utility $U^\star_{ij}$ for the pairwise encounter between two different aircraft i and j, assuming that the optimal policy for that encounter is followed in the future. For a set of pairwise encounter MDP states and actions

$$s = (s_1, \ldots, s_n), \qquad a = (a_1, \ldots, a_n)$$

the best action for each aircraft can be found using a proxy global utility function $U^\star$ that results from fusion function f:

$$U^\star(s, a) = f(\{U^\star_{ij}(s_i, s_j, a_i, a_j) | i, j \in \{1, \ldots, n\}, i < j\}) \quad (13)$$

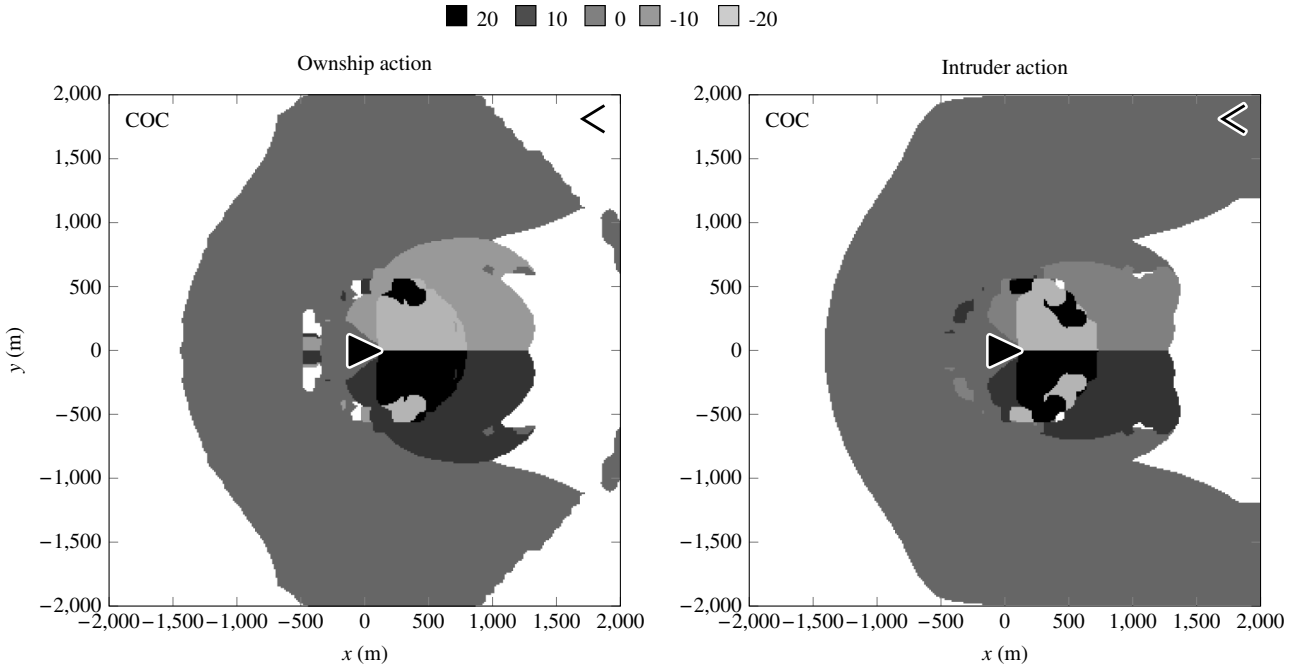
Our algorithm employs two fusion strategies. The first strategy defines f to be a summation:

$$f(\{U^\star_{ij}\}) = \sum U^\star_{ij} \quad (14)$$

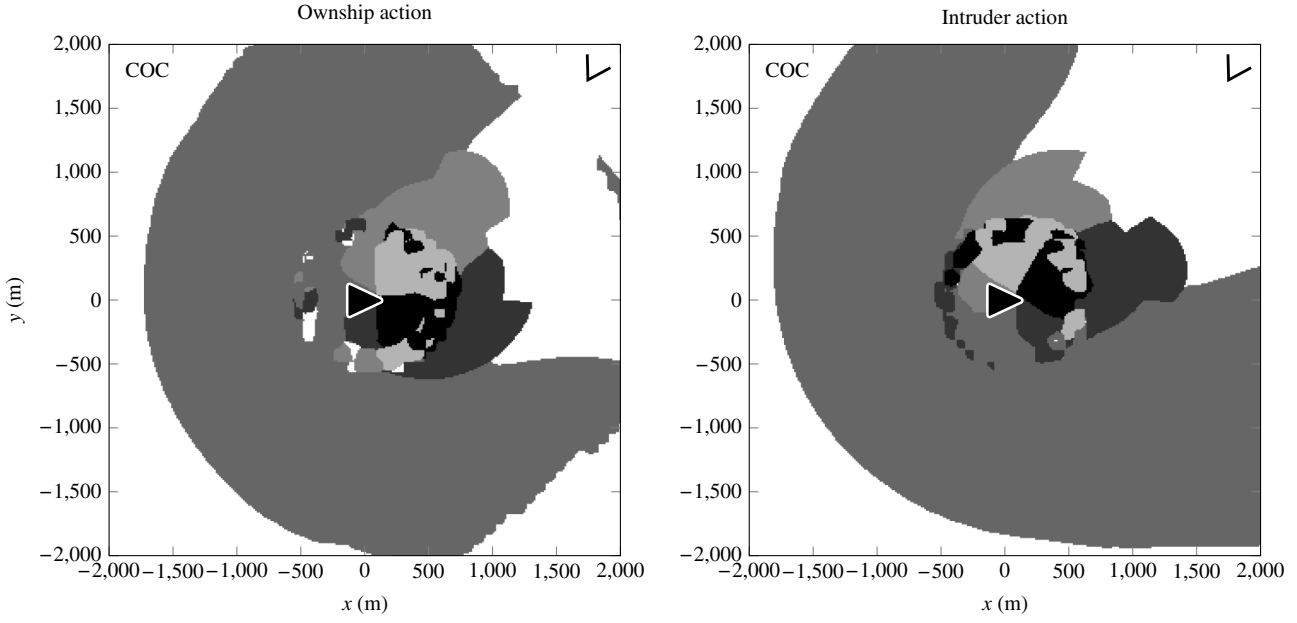
which is also called the max-sum strategy ("max" here refers to us taking the argmax of the joint action space). Intuitively, this is an averaging strategy. Defining f in this fashion leads to counting nonzero bank angle costs multiple times, which would be reflected in the state–action utilities for each pairwise encounter. Because the system incurs the action cost multiple times when in reality and the system can alert only once at each time step, the system is encouraged to delay issuing nonzero bank angles. This preference for later banking may be undesirable as it limits the amount of options available to each aircraft the closer they are to loss of minimum separation and can lead to more aggressive maneuvers at later times.

