A Fast Markov Decision Process based Algorithm for Collision Avoidance in Urban Air Mobility

Josh Bertram, *Student Member, IEEE,* Peng Wei, *Member, IEEE* and Joseph Zambreno, *Senior Member, IEEE*

***Abstract*—Multiple aircraft collision avoidance is a challenging problem due to a stochastic environment and uncertainty in the intent of other aircraft. Traditionally a layered approach to collision avoidance has been employed using a centralized air traffic control system, established rules of the road, separation assurance, and last minute pairwise collision avoidance. With the advent of Urban Air Mobility (air taxis), the expected increase in traffic density in urban environments, short time scales, and small distances between aircraft favor decentralized decision making on-board the aircraft. In this paper, we present a Markov Decision Process (MDP) based method, named FastMDP, which can solve a certain subclass of MDPs quickly, and demonstrate using the algorithm online to safely maintain separation and avoid collisions with multiple aircraft (1-on-n) while remaining computationally efficient. We compare the FastMDP algorithm's performance against two online collision avoidance algorithms that have been shown to be both efficient and scale to large numbers of aircraft: Optimal Reciprocal Collision Avoidance (ORCA) and Monte Carlo Tree Search (MCTS). Our simulation results show that under the assumption that aircraft do not have perfect knowledge of other aircraft intent FastMDP outperforms ORCA and MCTS in collision avoidance behavior in terms of loss of separation and near mid-air collisions while being more computationally efficient. We further show that in our simulation FastMDP behaves nearly as well as MCTS with perfect knowledge of other aircraft intent. Our results show that FastMDP is a promising algorithm for collision avoidance that is also computationally efficient.**

***Index Terms*—Collision avoidance, Markov Decision Process.**

I. Introduction

Unmanned aircraft concepts have developed over the past decade from hobbyist drones with limited capabilities into autonomous vehicles capable of travelling beyond line-of-sight with significant range and payload capabilities, and soon unmanned air taxis carrying passengers known as Urban Air Mobility (UAM) [1]–[5] or Advanced Air Mobility (AAM) [6] will be a reality. UAM aircraft will be scheduled in an on-demand basis by passengers using phone-based applications like today's ride-sharing services. This ad hoc demand will be significantly different than today's structured airspace for commercial air traffic. The air taxis will depart from and land at vertical take-off and landing (VTOL) airports known as vertiports and will need to avoid collisions with other aircraft (manned and unmanned), avoid hazards such as terrain and buildings, and respect airspace restrictions such as temporary flight restrictions and restricted flight corridors managed by air traffic control. While it should be expected that some types of structured airspace concepts will be applied such as separating different types of traffic by altitude, corridors or lanes, the unpredictable nature of UAM will result in complex traffic patterns in urban environments that will present new challenges for airspace management.

As compared to today's commercial air traffic management, the smaller scales of operation and flight times mean that the operations tempo will be faster, where rerouting decisions need to be made quickly and without error. Collision avoidance will be difficult as the aircraft will be constrained to similar altitudes, will be closer to the ground than typical commercial traffic, and may be more routinely affected by low-altitude hazards such as migrating birds, low-lying clouds, rain, and other issues that high-flying aircraft can avoid for much of the flight. While we may ultimately use a form of centralized air traffic management in and around an urban area, when we consider communications and surveillance faults or unpredictable (or unknowable) events such as bird flocks, we must acknowledge that some form of on-board collision avoidance capability will be necessary. Moreover, in a layered safety system, an on-board collision avoidance system may be used as part of a larger system of systems designed to ensure safety of passengers and the public.

Collision avoidance is ultimately an algorithmic issue where many potential algorithms can be chosen which each present tradeoffs with respect to other algorithms. Whatever algorithms are chosen, they must balance completeness, optimality, and computational efficiency. Avionics hardware is typically not as capable as traditional hardware available for mainstream computing. [7]–[9] Certification bodies such as the Federal Aviation Administration (FAA) and European Union Aviation Safety Agency (EASA) require a very high level of assurance from computing hardware and software typically including design artifacts from hardware manufacturers, special design processes, and extensive verification. Like other embedded environments, thermal, power, and size limits of the aircraft platform also limit the hardware selected. In computing hardware, this is most painfully felt as requiring low-power processors which are much slower than typical desktop class processors available for mainstream computing. Hardware selected for aviation typically is not the latest, most high-performing hardware available but is instead the most reliable hardware available that fits within the size, weight and power constraints of the platform, and this often means that algorithms will run much more slowly on this embedded hardware.

J. Bertram and J. Zambreno are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, 50011 USA e-mail: bertram1@iastate.edu, zambreno@iastate.edu.

P. Wei is with George Washington University. email: pwei@gwu.edu.

Manuscript received March 1, 2021.

Revision submitted September 29, 2021.

The algorithms that will be most effective for these problems will be ones that can navigate while avoiding collisions with large numbers of aircraft and obstacles while also remaining efficient enough to run on lower-powered, light-weight embedded computing hardware used in avionics with limited processing, memory, and storage capability.

Markov Decision Processes (MDPs) [10] have received attention in Air Traffic Management literature due to recent successes of the Airborne Collision Avoidance System X (ACAS-X) [11], [12]. ACAS-X provides advisories to pilots about impending 1-on-1 collisions with other aircraft and replaces an historical rules-based system known as Traffic Collision Avoidance Systems (TCAS) [13]–[15]. ACAS-X uses a Partially Observable Markov Decision Process (POMDP) formulation to describe a 1-on-1 aircraft encounter and provides the best solution for a pilot to follow to avoid a collision. MDPs are a powerful approach for describing sequential decision making problems and are the mathematical foundation that underpins Reinforcement Learning and Deep Reinforcement Learning.

ACAS-X solves the POMDP offline and stores the optimal policy in memory (or a "table") to look up the action in real time. Multiple ACAS-X variants exist, each with their own lookup tables on the order of 200 MB - 1 GB. Other offline MDP-based collision avoidance approaches have been proposed in [16]–[20] which use various MDP-based problem formulations and all result in large (multi-megabyte) lookup tables which are used at run time. In [16], the authors describe a similar multi-agent collision avoidance problem based on a multi-agent MDP formulation (MMDP) and is solved offline with value iteration and a QMDP heuristic. While the problem required 7 hours to solve and generated a 77 MB lookup table, the lookup table query performed for each pair-wise encounter is fast and results in good collision avoidance behavior. In [17], the authors follow a similar line of investigation to examine multi-rotor collision avoidance with a POMDP formulation solved offline using a QMDP solution method which converts a POMDP into an equivalent MDP which can be more easily solved. The authors report that finding an appropriate state discretization for the problem was possible in 11 days and optimization of POMDP parameters was performed over 3 weeks. While runtime performance measurements are not provided, this type of approach results in a table lookup which should be very fast. The lookup table developed for this approach contained approximately 9.5 million entries. Multi-agent collision avoidance is discussed as an extension but is not covered within the scope of the paper. In [18], the authors use a partially observable MDP (POMDP) solved with nominal belief-state optimization (an approximation method) to perform UAV path planning and explores collision avoidance between a small set of obstacles, including other UAVs. Though many applications and case studies are presented, only very vague computational performance measurements are taken (350 ms). It is unclear if all problems take 350 ms to solve, if the problem takes 350 ms per iteration, etc, so it is difficult to draw conclusions on performance using this approach. In [19], the authors use a mixed-observability MDP (MOMDP) formulation to address vehicles passing other vehicles on roads, which is a special case of a POMDP where certain dimensions of the belief space are assumed to be known in order to make the POMDP tractable. The authors use this approach to implement an agent which can safely enter the lane of oncoming vehicle traffic in order to pass a vehicle while avoiding collision with oncoming vehicles. The authors sidestep the problem of solution time required for the MOMDP and do not provide performance measurements for solving the problem, instead referring readers to the state-of-the-art offline POMDP solvers and indicate that performance will improve as computing technology and POMDP solvers improve. Nonetheless, once the problem is solved, a lookup table approach is used to efficiently obtain the required action, much in the way that ACAS-X operates. In [20], the problem of optimal control of a UAV with feedback in the presence of wind uncertainty is studied, and takes an approach that is equivalent to an MDP (though they do not use that term explicitly.)