The second strategy defines f to be the minimum state–action utility over all pairwise encounters:

$$f(\{U^\star_{ij}\}) = \min U^\star_{ij} \quad (15)$$



a) 180° head-on intruder approach.



b) 120° angled intruder approach.

Fig. 1 Pairwise encounter MDP policy for responsive pilots.

This method is also referred to as the max-min strategy. Intuitively, this is a worst-case strategy. As opposed to the max-sum strategy, the max-min strategy avoids accumulating the cost of alerting for each pairwise encounter. This method also tends to provide more conservative policies in the sense that earlier banking is preferred.

To extract the joint policy, we compute

$$a^\star = \underset{a \in \mathcal{A}^n}{\operatorname{argmax}} U^\star(s, a) \tag{16}$$

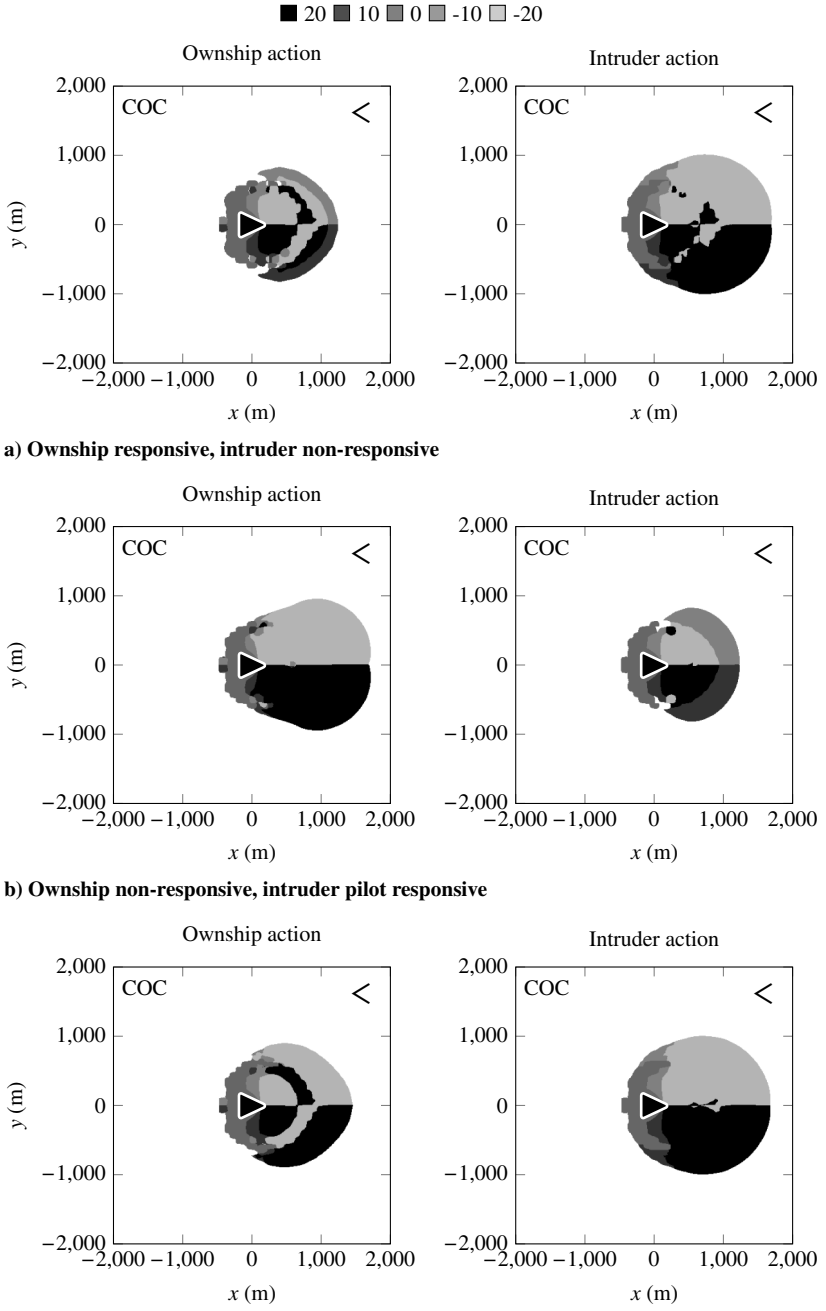
with a search heuristic that produces locally optimal solutions, outlined in Sec. IV.B.

B. Search Heuristic

As mentioned, the search space for joint policies for the approximate global utility function is on the order of $|\mathcal{A}|^n$ and can thus be large for large numbers of aircraft n. To alleviate this issue, our technique finds a solution in which each agent's policy is the best response to the policies employed by all other agents. The method relies on alternating maximization, which computes a policy for an agent that maximizes the joint reward while fixing the policies of other agents. The procedure is repeated until the joint policy converges to a local optimum, and is outlined in Algorithm 1. Note that the joint aircraft state is denoted s and the utility fusion strategy f.

C. Coordination Algorithm

In UTM, aircraft information is updated frequently to a centralized system. To avoid overestimating system specifications, we assume that the updates are simple and consist only of a unique aircraft identifier, latitude and longitude, heading, and speed. These values can be directly mapped to our large MMDP and pairwise encounter relative states.



a) Ownship responsive, intruder non-responsive

b) Ownship non-responsive, intruder pilot responsive

c) Both pilots non-responsive

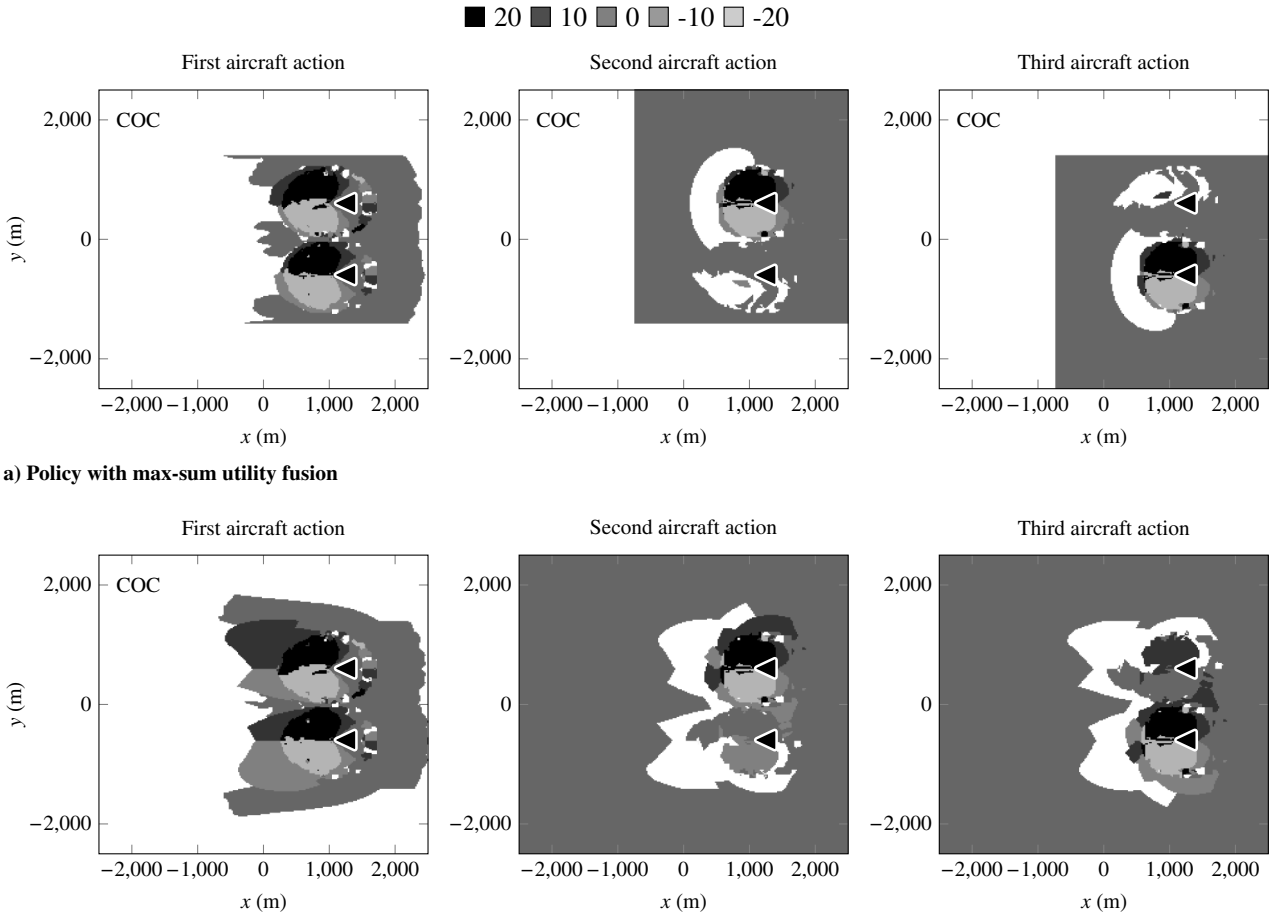
Fig. 2 Pairwise encounter MDP policy for pilots with varying responsiveness.

To incorporate the search heuristic into a serial coordination scheme, each aircraft first broadcasts its message to the system. The system then computes the locally optimal joint policy Algorithm 1, and sends resolution advisories to each aircraft individually. Figure 3 visualizes the joint policies computed with the max-sum and max-min utility fusion methods for a three-aircraft encounter scenario using a heat map. All aircraft are responsive. The first aircraft is flying straight with a heading of 0°, and the second and third aircraft are flying straight with headings of 180°. All aircraft are flying at 10 m/s. With the second and third aircraft coordinates fixed at (1200 m, 600 m) and (1200 m, −600 m), respectively, the first aircraft's coordinates are varied over $x, y \in [-2000\text{ m}, 2000\text{ m}]$. The left, center, and right insets plot the suggested action for the first, second, and third aircraft, respectively, when the first aircraft's coordinates are varied across the plot. The color at any coordinate indicates suggested bank angles for the case in which the first aircraft is at the coordinates flying with a 0° angle heading.

As explained in this section, max-sum utility fusion counts nonzero bank angle costs multiple times. This multiple counting thus produces the turn suggestion regions much smaller than the one generated by the max-min method. The "COC" status regions outside of the 4 km "fly-straight" zone on the max-sum plots are because of the limited range in the pairwise solution. An interesting feature from

Algorithm 1 Search Heuristic

procedure SEARCH (s, f)
  initialize: $a^\star$ with COC status for all aircraft
  while $a^\star$ not converged and not timeout **do**
    for each aircraft i **do**
      for each $a^{\text{new}} \in \mathcal{A}$ **do**
        $a \leftarrow a^\star$
        $a_i \leftarrow a^{\text{new}}$
        if $f(s, a) > f(s, a^\star)$ **then** $a^\star = a$
  output: $a^\star$



a) Policy with max-sum utility fusion

b) Policy with max-min utility fusion

Fig. 3 Triple aircraft encounter policy.

the max-min policy is the "patches" in the center and right insets. In the center inset, the light gray patch at around $y = -1000$ m tells the second aircraft at $y = 600$ m to bank right if the first aircraft is in that patch. Likewise, the dark gray patch in the right inset tells the third aircraft at $y = -600$ m to bank left if the first aircraft is in that patch. Intuitively, these patches indicate that both the second and third aircraft should synchronize their banking to avoid the first aircraft. One important property of the decomposition methods is that they do not begin advising nonzero bank angles (thus deviating aircraft from their intended trajectory) any earlier than the pairwise encounter policy. This behavior can be verified mathematically from the utility fusion definitions.

V. Numerical Experiments

The speed and scaling of our algorithm is demonstrated with a range of examples randomly generated from an encounter model. These examples are intentionally kept simple for illustrative purposes, but can easily be extended with more refined networks and aircraft models. This section presents the encounter model and baselines against which our method is benchmarked against and concludes with a discussion of simulation results.

A. Encounter Model

The coordination algorithm was evaluated against a stress-test set of multithreat encounters randomly generated from an encounter model. As described in the previous section, the positions and velocities of each aircraft were available to the centralized UTM system. Although encounters between more than three aircraft are very rare for the conventional transport aircraft, a large part of current commercial interest surrounding unmanned aircraft stems from the potential to use a large number of surveillance and transport drones.

Taking into account the above consideration, the multithreat encounters ranged from 2 to 10 aircraft. The positions were initialized uniformly randomly in an annulus such that they were not initially in conflict. Specifically, the annulus had inner and outer radii of 2000 and 3000 m, and if a new aircraft added is closer to some other aircraft than 600 m, we resample the new aircraft position to avoid initializing aircraft already in conflict. The speeds of the aircraft were uniformly randomly initialized between 10 and 20 m/s. The headings were initialized to always point straight toward the annulus center to ensure that every encounter would have potential conflicts. In the future, we could develop a more sophisticated encounter model from, for example, recorded radar data that is statistically representative of encounters between unmanned aircraft. At present, however, such data are not yet available at the level of what exists for commercial aircraft [36].

B. Aircraft Model

An ODE solver was used for Dubin's kinematic equations, with Gaussian noise in acceleration and banking. The aircraft maps the resolution advisories to PID control policies tuned to reach the target bank angle:

$$\ddot{\phi} = -2\omega_n\dot{\phi} + \omega_n^2(\phi^{\text{tgt}} - \phi) \tag{17}$$

where ω_n is a constant related to the amount of control power available and ϕ^{tgt} is the target bank angle defined in the joint policy. In simulation, $\omega_n = 0.2$.