In this paper, we describe an online algorithm called FastMDP which performs 1-on-n collision avoidance while remaining computationally efficient. FastMDP differs from traditional MDP solution approaches such as those described above by taking advantage of structure that is present in the MDP's value function which allows for a computationally faster way to solve the MDP. FastMDP solves the MDP online and does not require a large lookup table to be stored. FastMDP also computes portions of the value function on demand, meaning that only the portion of the value function that is needed for the agent to take an action need be computed. These features allow FastMDP to be used on low-power embedded hardware at high rates of processing and can be used in a real-time system. This performance comes at a cost though: FastMDP can currently only operate on a restricted subset of MDPs and is not a generic MDP solver, which we discuss in more detail in Section III, but the restricted MDP subset is still useful in solving problems such as aircraft collision avoidance. We compare the FastMDP algorithm to two baseline algorithms: Optimal Reciprocal Collision Avoidance (ORCA) [21] and Monte Carlo Tree Search [22], [23]. We show how FastMDP can be used to effectively perform collision avoidance and also provide some description of how FastMDP can be tuned for a problem such as aircraft collision avoidance.

Optimal Reciprocal Collision Avoidance (ORCA) [21] is a popular online collision avoidance algorithm that scales to many agents (n-body). ORCA was studied for 1-on-n aircraft collision avoidance in [24] which we compare to FastMDP in this paper.

Monte Carlo Tree Search (MCTS) [22], [23] is an online sampling based approach to solving MDPs. MCTS for 1-on-n aircraft collision avoidance has been explored in [24], [25] in 2D environments with varying strategies to keep the problem tractable, to improve collision avoidance performance, and to scale to large numbers of aircraft, which we also compare to FastMDP in this paper.

The FastMDP algorithm [26] is an online approach that solves MDPs quickly and has been applied to collision avoidance (2D [27] and 3D [28]), terminal area guidance [29], pursuit-evasion (dog fighting) [30], and pre-departure flight

planning [31] and has been shown to scale to thousands of aircraft. While the FastMDP algorithm has shown promise, this paper is the first extensive study of the algorithm compared to other well known collision avoidance algorithms. This paper sets out to answer the question of whether FastMDP is competitive with state of the art collision avoidance algorithms.

The major contributions of this paper are:

- Extension of the algorithm from [26] to deterministic, terminating MDPs.
- Extension of FastMDP with an intruder intent model.
- Characterization of collision avoidance, throughput, and computational performance for FastMDP and two other online collision avoidance algorithms.
- Extension of proofs from [28] to cover MDPs with terminal positive rewards.

We provide background on MDPs and related topics in Section II. In Section III we describe and extend the FastMDP algorithm. Section IV describes the experimental setup and simulation environment. In Section V, we show how this new algorithm compares to the baseline algorithms and provide concluding remarks in Section VI.

II. Background

A. Markov Decision Processes

MDPs are a framework for sequential decision making with broad applications to finance, robotics, operations research and many other domains [32]. MDPs are formulated as the tuple (S, A, R, T) where $s_t \in S$ 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, $r_t = R(s_t, a_t, s_{t+1})$ 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 captures the probability $p(s_{t+1} \mid s_t, a_t)$ 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 describes the expected future reward that will be obtained within the environment by following the policy π and can be expressed over an infinite horizon as:

$$V^\pi(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \left[\gamma^t R(s_t, a_t, s_{t+1}) \mid \pi\right]\right], \tag{1}$$

where s_0 is the initial state, γ is a discount factor that defines the infinite horizon and lies in the range $0 < \gamma < 1$, and s_{t+1} is sampled from the distribution described by the transition function $T(s_t, a_t, s_{t+1})$. The discount factor γ serves to balance immediate reward with future reward. Small values of γ close to zero favor immediate rewards, whereas large values of γ near one favor long term rewards. The value function aggregates all of the agent's knowledge about future expected reward into a single metric that can then be used as an indicator of how to obtain the best future expected reward.

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 future reward. 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 $s \in S$. For a given MDP, the optimal value function $V^*(s)$ is unique but multiple optimal policies $\pi^*(s)$ may exist which result in the same value function $V^*(s)$.

The fundamental way to solve a MDP is by way of a relation known as the Bellman Update Equation:

$$V(s_t) = \max_a \sum_{s_{t+1}} T(s_t, a, s_{t+1}) \left[R(s_t, a, s_{t+1}) + \gamma V(s_{t+1})\right] \tag{2}$$

which is a recursive relation that indicates that the value of the current state is related to the value of the states it is connected to. For the optimal value function, $V^*(s)$, the value at every state is maximized and following this relationship results in an optimal trajectory through the state space. The optimal value function can be found using an iterative approach known as Value Iteration which applies the Bellman Update Equation across the state space at each iteration. The Value Iteration algorithm is guaranteed to converge to the optimal value function $V^*(s)$ down to an arbitrarily small stopping condition ϵ known as the Bellman residual. Value Iteration is known to converge in polynomial time [33], [34] but this is polynomial in the size of the state space $|S|$ and action space $|A|$.

Historically, many algorithms have been devised to solve MDPs. A fundamental problem with MDPs is that the size of the state space $|S|$ and the size of the action space $|A|$ both can grow quickly either due to high-dimensionality of the space, or due to increasingly fine-grained discretization to approach a continuous representation. Bellman [10] referred to this as the the curse of dimensionality and much of the literature has been devoted to finding ways to mitigate these effects to keep a particular MDP tractable so that a solution may be found. We now describe one such method known as Monte Carlo Tree Search.

B. Monte Carlo Tree Search

MCTS [22] is a powerful approach to solve MDPs when a model is available for simulation of future possible actions but the state-action space is too large to represent efficiently. MCTS was explored for aircraft collision avoidance in [24] and we here we provide a brief summary of the main features of the algorithm.

MCTS is an approach to solving MDPs which uses trajectory sampling to arrive at a statistical representation of expected return. Starting from the current state, possible actions are tentatively explored. Actions which seem promising are then further expanded focusing search effort on actions which seem to be leading to higher reward. Multiple trajectories are sampled as the tree is built in order to develop a statistical understanding of the expected reward of each action.

As the tree is developed, a successively better approximation of expected reward is developed. In the limit with an infinite number of samples, the solution provided by MCTS converges

to the true value function for the MDP. In practice, a pre-determined finite number of samples is used by defining some termination criteria such as a fixed number of trajectories having been explored, or a pre-determined time budget has elapsed. After the termination criteria is met, the best available action is selected by examining the children of the root of the search tree to determine which action results in the most expected reward. This best action is then taken by the agent in the environment. MCTS is thus an any-time algorithm where a result can be provided at any point but can improve if more computational time is available. MCTS can then be used in real-time systems such as UAVs assuming enough computational power is available to compute a meaningful result.

III. FastMDP

FastMDP solves a certain class of MDPs very quickly compared to other methods by exploiting structure in the value function. FastMDP is built upon the observation that the Bellman Update results in predictable *peaks* in the value function, where the peaks are generated by rewards in the MDP. Given the rewards, FastMDP determines the corresponding peaks and is able to reconstruct the value function, leading to the ability to solve the MDP more directly than the traditional iterative approaches. In one sense, the Bellman Update can be considered a performance bottleneck in determining the solution, and FastMDP offers a way to bypass this bottleneck. FastMDP cannot currently be applied to a general MDP and is instead currently restricted to a subset of MDPs. This paper covers the following subclass of MDPs: (a) deterministic transition function, (b) a state space which maps to an underlying metric space with an available distance metric (e.g., Euclidean distance metric), (c) a terminating MDP where upon receiving a positive reward the MDP transitions to an absorbing state $S^{\varnothing}$. This subclass of MDP corresponds to an agent (e.g., an aircraft) exploring a 2D or 3D world with deterministic actions and dynamics where the simulation ends when the agent reaches a goal (e.g., a vertiport). (The non-terminating case is explored in [26].)

Formally, let us define a distance function $\delta(s, s_i)$ as the minimum number of actions $a \in A$ needed to reach state $s_i \in S$ starting from state $s \in S$.

$$\delta(s, s_i) = \min_t \{t \mid T(s, a_1, a_2, \cdots, a_t = s_i\}, \tag{3}$$

where $a_1, a_2, \cdots, a_t$ represent a sequence of actions taken at each time step, and $T(s, a_1, \cdots, a_t)$ represents the transition function applied at each time step t using the sequence of actions starting at state s.