To model state uncertainty because of measurement error from, for example, GPS inaccuracy, the simulations model the heading and speed error as zero-mean Gaussians with 2° and 1 m/s standard deviations, respectively. The aircraft longitude and latitude information are subject to zero-mean Gaussian noise with 50 m

standard deviation. To account for bank angle command errors, the inputs are subject to a zero-mean Gaussian noise with 2° standard deviation in simulation.

When the state is partially observable, a probability distribution over the state space can be inferred from a recursive Bayesian estimation from a sequence of state measurements [37]. This distribution is also called a belief state. In our algorithm, separate belief states are maintained for each aircraft. To extract the policy from a belief state instead of an exact state, we use the pairwise MDP solution $U^\star$ as follows:

$$\pi^\star(b_i) = \underset{a \in \mathcal{A}}{\operatorname{argmin}} \sum_{s_i} b_i(s_i) U^\star(s_i, a) \tag{18}$$

where b_i is the belief distribution for the ith pairwise encounter and $b_i(s_i)$ is the probability of being in state s_i based on the belief distribution. The pilot response indicator variable is deterministic. The initial response is always nonresponsive, and whether it switches to a responsive state is given by a response that the aircraft sends when it acknowledges receipt of the advisory. Future work could incorporate a probabilistic belief model for the pilot response based on the aircraft's flight history, similar to that used in [38].

C. Baseline Methods

Our coordination algorithm was tested against a simple command arbitration heuristic and uncoordinated algorithm that also uses utility fusion. The latter baseline was designed to demonstrate how well coordinated aircraft do against uncoordinated ones.

1. *Closest Threat Command Arbitration*

In the first of our baseline algorithms, we consider the closest threat command arbitration method. This method separates an n-aircraft multithreat scenario into n pairwise encounters for each of the n aircraft. Each aircraft then executes the action suggested by the solution to pairwise encounters with the lowest separation distance. Because the closest intruder is often the most immediate threat, prioritizing this pairwise encounter seems sensible.

2. *Uncoordinated with Utility Fusion*

In the second baseline, each aircraft is able to receive the basic state information for all other aircraft from a central system, but is unable to tell what other aircraft might do. Each aircraft then assumes that all other aircraft are white noise intruders that travel roughly along their initial headings. This setup is similar to the one in ACAS X [12,13]. The action policy is extracted using utility fusion greedily with fixed COC status advisories for all other aircraft.

D. Results

The experiments consist of 1,080,000 simulations with encounters ranging from 2 to 10 aircraft running on a 3.40 GHz Intel i7 processor with 32 GB RAM. This section discusses the results.

1. *Safety Versus Alert Rate Trade-Off*

Our approach involves first balancing safety and alert rate for the max-min and max-sum utility fusion methods individually. We compare only the two's performance after tuning this balance. To trade-off between safety (conflict probability) and alert rate, we vary the weight λ_4 on the reward component for the clear-of-conflict indicator function in Eq. (6). Specifically, we varied λ_4 on a uniform logarithmic scale from 10^{-3} to 10^2. The trade-off plot is shown in Fig. 4, in which the alert rate is defined to be the average number of alerts per aircraft per encounter simulation, and the conflict probability is defined to be the same as above. We plot only the curve generated by the algorithm that uses the max-min utility fusion method, because the results for the max-sum method show a similar trend, only with different numbers because of the differences in the optimization problem characteristics. The best performing points are at the bottom-left "knee" of the Pareto frontier, minimizing both

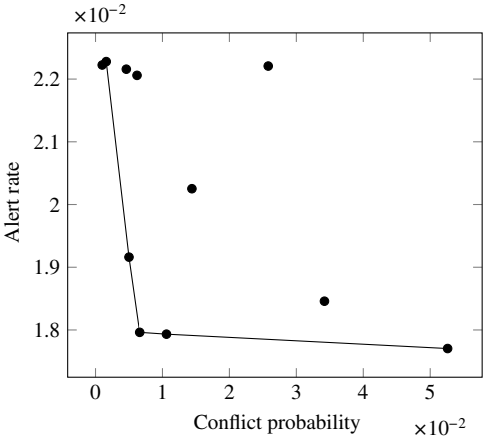


Fig. 4 Trade-off plot.

conflict probability and alert rate, and correspond to $\lambda_4 \in [1, 10]$. We use these points to evaluate the safety performance.

2. *Safety Performance*

Figure 5 illustrates the performance of the baseline and coordination algorithms as the number of aircraft in potential conflict with each other is increased. The plot is the result of encounter simulations for both utility fusion strategies. It plots the probability that an encounter results in a loss of minimum separation between any pair of aircraft for different algorithms and utility fusions based on the assumed bank angle and speed distributions. For example, if there were two distinct pairs of aircraft that came into conflict for a four-aircraft scenario throughout its entire simulation, the number of conflict increases by two. In the same scenario, the total number of potential conflicts is $\binom{4}{2} = 6$. Each encounter was simulated for 500 s. The figure also includes error bars indicating one standard deviation from the Monte Carlo simulation results. The figure omits conflict probabilities for the closest-threat command arbitration method as conflicts occur an order of magnitude more frequently than in the coordination method.

It is not surprising that our method performed much better than the closest-threat command arbitration method. Past work suggests that utilizing state–action values from subagents of a complex system can result in better performance than trying to arbitrate over actions from these subagents [13,30,39]. Even the uncoordinated baseline significantly outperformed the closest-threat baseline by five times for the max-sum strategy. As expected from the policy visualization and explanation, the max-min strategy is much more conservative than the max-sum strategy at improving the overall safety. In all cases, the max-min utility fusion produced policies that were about an order of magnitude safer than the max-sum utility fusion policies.

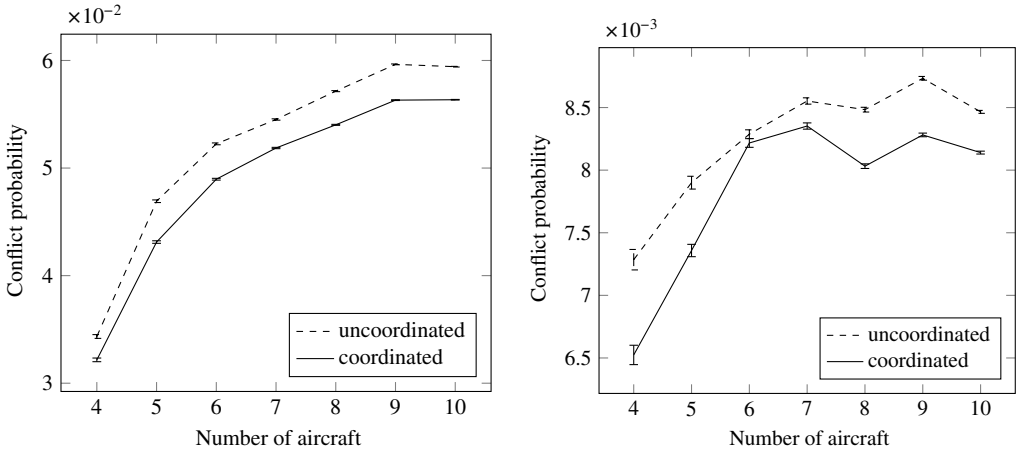
Although there were a small number of loss of minimum separation, there were no cases of near midair collisions (NMACs) in our simulations. Here, we define an NMAC to be when two aircraft come closer than 30 m. Our NMAC threshold of 30 m was obtained from scaling the standard 500 ft. NMAC threshold for traditional aircraft by a reasonable factor for smaller drones [26,40]. This result can be attributed to the penalty on smaller separation distances even in conflict (see reward function in Sec. II).

3. *Decision Time*

The decision time for any algorithm was on the order of 10 ms, which is certainly acceptable given that decisions are made only once every 5 s. The decision time results for each algorithm using the max-min utility function are plotted in Fig. 6, and the max-sum results are essentially identical.

4. *Impact of Search Timeout*

In our simulations, the median number of iterations over each aircraft's action in Algorithm 1 was 6. As there is no convergence



a) Conflict probability with max-sum utility fusion **b) Conflict probability with max-min utility fusion**

Fig. 5 Conflict probability as the number of aircraft in encounter increases.

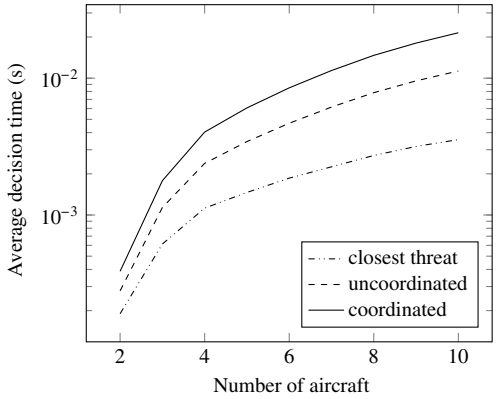


Fig. 6 Average decision times.

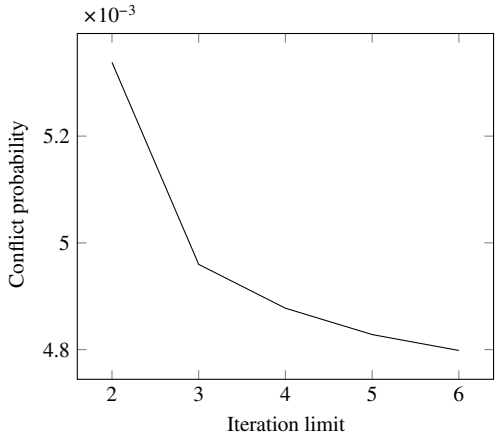


Fig. 7 Impact of search timeout.

guarantee, practical implementation requires the inclusion of a search timeout condition. As expected, Fig. 7 shows that the probability of conflict decreases with iteration limit for the max-min strategy. However, we observe diminishing returns on reduced conflict rates as the limit approaches the median number of iterations in our experiments. Our limit was set conservatively at 100.

VI. Distributed Conflict Resolution

Our algorithm discussed thus far is implemented as a standalone conflict resolution service that works with a separate UTM client server. The UTM client server is designed to provide resource mitigation, contingency management, and system information to operators of UAS flights. In operation, clients such as UAS transport companies would submit and receive flight approvals from the client server. Once the aircraft begins its flight, it will periodically update the client server of its state information such as heading, coordinates, and other data.

Our conflict resolution service listens to data feeds containing real-time flight state information published by the UTM client server and in turn alerts clients in potential conflict. This service features a cluster computing framework for large-scale data processing, streaming analytics for real-time conflict resolution, and built-in system tolerance for stability. The source code can be found at https://github.com/sisl/Discra [41].

A. Distributed Architecture

We now introduce Discra (Distributed Conflict Resolution Architecture), a software framework for massively distributed conflict resolution. The Discra architecture shown in Fig. 8 consists of four main components: 1) parallel ingestors that listen to state updates on the UTM client server's data feeds, 2) a belief state database that the ingestors store and read internal belief state representation from, 3) parallel advisors that publish resolution advisories for clients in potential conflict, and 4) a publish–subscribe advisory server that clients listen to for resolution advisories.

1. Ingestor Server

The ingestor server consists of parallel worker nodes or executors that subscribe to the UTM client server for flight state data streams. For the purposes of our work, we assume that the flight state consists of a global unique flight identifier (GUFI), latitude, longitude, heading, speed, and a response indicator on whether the aircraft is executing a resolution advisory. These streams are generated by UTM clients that periodically update the client server with their flight state information.

The Discra architecture contains multiple ingestion worker processes running concurrently. Here, ingestion refers to a single task of taking in new (and possibly noisy) flight state information, processing it, updating the belief state that estimates the real flight state, and, when a potential conflict arises, publishing a formatted conflict message for the advisor server. A potential conflict is identified by crudely comparing the proximity of current aircraft coordinates with neighbors in the same airspace. The ingestor formats the relevant aircraft belief states into a conflict message as a JavaScript Object Notation (JSON) string and sends it to the advisor server for resolution. The transmission of conflict messages is handled by an internal message broker server.

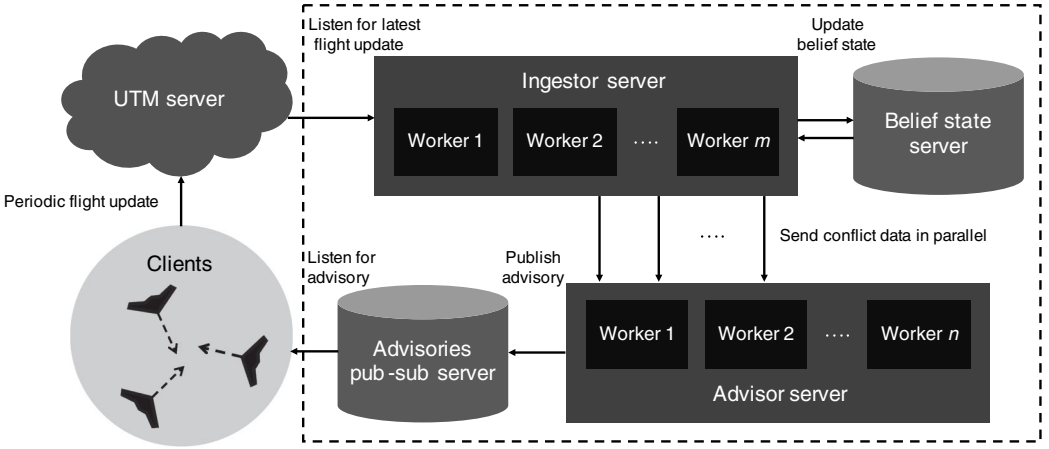


Fig. 8 Discra (boxed) distributes conflict resolution by parallelizing state updates and advisory generation.