We define a *point source reward* as a positive reward of magnitude r_g that is obtained at only one state s_g and zero everywhere else:

$$r(s) = \begin{cases} r_g > 0 & \text{if } s = s_g \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Theorem 1. *The value function for a deterministic terminating MDP with a single positive point source reward with magnitude r_g at state s_g with MDP discount factor $0 < \gamma < 1$ has the form:*

$$V(s) = \gamma^{\delta(s, s_g)} \cdot r_g, \tag{5}$$

where $s \in \mathcal{S}$.

Proof. By definition, if our initial state is s_g we collect reward with magnitude r_g, whereupon the MDP terminates, resulting in a value at state s_g of $V(s_g) = r_g$.

Let us denote the set of all states k-steps from s_g as $S^k = \{s \in S \mid \delta(s, s_g) = k\}$ where $k \geq 0$ is an integer, with $S^0 = \{s_g\}$, S^1 with all states one step from s_g, etc. Let us now assume that we start not at state s_g, but at some state $s_1 \in S^1$. As no immediate reward is collected at state s_1 if we take an optimal action $a^* \in A$ which leads to S^0, the expected future reward is:

$$\begin{aligned} V(s_1) &= \gamma \cdot V(s_g) \\ &= \gamma^{\delta(s_1, s_g)} \cdot V(s_g) \\ &= \gamma^{\delta(s_1, s_g)} \cdot r_g. \end{aligned} \tag{6}$$

Suppose for states $s_{k+1} \in S^{k+1}$ and $s_k \in S^k$:

$$V(s_{k+1}) = \gamma \cdot V(s_k). \tag{7}$$

Then for states $s_2 \in S^2$ and $s_1 \in S^1$:

$$\begin{aligned} V(s_2) &= \gamma \cdot V(s_1) \\ &= \gamma \cdot \left[\gamma^{\delta(s_1, s_g)} \cdot r_g\right] \\ &= \gamma \cdot \gamma \cdot r_g \\ &= \gamma^{\delta(s_2, s_g)} \cdot r_g. \end{aligned} \tag{8}$$

Then by induction, we see that the value of any state s is as follows, completing the proof:

$$V(s) = \gamma^{\delta(s, s_g)} \cdot r_g. \tag{9}$$

□

When considering multiple positive rewards, multiple peaks form in the value function and the resulting value function is the max over all peaks at each state $s \in S$. To prove this, we will show equivalence to the Bellman optimality equation $V^* = LV^*$, where L is the Bellman operator. It is well known that the Bellman operator L is a contraction mapping over the max norm of the functional $V^\pi(s)$ of all possible value functions resulting from all possible policies $\pi \in \Pi$, and that the optimal value function V^* is a fixed-point solution to $V = LV$ due to the contraction mapping property of L [35], making the optimal value function V^* a unique solution and $V^*(s) \geq V^\pi(s)\ \forall s \in S$ for all policies $\pi \in \Pi$ and it follows that $V^*(s) = \max_{\pi \in \Pi} V^\pi(s),\ \forall s \in S$.

Let $S^+ = \{s \in S \mid R(s, a) > 0\}$ be the set of states where positive reward is collected, and $S^Z = \{s \in S \mid R(s, a) = 0\}$ be the set of states where no reward is collected.

Note that the value at any state $s_i \in S^+$ for a terminating MDP is by definition known and fixed, as the state s_i is the absorbing state where reward R_i with magnitude r_i is collected and no subsequent reward can be collected. The value at such

a state s_i is $V(s_i) = r_i$. Thus what remains is to identify the value at the other states in S^Z.

Theorem 2. *The maximum of the optimal value function $V^*(s)$ occurs within S^+.*

Proof. We prove by contradiction. Let $s_{max} = \arg\max_{s \in S} V^*(s)$ and assume $s_{max} \in S^Z$. From Theorem 1, it is clear that any state $s \in S^Z$ requires one or more steps to reach its goal s_g and is therefore in S^k where $k > 0$. But for $k > 0$, $s_{k+1} \in S^{k+1}$, and $s_k \in S^k$, $V(s_{k+1}) = \gamma \cdot V(s_k) = \gamma^k \cdot V(s_g)$ and given $0 < \gamma < 1$, then $V(s_{k+1}) < V(s_k) < V(s_g)$ and implies that s_{max} then lies within S^0 and therefore S^+ which is a contradiction. □

Theorem 3. *States with reward of zero, S^Z, are determined from the states with non-zero reward, S^+.*

Proof. If we examine the recursive form of the Bellman equation at the optimal policy π^* with the (stationary) V^*:

$$V^*(s) = \max_a \sum_{s' \in S} T(s, a, s') \left[R(s, a) + \gamma V^*(s') \right], \quad (10)$$

where s' is a possible next state resulting from taking action a from state s and note that the immediate reward $R(s_z, a) = 0,\ \forall s_z \in S^Z, \forall a \in A$, then for $s_z \in S^Z$ the value $V^*(s_z)$ is then determined only by discounted future reward:

$$\begin{aligned} V^*(s_z) &= \sum_{s' \in S} T(s, a^*, s') \gamma V^*(s') \\ &= \gamma \sum_{s' \in S} T(s, a^*, s') V *^(s'), \end{aligned} \quad (11)$$

noting that $\sum_{s' \in S} T(s, a, s') = 1$.

Thus, for the optimal action $a^* \in A$:

$$a^* = \arg\max_a \sum_{s'} T(s_z, a, s') V^*(s'), \quad (12)$$

we can rewrite the Bellman equation as,

$$V^*(s_z) = \gamma \sum_{s'} T(s_z, a^*, s') V^*(s') \quad (13)$$

And given that the discount factor $0 < \gamma < 1$, we see that:

$$V^*(s_z) \leq \sum_{s'} T(s_z, a^*, s') V^*(s') \quad (14)$$

Furthermore, if $V^*(s') > 0$, then:

$$V^*(s_z) < \sum_{s'} T(s_z, a^*, s') V^*(s'), \quad (15)$$

which is to say that $V^*(s_z)$ can only be equal to $V^*(s')$ if both are zero.

Consider a sequence of states in S^Z over n time steps, $\{ s_z^{(1)}, s_z^{(2)}, \cdots, s_z^{(n)} \}$ and suppose that each element in the sequence is the result of the optimal action $a^* \in A$ at each step that satisfies $a^* = \arg\max_a \gamma \sum_{s'} T(s_z, a, s') V^*(s')$. Let us say that at time step $n+1$, by taking the optimal action a^* we reach some state $s_p \in S^+$. If we then consider s_p and $s_z^{(n)}$, we recognize that that $s_p \in S^0$ and $s_z^{(n)} \in S^1$ and that $s_z^{(i)} \in S^{i-(n-1)}$. Thus by induction, all states $s_z \in S^Z$ lie within some S^k with $k > 0$ and their value is therefore derived from the value of states in S^+. □

Theorems 2 and 3 taken together imply that that all sequences starting in S^Z must therefore terminate in S^+. The question is from a given state $s_z \in S^Z$, which in which state $s_p \in S^+$ will the sequence terminate?

Theorem 4. *For a deterministic MDP with N positive terminal rewards $R_i \in \{R_1, \cdots, R_N\}$ with reward R_i located at s_i having magnitude r_i and a single point source reward value function of $V_i(s) = \gamma^{\delta}(s, s_i) \cdot r_i$, the MDP's optimal value function $V^*(s)$ is:*

$$V^*(s) = \begin{cases} r_i & \text{if } s = s_i \in S^+ \\ \max_i V_i(s) & \text{if } s \in S^Z. \end{cases} \quad (16)$$

Proof. By definition of this MDP with terminating positive rewards, the value at any state within S^+ is known and fixed to be $V(s_i) = r_i$, as the state s_i is an absorbing state where reward R_i with magnitude r_i is collected and no subsequent reward can be collected.