The current implementation of the ingestor maps the reported flight states from the UTM client server to belief states using multilinear interpolation on a grid-based discretization scheme. A better alternative is to use a belief update algorithm such as particle sampling, which can simply replace the interpolation routine in the ingestor executors. A good research direction is thus to better estimate aircraft state and account for measurement noise and even temporary communication loss. Because the design of the ingestor is modular, we can simply replace the worker nodes' subroutines with more robust belief update algorithms.

2. *Belief State Database*

The belief state for each client aircraft is represented as an array with probability values assigned to each possible discrete state in our grid-based representation of the MDP state space (see Sec. II). Each entry is timestamped and marked with the corresponding flight's GUFI. We consider two forms of belief state database. First, for each ingestor worker node, a *local* belief state database stores flight entries the executor manages. In this design, each executor is preassigned a geographic region to manage, and the data are stored on the machine the executor process lives on. Second, a *global* belief state database aggregates all entries into a distributed database or, as in the case of our single-machine multicore prototype, a database accessible to all executors.

3. *Advisor Server*

Similar to the ingestor server, the advisor server consists of parallel worker nodes that subscribe to and listen for formatted conflict messages published by the ingestor server. A single task for an advisor executor consists of reading in a new JSON conflict message when one is published, solving for a joint policy using the decompose-and-coordinate procedure established in Secs. III and IV, and publishing a GUFI–advisory pair to a public publish–subscribe server that clients listen to. The published advisory message is formatted in the Extensible Markup Language (XML), which is the standard for UTM client server services.

4. *Advisories Publish–Subscribe Server*

In the prototype implementation, the XML advisories are published to a server that all clients subscribe and listen to for relevant resolution advisories. Advisories are published with information tagged to their client's identifier, such that clients can easily access information relevant to their aircraft. Future implementation will place greater emphasis on information privacy and security, but the focus is for clients to test their flights with a system and get a sense of how traffic will be managed.

B. Implementation

The implementation of Discra relies on the Apache Spark distributed computing framework [42,43] and the Apache Kafka distributed message broker [44,45]. These computing frameworks provide the scalability and fault tolerance that are needed to run the traffic management software on a practical level.

1. *Kafka*

The advisories publish–subscribe server and the internal conflict message relay server between the ingestor and advisor servers are implemented using Kafka. Designed as a unified, high-throughput, low-latency platform for handling real-time data feeds, Kafka is an open-source project with active industry use [46]. In a distributed message broker, producers publish data tagged by "topics" and consumers subscribe to data of a particular topic.

This abstraction facilitates simple separation of data for different processes running in parallel. In our advisories publish–subscribe server, GUFI–advisory pair messages are tagged to airspace regions — the relevant topic on which clients listen to. (An equally valid alternative is tagging advisory messages to unique client identifiers.) The internal conflict message relay server simply uses a single topic from which all advisor worker nodes draw formatted JSON conflict messages from.

2. *Spark*

The ingestor and advisor servers are implemented on the Spark Streaming library, which provides a scalable, fault-tolerant stream processing system [47,48]. In brief, the Spark programming model is centered around the resilient distributed datasets (RDDs) abstraction. RDDs are distributed, partitioned collections of objects that are manipulated through parallel transformations that, in our case, are algorithms that process flight states and conflict messages.

In Spark Streaming, there are multiple executors that each contain a receiver and a task processor. These executors are managed by a process manager/scheduler called a driver, which launches tasks and, if needed, broadcasts objects to the worker nodes. In the advisor server's case, the Spark driver broadcasts policy objects, which contains the lookup table for the pairwise encounter MDP. Receivers buffer data streams in the executors' memory and chop them up into periodic batches. These periodic batches are implemented as an infinite stream of RDDs, which contain flight-state XML objects from the UTM client server and formatted JSON conflict messages from the ingestor server. Once these RDDs are created, the Spark driver launches and delegates tasks to worker nodes. Each task consists of multiple objects that are processed by the worker node they are buffered in. The multiple worker nodes in the Spark processing engine then process these batches of data based on a user-defined algorithm that, in our case, tracks aircraft and resolves

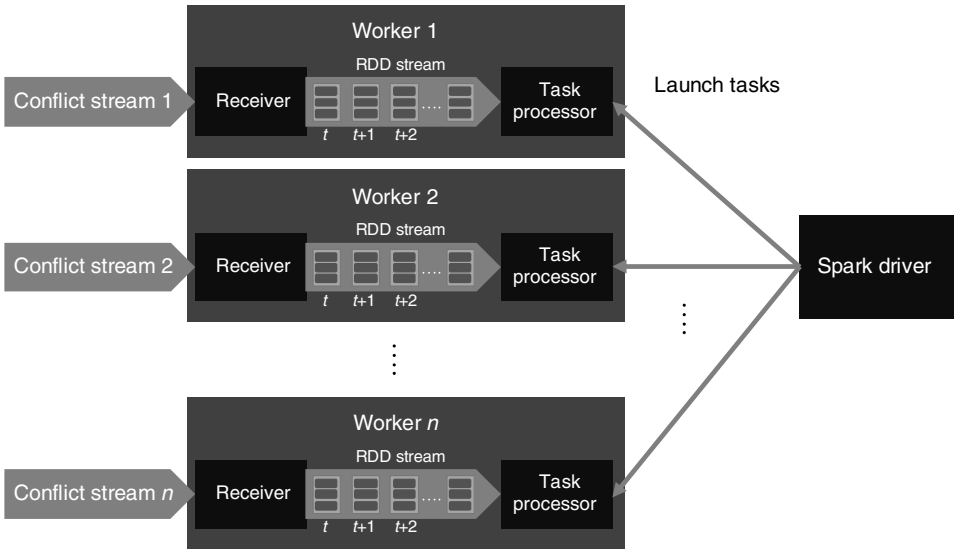


Fig. 9 Spark Streaming model used for the advisor server.

conflicts on our ingestor and advisor servers, respectively. The above description is illustrated in Fig. 9.

VII. Conclusions

A robust and efficient conflict resolution algorithm is presented in this paper. The problem was formulated as an multiagent Markov decision process (MMDP), with uncertainty in the aircraft dynamics, environment, and pilot response. Conflict resolution for this MMDP setting was done by using a decompose-and-coordinate scheme. This scheme involves various principled heuristics such as multilinear interpolation, worst-case utility fusion, and alternating maximization to give an efficient and robust method to multiaircraft conflict resolution. Numerical experiments show that the resolution algorithm can deconflict encounter scenarios with up to 10 aircraft in milliseconds. Our approach also significantly outperformed our benchmark baseline algorithms, which were based on closest-threat heuristics and an uncoordinated algorithm that planned advisories for each aircraft assuming that all intruders followed a white-noise acceleration model. The latter was in contrast to our coordination algorithm, which explicitly models how all unmanned aircraft involved in a potential conflict interact under a set of advisories simultaneously issued.

Even though resolving single multiaircraft conflicts can be done quickly, having to resolve the set of all multiaircraft conflicts simultaneously for a continental airspace serially would be infeasible. Thus, in addition to demonstrating the capabilities of the conflict resolution algorithm, this paper presented Discra, the Distributed Conflict Resolution Architecture. Implemented on the Spark Streaming and Kafka distributed frameworks, Discra schedules and delegates the tasks of resolving conflicts among parallel worker nodes and hence enables large-scale, practical unmanned air traffic management. Specifically, the architecture comprises of four key components: 1) parallel ingestors that actively subscribe to aircraft flight state updates on the unmanned aircraft system traffic management (UTM) server, 2) a belief state database that stores belief states of each aircraft being tracked by UTM, 3) parallel advisors that publish resolution advisories for aircraft in potential conflict, and 4) a distributed publish–subscribe server that clients listen to for resolution advisories. Each component is modular and, overall, provides a consistent platform for our system.

Small unmanned aircraft are extremely useful and critical tools for various commercial activities. In order for these missions to become viable, an efficient and scalable conflict resolution system must be developed. The distributed architecture coupled with the algorithm developed in this paper is a necessary step toward this goal.

Acknowledgments

This research is sponsored by NASA under contract NAS2-03144 with UCSC (UARC). The authors thank Bassam Musaffar and Heinz Erzberger from NASA Ames for their guidance throughout this project. This work has also benefited from conversations with Claire Tomlin, Mo Chen, Qie Hu, Jaime Fernandez-Fisac, Casey D. Mackin, Zachary Sunberg, Eric Mueller, and Michael Owen.

References

[1] D'Onfro, J., "NASA Drone Traffic Management System: This NASA Project Is the Best Hope to Stop a Potential Drone Disaster," *Business Insider*, Sept. 2014.

[2] O'Brien, M., "Drone Traffic Control? NASA, Amazon, Google Partner to Manage Self-Driving Swarms," *San Jose Mercury News*, July 2015.

[3] Pilkington, E., "Amazon Proposes Drones-Only Airspace to Facilitate High-Speed Delivery," *The Guardian*, July 2015.

[4] Kopardekar, P., Rios, J., Prevot, T., Johnson, M., Jung, J., and Robinson, J. E. R., III, "Unmanned Aircraft System Traffic Management Concept of Operations," *AIAA Aviation Technology, Integration, and Operations Conference*, AIAA Paper 2016-3292, 2016. doi:10.2514/6.2016-3292

[5] Kuchar, J. K., and Yang, L. C., "A Review of Conflict Detection and Resolution Modeling Methods," *IEEE Transactions on Intelligent Transportation Systems (ITS)*, Vol. 1, No. 4, 2000, pp. 179–189. doi:10.1.1.15.6885

[6] Schouwenaars, T., Valenti, M., Feron, E., and How, J., "Implementation and Flight Test Results of MILP-Based UAV Guidance," *IEEE Aerospace Conference*, Inst. of Electrical and Electronics Engineers, IEEEAC Paper 1396, 2005, pp. 1–13. doi:10.1109/AERO.2005.1559600

[7] Mellinger, D., Kushleyev, A., and Kumar, V., "Mixed-Integer Quadratic Program Trajectory Generation for Heterogeneous Quadrotor Teams," *IEEE International Conference on Robotics and Automation (ICRA)*, Inst. of Electrical and Electronics Engineers, May 2012, pp. 477–483. doi:10.1109/ICRA.2012.6225009

[8] Augugliaro, F., Schoellig, A. P., and D'Andrea, R., "Generation of Collision-Free Trajectories for a Quadrocopter Fleet: A Sequential Convex Programming Approach," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Inst. of Electrical and Electronics Engineers, Oct. 2012, pp. 1917–1922. doi:10.1109/IROS.2012.6385823

[9] Ong, H. Y., and Gerdes, J. C., "Cooperative Collision Avoidance via Proximal Message Passing," *American Control Conference (ACC)*, Inst. of Electrical and Electronics Engineers, July 2015. doi:10.1109/ACC.2015.7171976

[10] Erzberger, H., and Heere, K., "Algorithm and Operational Concept for Resolving Short-Range Conflicts," *Journal of Aerospace Engineering*, Vol. 224, No. 2, 2010, pp. 225–243. doi:10.1243/09544100JAERO546

[11] Erzberger, H., Lauderdale, T., and Chu, Y., "Automated Conflict Resolution, Arrival Management, and Weather Avoidance for Air

Traffic Management," *International Congress of the Aeronautical Sciences*, Hindawi Publ. Corporation, Cairo, Egypt, 2011, pp. 930–949.
doi:10.1177/0954410011417347

[12] Kochenderfer, M., and Chryssanthacopoulos, J., "Robust Airborne Collision Avoidance Through Dynamic Programming," Project Rept. ATC-371, MIT Lincoln Lab., Lexington, MA, 2011.

[13] Chryssanthacopoulos, J. P., and Kochenderfer, M. J., "Decomposition Methods for Optimized Collision Avoidance with Multiple Threats," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 2, 2012, pp. 398–405.
doi:10.2514/1.54805

[14] AeroVironment, "NASA, AeroVironment and SmartC2 Team for BVLOS UAS Traffic Management Flight Demonstration," AeroVironment, Inc., May 2016, http://www.uasvision.com/2016/05/06/nasa-aerovironment-and-smartc2-team-for-bvlos-uas-traffic-management-flight-demonstration/ [retrieved 01 July 2016].

[15] Nair, R., Tambe, M., Yokoo, M., Pynadath, D., and Marsella, S., "Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings," *International Joint Conference on Artificial Intelligence (IJCAI)*, Morgan Kaufmann Publ. Inc., San Francisco, CA, 2003, pp. 705–711.

[16] Bellman, R., "A Stochastic Multi-Stage Decision Process," *Dynamic Programming*, Princeton Univ. Press, Princeton, NJ, 1957, pp. 61–80, Chap. 2.

[17] Bertsekas, D. P., "Discounted—Problems Computational Methods," *Dynamic Programming and Optimal Control*, 3rd ed., Vol. 2, Athena Scientific, Belmont, MA, 2005, pp. 82–171, Chap. 2.

[18] Puterman, M. L., "Discounted Markov Decision Problems," *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley, Hoboken, NJ, 2005, pp. 142–276, Chap. 6.

[19] Kochenderfer, M. J., "Optimized Airborne Collision Avoidance," *Decision Making Under Uncertainty: Theory and Application*, edited by Kochenderfer, M. J., MIT Press, Cambridge, MA, 2015, pp. 249–275, Chap. 10.