For all states in $s_z \in S^Z$, the expected value of obtaining reward R_i located at state $s_i \in S^+$ is:

$$V_i(s_z) = \gamma^{\delta(s_z, s_i)} \cdot r_i. \quad (17)$$

It follows that the maximum value is then obtained at each state with:

$$V^*(s_z) = \max_i V_i(s_z). \quad (18)$$

By definition, all states in S^+ are at a location of a reward, and a point source reward R_i at the reward's state s_i has value of $V_i(s_i) = r_i$. Therefore the maximum possible value at each state is then $V(s) = \max_i V_i(s),\ \forall s \in S$ completing the proof. □

In [26] it is shown that in a deterministic MDP, negative point source rewards do not have the same exponential decay behavior and are confined to a single state where the negative reward is located. The concept of a risk well is introduced to model the exponential decay of a negative reward which behaves similarly to the positive reward above. Risk wells can be explicitly constructed with a finite number of negative rewards carefully constructed to result in the desired exponential decay, but it was shown that this was computationally intensive to do so. Instead, the risk well is modelled as a single negative reward with magnitude r_n centered at a state s_n which decays exponentially out to a radius R_n. With this model, a similar algorithm used to compute positive rewards can be used to also compute the negative risk wells. In [26], this result was not proven and only shown to approximate the true value function with both positive and negative rewards, and thus our solution here is also only an approximation of the true value function $V^*(s)$ when negative rewards are present and we leave proofs for negative reward for future work. Figure 1 illustrates a risk well.

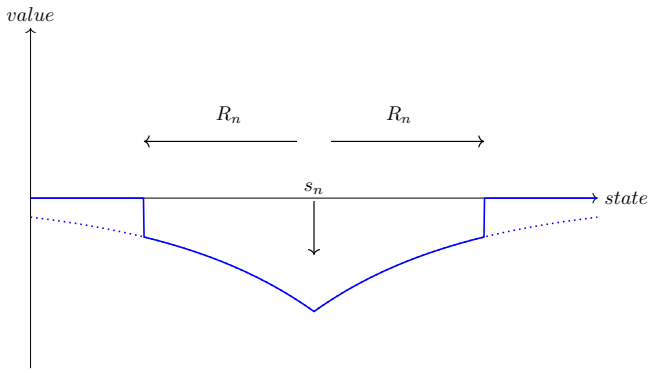

Fig. 1. Risk well centered at state s_n with radius R_n.

For a negative reward R_j with magnitude r_j at state s_j, the well P_j^- formed by the negative reward is:

$$P_j^-(s) = \gamma^{\delta(s,s_j)} \cdot -r_j. \tag{19}$$

A positive peak P_i^+ is the value function formed from a positive reward R_i with magnitude r_i at state s_i is defined as:

$$P_i^+(s) = \gamma^{\delta(s,s_i)} \cdot r_i, \tag{20}$$

The value function V^+ from all N^+ positive reward peaks $P^+ = \{P_1, P_2, \cdots, P^{N^+}\}$ is the state-wise maximum over all positive peaks:

$$V^+(s) = \max_i P_i^+(s), \forall i = \{1, \cdots, N^+\}. \tag{21}$$

Likewise, the value function V^- for all N^- negative rewards peaks $P^- = \{P_1, P_2, \cdots, P^{N^-}\}$ is the state-wise minimum over all negative peaks:

$$V^-(s) = \min_j P_j^-(s), \forall j = \{1, \cdots, N^-\}. \tag{22}$$

The resulting value function for the MDP is then the state-wise sum of V^+ and V^-:

$$V^*(s) = V^+(s) + V^-(s). \tag{23}$$

Forward projection is used to determine which states are reachable within the state space given the allowable actions the agent can take from the current state. The forward projection applies an action (or actions) to the dynamics equations (defined in Section IV) for a duration of time and returns the resulting state(s).

Previous FastMDP approaches [26]–[28] assumed that intruder aircraft maintained their present heading and velocity when performing planning. In this paper we expand on this previous work by borrowing the roll-out concept from MCTS to better predict intent of intruder aircraft. At each planning step of the algorithm, we begin by computing a tree T_I for each intruder aircraft I out to a depth R_W in seconds (an experimental parameter) in steps of R_K seconds (an experimental parameter) where each intruder takes a random action for the duration of the R_K time step. We perform this randomized action roll-out multiple times growing T_I into a tree of possible actions that the intruder may perform. The tree overall will contain a number of nodes of order $K = |A|^D$ nodes $N = \{N_1, \cdots, N_K\}$, where $|A|$ is the number of actions that can be taken from a given state. Here the actions intruders can take R_A are a configurable parameter. We refer to this tree T_I as the intent tree and the nodes $N_K \in N$ as intent nodes. As a computational optimization, we generate these intent nodes at the beginning of the algorithm's operation each time it is run, running it for all aircraft. During the execution of the algorithm when we need to know the intent of the intruders, we filter out the ownship's pre-computed intent nodes, leaving only the intruders' intent nodes. These intent nodes are then converted into negative rewards and fed into the FastMDP algorithm, along with a single positive reward with magnitude 200 corresponding to the ownship's destination vertiport. Each negative reward is assigned a magnitude of R_M (a parameter) and a radius based off the "loss of separation" (discussed in next section) distance D_{LOS} times a scaling factor R_S (a parameter).

See Algorithm 1 for a description of the FastMDP algorithm. Lines 2-4 set initial conditions for the algorithm, where the actions $\mathbf{A}$ and limits $\mathbf{L}$ for the agent are provided as inputs. Each time step of the simulation where FastMDP is invoked (line 5), the intent tree T_I is rebuilt to describe the likely future actions of all aircraft (line 6). For each aircraft controlled by FastMDP (line 7), we perform the core of the FastMDP algorithm starting at line 8 where we load the current state s_t of the aircraft under control (i.e., the ownship) where t indicates the current time. Line 10 builds the positive peak(s) associated with the ownship's goal. Line 11 builds the negative peak(s) associated with all intruders (from the perspective of the ownship) using the information in the intent tree T_I. Line 13 performs forward projection of the dynamics of the ownship using the list of possible actions (and limits) which computes a set of trajectories in the form of a set of points $\mathbf{\Delta}$ which represent the reachable states from the current state s_t. Starting at line 15 we begin to compute the value for these reachable states for each state $s_j \in \mathbf{\Delta}$ (line 16). We compute the value associated with each positive peak p_i, where i is an index identifying which positive peak, by computing distance d_p from the current state to the reward, and use it to compute the value gained by obtaining the reward (lines 19-22), saving off the highest value contribution (line 24) known as the "dominant reward". We do a similar calculation (lines 26-32) to identify the negative value associated with the negative rewards n_k, where k is an index indicating which negative peak, and the reward that contributes the most negative value (line 33). Note that the term ρ_n is a binary logical value of 0 or 1 and indicates whether the radius of the negative peak has been exceeded (0) or not (1) and is a multiplication term (line 31) which implements truncation of the negative value beyond the radius r_n per the definition of a risk well above. We compute the MDP value function at state s_j which identifies the expected value of reaching state s_j. The most valuable action a_{max} is identified and recorded (lines 37-39) so that it can be taken in the environment. As this environment in this

paper is a simulation based environment, the action is taken in simulation on line 42, and we note that all aircraft take their actions in this simulation simultaneously.

Our expected usage for this algorithm is for it to run on-board an aircraft in real-time. In this configuration, the run time complexity of the algorithm (line 6, lines 8-39) is $O(|\Delta| \times |R|)$ where $|\Delta|$ is the number of reachable states to consider and $|R|$ is the number of rewards to consider. For this problem, we compute the reachable states for each action over a time horizon defined by a constant parameter c_1 which means that $|\Delta| = c_1|A|$. At any time step, the number of rewards is $|R|$ which is some constant c_2 times the number of intruders $|I|$ for this problem, $|R| = c_2|I|$. Thus the complexity of the algorithm is $O(|A| \times |I|)$ for this problem. One cautionary note for readers is that $|A|$ can quickly explode and must be carefully managed. This dependency on $|A|$ is the reason that the algorithm in its current form is restricted to a finite action space A. Note that there is no dependence on the size of the state space $|S|$, which is what allows the algorithm to be used on continuous state spaces. MCTS has a similar dependence on a finite action space but can also be used on a continuous state space. Most algorithms based on MDPs have some dependence on the size of the state space, which also often grows exponentially as the fidelity of a problem increases (i.e., Bellman's well known "curse of dimensionality" [10].)

Algorithm 1 FastMDP algorithm.

1: **procedure** FASTMDP
2: $\mathbf{S} \leftarrow$ *initial aircraft states*
3: $\mathbf{A} \leftarrow$ *aircraft actions (a priori)*
4: $\mathbf{L} \leftarrow$ *aircraft limits (a priori)*
5: **while** aircraft remain **do**
6: $T_I \leftarrow$ *(re)build intent tree for intruders*
7: **for** each aircraft **do**
8: $s_t \leftarrow \mathbf{S}[aircraft]$
9: // Build peaks
10: $\mathbf{P}^+ \leftarrow$*pos reward for destination vertiport*
11: $\mathbf{P}^- \leftarrow$*neg rewards from* T_I
12: // Perform forward projection of aircraft dynamics
13: $\mathbf{\Delta} \leftarrow fwdProject(s_t, \mathbf{A}, \mathbf{L})$
14: // Compute the value at each reachable state
15: $\mathbf{V}^* \leftarrow$ *allocate space for each reachable state*
16: **for** $s_j \in \mathbf{\Delta}$ **do**
17: // First for positive peaks
18: **for** $p_i \in \mathbf{P}^+$ **do**
19: $d_p \leftarrow \|s_j - \mathbf{location}(p_i)\|_2$ ▷ distance
20: $r_p \leftarrow \mathbf{reward}(p_i)$
21: $\gamma_p \leftarrow \mathbf{discount}(p_i)$
22: $\mathbf{V}^+(p_i) \leftarrow |r_p| \cdot \gamma_p^{d_p}$
23: **end for**
24: $V^+_{max} \leftarrow \max_{p_i} \mathbf{V}^+$
25: // Next for negative peaks
26: **for** $n_k \in \mathbf{P}^-$ **do**
27: $d_n \leftarrow \|s_j - \mathbf{location}(n_k)\|_2$ ▷ distance
28: $\rho_n \leftarrow d_n < \mathbf{radius}(n_k)$ ▷ within radius
29: $r_n \leftarrow \mathbf{reward}(n_k)$
30: $\gamma_n \leftarrow \mathbf{discount}(n_k)$
31: $\mathbf{V}^-(n_k) \leftarrow \rho_n \cdot |r_n| \cdot \gamma_n^{d_n}$
32: **end for**
33: $V^-_{max} \leftarrow \max_{n_k} \mathbf{V}^-$
34: $\mathbf{V}^*[s_j] \leftarrow V^+_{max} - V^-_{max}$
35: **end for**
36: // Identify the most valuable action
37: $a_{max} \leftarrow \arg\max_a(\mathbf{V}^*)$ per method F_M (parameter)
38: // Record each aircraft's action
39: $\mathbf{A^*}_{t+1}[aircraft] \leftarrow a_{max}$
40: **end for**
41: // Now that all aircraft have selected an action, apply it
42: $\mathbf{S} = SimulationUpdate(\mathbf{A^*}_{t+1})$
43: **end while**
44: **end procedure**

IV. Experimental Setup

All three algorithms are evaluated in the same simulation environment, a 2D aircraft simulation from [24], [25]. The aircraft are represented with a continuous state space and a discretized action selection. Uncertainty is present in the environment in the form of Gaussian process noise: noise is added to both the selected heading and velocity at each time step. While the environment uses simplified aircraft dynamics, the noise applied leads to a challenging environment within which the agent must plan. When describing the planning, we refer to the aircraft for which a plan is being constructed as the "ownship" and all other aircraft as "intruders". The simulation operates by treating each aircraft as the ownship and at each simulation time step all aircraft complete a planning cycle. Within the simulation all aircraft construct their plan simultaneously and all actions are taken simultaneously.

The aircraft dynamics model used in this simulation is:

$$\dot{v} = a_v + \epsilon_v \tag{24}$$

$$\dot{\phi} = a_\phi + \epsilon_{\dot{\phi}} \tag{25}$$

$$\dot{\psi} = \frac{g \tan \phi}{v} \tag{26}$$

$$\dot{x} = v \cos \psi \tag{27}$$

$$\dot{y} = v \sin \psi \tag{28}$$

where a_v is the commanded change in airspeed and a_ϕ is the commanded change in bank angle. At each step Gaussian process noise is inserted into the velocity ϵ_v and at the bank angle $\epsilon_{\dot{\phi}}$. Of the formulations used in [24], [25], we use the 3-action version of the simulation which selects only changes in bank angle and always selects no commanded change in airspeed (though noise is still applied at each step). This form is a challenging enough environment that collision avoidance can be tested without needing to use the 9-action version which is slower.

In the environment, there are a number of vertiports which agents must navigate to. The vertiports are arranged such that they generate conflict so that collision avoidance can be effectively measured. There are two levels of collision, with terminology taken from the aviation community: a "loss of separation" conflict (LOS) and a "near mid-air collision" (NMAC). Within this simulation taken from [24], a LOS occurs when two aircraft are within 926 meters, and an NMAC occurs when two aircraft are within 150 meters. The simulation maintains a record of reward earned over time (only used for training MCTS, not required for FastMDP.) Positive reward of 1 is provided to the agent if the agent reaches its destination

vertiport. Negative reward of -1 is provided to the agent if an NMAC event occurs and -0.5 if a LOS event occurs. At each time step, a -0.001 fuel penalty is applied. An agent then obtains reward by navigating its aircraft to the destination vertiport while avoiding collisions with other agents.

We assess performance of the algorithms along three dimensions: computational performance, collision avoidance performance, and throughput. Computational performance is measured as the time that is required to perform one planning cycle for an agent on the computing hardware. Collision avoidance performance is measured by counting the number of LOS and NMAC events that occur per hour of simulated time. Throughput is measured by the number of aircraft that reach their destination vertiport per hour of simulated time. As a summary metric, the reward received over time is also instructive in comparing the algorithms. Simulations are performed over 120000 seconds (approx. 33 hrs) of simulated time with 10 random seeds to obtain a statistical mean and standard deviation of the performance of the algorithms over different random seeds. Simulations were performed on a compute cluster with Intel(R) Xeon(R) CPU X5650 cores running at 2.67GHz. Each run was confined to a single CPU core and is implemented single-threaded to isolate multicore effects from the algorithms' performance.

Simulation begins with a single aircraft spawned at a randomly selected vertiport. New aircraft are spawned at randomly selected times as the simulation progresses, until a pre-specified number of aircraft (10,000) have been spawned, at which point no new aircraft are generated. Aircraft are removed from the simulation when they reach their vertiport. If NMACs occur, the aircraft involved are temporarily removed from simulation and reintroduced in the simulation at a later time in order to continue to obtain useful encounters despite an NMAC occurring. (The purpose of the simulation is to generate as wide a variety of encounters as possible in order to best evaluate the algorithm.) The algorithms all cause aircraft to navigate to the vertiports and generally reach a steady state of approximately 30-40 aircraft in the air at any one time. Simulation terminates when all aircraft reach their destinations.

A. Intruder Intent

To study the effects of intent, we examine three variations of MCTS. In the first variation which we will refer to as MCTS-perfect, following [24] we provide MCTS with perfect knowledge of the intruders' actions as they are developed. Consider a planning horizon with time steps $W = \{w_1, w_2, \cdots, w_N\}$. At each time step w_i, all agents are informed of all other agents decisions at time step w_{i-1}. For this to be possible for real aircraft, the agents would require perfect communications with zero latency and zero packet loss, and the agents would need to operate on a common clock so that the decision time steps $\{\cdots, w_i, w_{i+1}, \cdots\}$ occur simultaneously so that all agents decisions from w_i are available to all agents at w_{i+1}. In practice, these assumptions would not hold for real systems but serves as useful upper bound on collision avoidance performance as perfect knowledge should always perform better than imperfect knowledge.

We define two variations of MCTS which are fully decentralized and have no intent messages from intruders. First we define a variant MCTS-straight which assumes that all intruders continue on their present course. Second we define a variant MCTS-random which assumes that all intruders take random actions over the planning window W.

We additionally include the ORCA algorithm as a well known algorithm that is not based on an MDP formulation. In this implementation from [24], the n-body ORCA algorithm has no knowledge of the intruders' intent. The ORCA algorithm inherently plans such that the agents take actions which are designed not to conflict with possible actions of other agents and in the absence of uncertainty should generate conflict free trajectories. Due to ORCA's inherent design, in these simulations ORCA not only has heading control but also is allowed speed control and therefore has better control authority in the simulations than the other two algorithms. Within the simulation, for all algorithms equivalent Gaussian noise is injected into the heading and speed after the commanded heading and speed changes are applied leading to all algorithms being evaluated under uncertainty in the following simulations.