[20] Powell, W. B., "Introduction to Approximation Dynamic Programming," *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, Wiley, Hoboken, NJ, 2007, pp. 109–164, Chap. 4.

[21] Bellman, R., "Multistage Decision Processes and Dynamic Programming," *Adaptive Control Processes*, Princeton Univ. Press, Princeton, NJ, 1961, pp. 51–60, Chap. 3.

[22] Dubins, L. E., "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, Vol. 79, No. 3, 1957, pp. 497–516.

[23] "Unmanned Aerial System Traffic Management Fact Sheet," Digital Press Kit, NASA Ames, Unmanned Aerial Systems Traffic Management Convention, 2015, http://www.nasa.gov/ames/utm2015 [retrieved 02 July 2016].

[24] Semke, W. H., Stuckel, K., Anderson, K., Spitsberg, R., Kubat, B., Mkrtchyan, A., and Schultz, R. R., "Dynamic Flight Characteristic Data Capture for Small Unmanned Aircraft," *Proceedings of the Conference & Exposition on Structural Dynamics — Model Verification & Validation*, Soc. for Experimental Mechanics Inc., Bethel, CT, Feb. 2009.

[25] ICAO Annex 10, *Aeronautical Telecommunications - Surveillance and Collision Avoidance Systems*, Vol. IV, 4th ed., July 2007.

[26] ICAO Doc 4444/ATM 501, "Air Traffic Management: Procedures for Air Navigation Services," 15th ed., 2007.

[27] Boeing, "737 Airplane Characteristics for Airport Planning," Tech. Rept. D6-38A004, Boeing Commercial Airplanes, Renton, Washington, 2013.

[28] Rothkopf, C. A., and Dimitrakakis, C., "Preference Elicitation and Inverse Reinforcement Learning," *Machine Learning and Knowledge Discovery in Databases*, edited by Gunopulos, D., Hofmann, T., Malerba, D., and Vazirgiannis, M., Springer, Berlin, 2011, pp. 34–48.

[29] Lepird, J. R., Owen, M. P., and Kochenderfer, M. J., "Bayesian Preference Elicitation for Multiobjective Engineering Design Optimization," *Journal of Aerospace Information Systems*, Vol. 12, No. 10, 2015, pp. 634–645.
doi:10.2514/1.I010363

[30] Russell, S. J., and Zimdars, A., "Q-Decomposition for Reinforcement Learning Agents," *International Conference on Machine Learning (ICML)*, The International Machine Learning Soc., Aug. 2003, pp. 656–663.

[31] Ong, H. Y., and Kochenderfer, M. J., "Short-Term Conflict Avoidance for Unmanned Aircraft Traffic Management," 2015, https://github.com/sisl/ConflictAvoidanceDASC [retrieved 2 July 2016].

[32] Julier, S., and Uhlmann, J., "Unscented Filtering and Nonlinear Estimation," *Proceedings of the IEEE*, Vol. 92, No. 3, 2004, pp. 401–422.
doi:10.1109/JPROC.2003.823141

[33] Littman, M. L., Cassandra, A. R., and Kaelbling, L. P., "Learning Policies for Partially Observable Environments: Scaling Up," *International Conference on Machine Learning (ICML)*, The International Machine Learning Soc., 1995.

[34] Hauskrecht, M., "Value-Function Approximations for Partially Observable Markov Decision Processes," *Journal of Artificial Intelligence Research*, Vol. 13, No. 1, 2000, pp. 33–94.
doi:10.1613/jair.678

[35] Bezanson, J., Karpinski, S., Shah, V. B., and Edelman, A., "Julia: A Fast Dynamic Language for Technical Computing," arXiv preprint arXiv:1209.5145, 2012.

[36] Kochenderfer, M. J., Edwards, M. W. M., Espindle, L. P., Kuchar, J. K., and Griffith, J. D., "Airspace Encounter Models for Estimating Collision Risk," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 2, 2010, pp. 487–499.
doi:10.2514/1.44867

[37] Dutech, A., and Scherrer, B., "Partially Observable Markov Decision Processes," *Markov Decision Processes in Artificial Intelligence*, edited by Sigaud, O., and Buffet, O., Wiley, Hoboken, NJ, 2013, pp. 187–228.

[38] Chryssanthacopoulos, J. P., and Kochenderfer, M. J., "Collision Avoidance System Optimization with Probabilistic Pilot Response Models," *American Control Conference*, Inst. of Electrical and Electronics Engineers, 2011.
doi:10.1109/ACC.2011.5990776

[39] Rosenblatt, J. K., "Optimal Selection of Uncertain Actions by Maximizing Expected Utility," *Autonomous Robots*, Vol. 9, No. 1, 2000, pp. 17–25.
doi:10.1023/A:1008916000526

[40] "Aeronautical Information Manual," *Aeronautical Information Manual: Official Guide to Basic Flight Information and ATC Procedures*, US Department of Transportation, Federal Aviation Administration, 2016.

[41] Ong, H. Y., and Kochenderfer, M. J., "Discra: Distributed Conflict Resolution Architecture," 2015, https://github.com/sisl/Discra [retrieved 2 July 2016].

[42] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I., "Spark: Cluster Computing with Working Sets," *USENIX Conference on Hot Topics in Cloud Computing*, USENIX Association, Berkeley, CA, 2010, p. 10.
doi:10.1145/1327452.1327492

[43] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," *USENIX Conference on Networked Systems Design and Implementation*, USENIX Association, Berkeley, CA, 2012, p. 2.
doi:10.1145/1057977.1057978

[44] Kreps, J., Narkhede, N., and Rao, J., "Kafka: A Distributed Messaging System for Log Processing," *ACM/SIGMOD NetDB'11 Workshop in Athens*, ACM, New York, June 2011.

[45] Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J., and Stein, J., "Building a Replicated Logging System with Apache Kafka," *Proceedings of the VLDB Endowment*, Vol. 8, No. 12, 2015, pp. 1654–1655.
doi:10.14778/2824032.2824063

[46] Auradkar, A., , Botev, C., Das, S., De Maagd, D., Feinberg, A., Ganti, P., Gao, L., Ghosh, B., Gopalakrishna, K., and Harris, B., et al., "Data Infrastructure at Linkedin," *International Conference on Data Engineering (ICDE)*, IEEE Computer Soc., Washington, D.C., 2012, pp. 1370–1381.
doi:10.1109/ICDE.2012.147

[47] Zaharia, M., Das, T., Li, H., Shenker, S., and Stoica, I., "Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters," *USENIX Conference on Hot Topics in Cloud Computing*, USENIX Association, Berkeley, CA, 2012, p. 10.

[48] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I., "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," *ACM Symposium on Operating Systems Principles*, ACM, New York, 2013, pp. 423–438.
doi:10.1145/2517349.2522737