B. Experiment Design

In this section we describe experiments performed on the FastMDP algorithm, both to identify the best performing variation of FastMDP and to understand what aspects of the problem are most relevant for safety. We also include a description of this procedure to illustrate how to perform tuning of the FastMDP algorithm for problems of interest to future readers. We define experimental parameters which may be numerical values inherent in the algorithm (e.g., MDP discount factor γ) sometimes referred to as *hyperparameters*, or may be parameters that capture hypotheses on the best problem formulation (e.g., how to express intruder intent). Our intent here is optimization for a specific problem (i.e., collision avoidance) but not good behavior over a range of problems (e.g., landing, takeoff). Note that the MCTS baseline [24], [25] we compare to had a similar investigation performed in defining the best performing MCTS and experimental parameters and our intent is to measure against that published work.

For a problem of interest to a reader, there is no simple way to describe how to choose what parameters to study for a given problem. However, the authors suggest the geometry of positive and negative rewards be selected to be of an appropriate size for the agent's dynamic constraints (e.g., minimum turn radius) and several possible geometries be considered (such as how we have examined the intruder roll-out in this paper.) We also suggest that for a problem with distances of the scale used in this problem, the distance metric used will also affect the value of the MDP discount factor γ used. In this problem, we measure distance with meters and found that we needed to use a relatively high value of γ. If other units were used (such as kilometers), it may be that a lower value of γ may be needed (i.e., perhaps 0.8 or 0.9.) The parameter γ here defines the rate of exponential decay

of the rewards and the most important aspect of γ is that it is chosen to provide an identifiable gradient to the agent over the expected range of the reachable points that will be sampled by the algorithm. The parameter γ must be chosen so that the expected distances are not too far out on the tails of the exponential decay so that two closely sampled states have a detectable difference (i.e., larger than the order of magnitude of the smallest numbers representable by floating point numbers in computer hardware.) When using single-precision floating point, this can be a problem but typically is not an issue with double-precision floating point. Note that this choice of the value of γ is inherent in any MDP formulation and any algorithm used to solve the MDP and the selection of γ is a commonly reported problem in the literature. FastMDP offers insight into why certain values of γ may work for a problem while others may not as γ defines the rate of exponential decay of peaks in the value function.

Multiple possible influencing factors were identified and isolated into configuration variables. Test runs using a single random seed were performed initially altering one variable at a time from a baseline setting. Promising settings from the first round of testing were combined into mixtures of parameters until the best performing candidates were identified. Final candidates were evaluated with multiple random seeds using the same procedure as the baseline algorithms to develop a statistical characterization over the multiple random seeds. In all, 34 runs were performed, and 2 final candidates were selected for multiple random seed runs. To differentiate between the runs, we evaluated each run for the number of NMAC events, LOS events, throughput, and computational performance to try to strike a balance between them, generally favoring lower NMACs and LOS over throughput and computational performance.

Table I identifies the factors that we hypothesized may affect the FastMDP algorithms performance. Table II defines the experiments that were performed. The default parameters that form the baseline were $R_S = 1.0$, $R_W = 20$, $R_A = \{-5, -2.5, 0, 2.5, 5\}$, $R_K = 5$, $R_M = 1000$, $R_F = True$, $F_W = 60$, $F_M = mean$. Experiments 1-26 tested one change at a time from the defaults, and only parameter values that differ from these defaults are called out in the experiment definitions due to space considerations. Experiments 27-34 were a second round of experiments formed by combining promising results from experiments 1-26 in the hope of identifying synergies between factors.

TABLE I
HYPOTHESIZED INFLUENCING FACTORS FOR EXPERIMENTATION.

Factor	Description
R_S	Scale of negative rewards formed around intruder roll-out positions
R_W	Intruder roll-out window length in seconds
R_A	Actions taken by intruder during roll-out
R_A	Actions taken by intruder during roll-out
R_K	Time-step delta between intruder positions during roll-out
R_M	Magnitude of negative rewards formed around intruder roll-out positions
R_F	Whether to apply a filter on the intruder positions during roll-out which favors intruder positions closest to the intruder's goal
F_W	FastMDP's forward projection window length in seconds
F_M	Method FastMDP uses to select the best action; one of { mean, sum, max }.

V. RESULTS

We first discuss the experiment results for FastMDP and then compare the best candidates with the baseline algorithms ORCA and MCTS.

A. Experiment results

Table III summarizes the experiment results for the experiments we conducted. Experiments 1-3 examine the effect of the rollout scale parameter R_S and do not show a strong preference for any of the values. We therefore should generally favor a smaller value of R_S as larger values will from these results unnecessarily reserve a larger portion of the airspace with higher values. Experiments 4-7 evaluate the rollout window length R_W parameter and show that values above 45 seconds appear to have a negative effect in this problem while also reducing computational performance. Experiments 8-10 study the effect of the intruder's actions parameter R_A when performing the intruder roll-out. The results here appear to be inconclusive and do not seem to significantly favor higher or lower fidelity action definition of the intruders during roll-out, though we can observe that higher fidelity actions appear to provide a throughput boost. Experiments 11-13 study the effect of the roll-out time step parameter R_K and indicate that smaller values appear to perform much better in terms of avoiding NMACs. Interestingly, we also see that with smaller R_K we also appear to receive a throughput boost at the expense of significantly increasing computational cost. Experiments 14-17 examine the impact of the magnitude of the negative rewards formed at each intruder position computed during roll-out. From these runs it is difficult to understand if there is a significant difference here but we can observe that lower values at or below the value of positive reward magnitude of 200 appear to provide a modest benefit. We observe that low values of 100 appear to provide improved throughput, reduced computational demand, but at an expense of more conflicts. This can be explained as the reduced negative penalty as providing more wiggle room for the agent to cut corners during conflict. This more risky behavior provides a benefit near congested vertiports where multiple agents wish to land, resulting in higher throughput (and therefore reduced average number of aircraft in flight at any one time.) Given the potential safety impacts however, we feel that this increased risk taking is not a desirable characteristic of the agent and therefore will favor R_M values of 200 or 500. Experiments 18-19 study the effect of a filtering mechanism that we hypothesized would lead to improve agent behavior. We expected that behavior would be improved with $R_F = True$, but the data suggests otherwise showing a clear preference for $R_F = False$. The effect of this filter is to remove approximately half of the intruder's roll-out

TABLE II
EXPERIMENT DEFINITIONS.

Exp Num	Description
1	$R_S = 1.25$
2	$R_S = 1.50$
3	$R_S = 2.00$
4	$R_W = 25$ sec
5	$R_W = 30$ sec
6	$R_W = 45$ sec
7	$R_W = 60$ sec
8	$R_A = \{-5, 0, 5\}$ deg
9	$R_A = \{-5, -2.5, 0, 2.5, 5\}$ deg
10	$R_A = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$ deg
11	$R_K = 5$ sec
12	$R_K = 2$ sec
13	$R_K = 1$ sec
14	$R_M = 1000$
15	$R_M = 500$
16	$R_M = 200$
17	$R_M = 100$
18	$R_F = True$
19	$R_F = False$
20	$F_W = 60$ sec
21	$F_W = 75$ sec
22	$F_W = 45$ sec
23	$F_W = 30$ sec
24	$F_M = mean$
25	$F_M = sum$
26	$F_M = max$
27	$R_S = 1$, $R_W = 45$, $R_A = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$, $R_K = 1$, $R_M = 100$, $R_F = False$, $F_W = 30$, $F_M = sum$
28	$R_S = 1$, $R_W = 45$, $R_A = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$, $R_K = 1$, $R_M = 200$, $R_F = False$, $F_W = 30$, $F_M = sum$
29	$R_S = 1$, $R_W = 45$, $R_A = \{-5, -2.5, 0, 2.5, 5\}$, $R_K = 1$, $R_M = 200$, $R_F = False$, $F_W = 45$, $F_M = sum$
30	$R_S = 1$, $R_W = 30$, $R_A = \{-5, -2.5, 0, 2.5, 5\}$, $R_K = 2$, $R_M = 500$, $R_F = False$, $F_W = 45$, $F_M = sum$
31	$R_S = 1$, $R_W = 30$, $R_A = \{-5, -2.5, 0, 2.5, 5\}$, $R_K = 1$, $R_M = 500$, $R_F = False$, $F_W = 45$, $F_M = sum$
32	$R_S = 1$, $R_W = 30$, $R_A = \{-5, -2.5, 0, 2.5, 5\}$, $R_K = 1$, $R_M = 500$, $R_F = False$, $F_W = 30$, $F_M = sum$
33	$R_S = 1$, $R_W = 30$, $R_A = \{-5, -2.5, 0, 2.5, 5\}$, $R_K = 2$, $R_M = 500$, $R_F = False$, $F_W = 45$, $F_M = sum$
34	$R_S = 1.25$, $R_W = 30$, $R_A = \{-5, -2.5, 0, 2.5, 5\}$, $R_K = 2$, $R_M = 500$, $R_F = False$, $F_W = 45$, $F_M = sum$

TABLE III
EXPERIMENT RESULTS.

Exp Num	Through-put	Perf (ms)	NMACs	LOSs
1	166.22	20.37	9	92
2	167.64	20.45	9	87
3	167.57	18.70	8	94
4	166.15	22.18	4	69
5	166.76	18.60	7	59
6	164.47	20.50	3	67
7	161.53	44.08	33	318
8	166.26	23.82	9	91
9	165.92	22.48	12	95
10	210.06	30.27	5	124
11	165.70	21.88	10	85
12	166.92	19.73	7	87
13	212.78	58.22	1	79
14	166.88	21.11	7	107
15	166.24	21.23	10	106
16	166.26	21.92	4	109
17	211.99	12.23	3	149
18	165.06	19.45	3	22
19	212.74	37.77	1	38
20	166.54	22.80	11	88
21	166.35	19.70	4	88
22	168.41	19.30	6	40
23	216.00	7.04	2	124
24	168.85	22.59	2	35
25	166.81	18.35	13	93
26	206.82	26.21	582	7403
27	187.01	255.99	35	245
28	185.68	217.39	43	241
29	170.65	250.66	4	31
30	194.54	21.41	3	31
31	192.58	16.03	1	19
32	192.03	234.25	2	26
33	194.75	42.10	4	37
34	176.81	79.23	4	29

positions by determining the mean distance of all of the roll-out positions to the intruder's destination vertiport, and only keeping those points which were below the mean distance. It was hypothesized that among all of the possible randomly generated roll-out positions, the intruder would tend to favor those positions which were closest to the vertiport. Removing the unimportant positions was hypothesized to improve performance by not having to consider positions that are unlikely to be visited. In reality, however, inspection of simulation runs showed that due to the congestion in this environment, agents must often turn away from their destination vertiports in order to avoid collisions. Therefore we favor $R_F = False$ for this problem. Experiments 20-23 explore the effect of the length of the FastMDP algorithm's forward projection window. We expected that a longer forward projection window would result in better performance, but the data suggests that shorter windows result in better performance and we therefore favor values of 30 or 45 seconds. Experiments 24-26 examine the mechanism used by FastMDP to determine which action is the best action. The "mean" and "sum" actions are not that significantly different but the "max" method is clearly inferior. Experiments 27-34 are used to explore mixtures of promising parameters from the earlier experiments in order to identify possible synergies between parameters. Through a process of iterative examination of results, we ultimately selected experiments 19 and 31 for performing more extensive

tests. These runs had good NMAC performance (our highest-weighted consideration), reasonable LOS performance, and reasonable throughput values. In subsequent sections, we label these runs as FastMDP-19 and FastMDP-31 in the plots.

B. FastMDP comparison with Baseline Algorithms

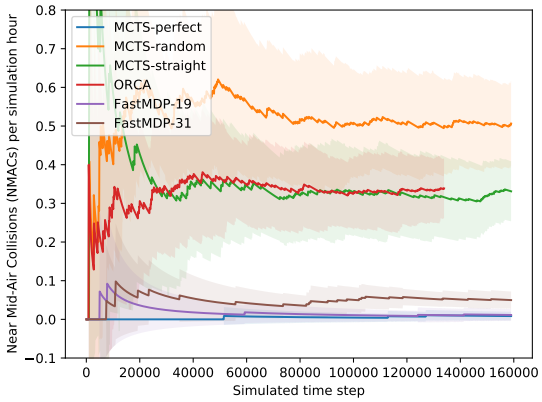

Fig. 2. Near Mid-Air Collision (NMAC) events per hour of simulated time.

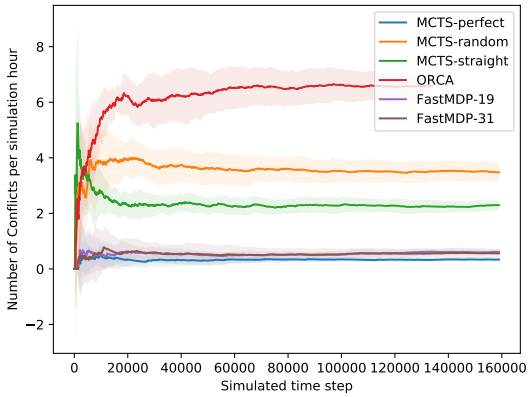

Fig. 3. Loss of Separation (LOS) events per hour of simulated time.

We now present the two candidate tunings of the FastMDP for this problem against the baseline algorithms.

From the experiment definition section, recall that we examine three versions of the MCTS algorithm: MCTS-perfect, MCTS-straight, and MCTS-random. MCTS-perfect has perfect knowledge of intruder intent (an unrealistic assumption for a real-world problem, but a useful bound on what we should expect as a best case for collision avoidance performance). MCTS-random assumes intruders select random actions during MCTS roll-out, and MCTS-straight assumes intruders maintain current heading during MCTS roll-out. We also compare to the ORCA algorithm, a well-known collision avoidance algorithm which is not based on a MDP-based formulation, as a control variable on whether MDPs are a suitable way to represent this collision avoidance problem. Note that for each algorithm, 10 simulations runs were performed with different random seeds in order to obtain a statistical measurement of each algorithm's performance. In the plots, the dark lines indicate the mean where the shaded regions indicate standard deviation.

TABLE IV
NEAR MID-AIR COLLINS (NMAC) EVENTS OVER 10 RANDOM SEEDS SORTED BY MEAN NUMBER OF EVENTS PER SIMULATION HOUR.

Algorithm	Mean	Std Dev	Mean/Hr
MCTS-random	22.4	4.8	0.507
ORCA	11.9	3.0	0.335
MCTS-straight	14.6	3.4	0.331
FastMDP-31	2.0	0.775	0.052
FastMDP-19	0.5	0.671	0.011
MCTS-perfect	0.4	0.5	0.008

TABLE V
LOSS OF SEPARATION (LOS) EVENTS OVER 10 RANDOM SEEDS SORTED BY MEAN NUMBER OF EVENTS PER SIMULATION HOUR.

Algorithm	Mean	Std Dev	Mean/Hr
ORCA	233.7	16.3	6.578
MCTS-random	153.6	14.7	3.478
MCTS-straight	101.4	8.8	2.295
FastMDP-19	27.1	4.5	0.615
FastMDP-31	21.8	6.7	0.567
MCTS-perfect	14.9	3.4	0.337

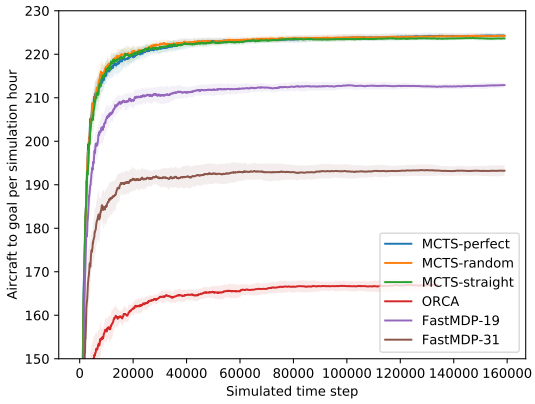

Fig. 4. Throughput per hour of simulated time.

Figure 2 and Table IV show the resulting NMAC events per hour of simulated time for each algorithm. The data shows

TABLE VI
THROUGHPUT (AIRPLANES ARRIVING AT DESTINATION PER HOUR OF SIMULATED TIME) OVER 10 RANDOM SEEDS SORTED FROM SLOWEST TO FASTEST.

Algorithm	Mean (ms)	Std Dev (ms)
ORCA	162.8	13.2
FastMDP-31	190.1	13.9
FastMDP-19	209.8	14.4
MCTS-perfect	220.8	14.7
MCTS-straight	220.6	14.7
MCTS-random	221.0	14.7

TABLE VII
ALGORITHM PERFORMANCE PER AIRCRAFT OVER 10 RANDOM SEEDS SORTED FROM SLOWEST TO FASTEST.

Algorithm	Mean (ms)	Std Dev (ms)
MCTS-random	175.5	11.9
MCTS-perfect	155.2	10.4
ORCA	139.5	7.9
MCTS-straight	117.4	7.7
FastMDP-31	68.3	4.7
FastMDP-19	29.8	1.7

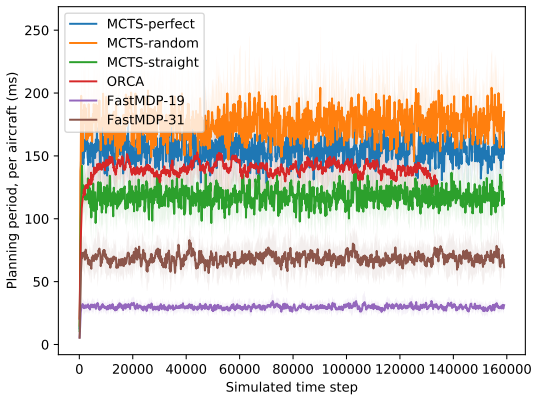

Fig. 5. Computational time of the algorithms.

that MCTS-random has the most NMACs/hr and that ORCA and MCTS-straight have about equivalent NMACs/hr in these simulations. (Note that due to job duration constraints ORCA was not able to finish the simulation run.) As expected, MCTS-perfect acheives the best NMAC/hr performance because it has perfect knowledge of intruders' intent and should represent the best we can expect for performance in these simulations. What is interesting is that the two FastMDP runs achieve a similar level of avoiding NMAC events despite not having access to this perfect knowledge that is available to MCTS-perfect. Due to the scale of the plots, it is difficult to discern MCTS and

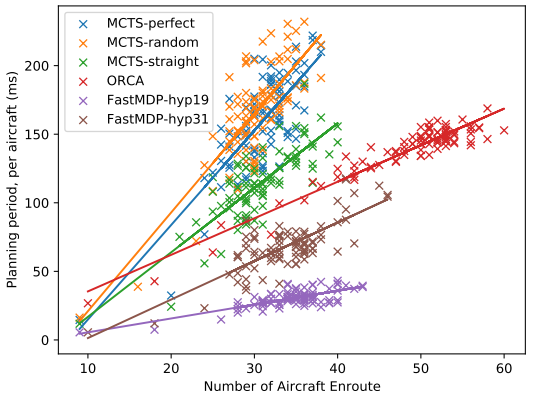

Fig. 6. Computational time varying with number of aircraft.

FastMDP-19's performance on the plot and Table IV should be consulted. Note that in this plot, the early spikes result from large initial variance as some simulation runs generate NMACs early on while others do not. As more time elapses, the variances over the simulation runs average out revealing the expected steady state performance of each algorithm.

Figure 3 and Table V show the Loss of Separation (LOS) event per hour of simulated time for each algorithm. Here we see that ORCA generates the highest rate of LOS events for these simulations. MCTS-random and MCTS-straight result in somewhat better but still elevated levels of LOS events. Here again we see that MCTS-perfect achieves the best results due to having perfect knowledge of intruders' intent, and again we see that the two FastMDP runs achieve nearly the same level of conflict avoidance as MCTS-perfect. Due to the scale of the plot, Table V should be consulted to more easily discern the small difference in performance between FastMDP-19, FastMDP-31 and MCTS-perfect.

Figure 4 and Table VI show the throughput per hour of simulated time for each algorithm. Here the MCTS algorithms variants excel achieving the best throughput, though it may be difficult to discern on the plot that the plots for MCTS-perfect, MCTS-random, and MCTS-straight nearly coincide. This is an interesting result in that the intruder intent does not seem to have a strong impact on the throughput, despite it having a large impact on the number of LOS and NMAC events above. We expected that having a better understanding of where the intruder is likely to go would result in a better ability for an agent to "slip by" other agents, but this does not appear to be the case for this environment. We speculate that in this environment, the agents are not often able to choose actions that fit our narrowed definition of intent and instead more closely match more random definition of intent (which corresponds more closely to what other agents are capable of doing rather than what they are likely to do.) In this plot we also see a more marked difference between the two FastMDP runs. FastMDP-19 achieves a higher throughput as compared to FastMDP-31. Recall from Table II that FastMDP-19 eliminates an intruder roll-out filtering mechanism which was hypothesized would improve performance, but appeared not to from the data. FastMDP-31 mixed multiple promising parameter values in an attempt to identify synergies. What is apparent from this plot is that this mixture of parameter values used for FastMDP-31 resulted in a slight decrease of throughput as compared to the values used for the FastMDP-19 run.

Figure 5 and Table VII show the computational performance of each algorithm at each time step during the simulation. Here we see that MCTS-perfect and MCTS-random perform approximately equally to ORCA, with MCTS-straight performing slightly more efficiently. We see here that the FastMDP variants are much more efficient than either the MCTS or ORCA algorithms for this problem. Here again we see that the FastMDP-19 variant has an advantage over the FastMDP-31 variant due to the FastMDP-31's larger number of intruder roll-out positions that are generated due to the parameter settings.

Figure 6 show how the computational performance of each

algorithm varies with the number of aircraft in flight. The simulations initially start with one aircraft spawned at a randomly selected vertiport and new aircraft are spawned at random intervals over time. The plot shows how the computation time varies for the number of aircraft currently in flight. The regression trend lines show that as expected performance increases for all algorithms with fewer aircraft and the advantages of using the FastMDP approach is more substantial with larger numbers of aircraft.

Taken in aggregate, FastMDP-19 appears to be the best performing FastMDP variant in these simulation runs. In general, from these results we can say that FastMDP can be tuned to trade collision avoidance performance with throughput and that a satisfactory level of throughput can be achieved while obtaining collision avoidance performance that approaches MCTS with perfect knowledge of the intruders' intent, while being more computationally efficient. In particular, the FastMDP-19 variant achieves a speed up factor of $139.5s/29.8s = 4.7$ versus ORCA and a speed up factor of $155.2/29.8 = 5.2$ versus MCTS-perfect in these simulations.

VI. Conclusion

In this paper, we demonstrate the FastMDP algorithm applied to a 2D collision avoidance problem and compare the performance to Monte Carlo Tree Search (MCTS) and Optimal Reciprocal Collision Avoidance (ORCA) in an environment with significant uncertainty. We compare the algorithms along collision avoidance, throughput, and computational performance metrics and find that FastMDP executes approximately 5 times faster than MCTS and ORCA while achieving collision avoidance performance that exceeds ORCA and nearly approaches MCTS with perfect knowledge of intruders' intent with only a modest decrease in throughput as compared to MCTS. We performed experiments that both identified the best performing parameters for this problem, but can also serve as a guide to future researchers on how to tune FastMDP for other problems. These results show that FastMDP is competitive with ORCA and MCTS and in some ways can outperform both algorithms in uncertain environments, despite FastMDP's current limitation of a restricted subset of MDPs. These conclusions motivate further study of the FastMDP approach for collision avoidance problems.

VII. Acknowledgement

The authors wish to thank Xuxi Yang for access to his original simulation environment and implementations of ORCA and MCTS that this paper used as a baseline. The authors also wish to thank the anonymous reviewers and the editors for their helpful and constructive feedback during the review process. We appreciate the time and careful thought they put into the review comments which we believe strengthened the final version of the paper.

References

[1] M. Mulvaney and M. Kratsios, "M-18-22: Memorandum for the Heads of Executive Departments and Agencies - FY 2020 Administration Research and Development Budget Priorities," Executive Office of the President, the White House, Tech. Rep., 2018.

[2] R. Vought and K. Droegemeier, "M-19-25: Memorandum for the Heads of Executive Departments and Agencies - FY 2021 Administration Research and Development Budget Priorities," Executive Office of the President, the White House, Tech. Rep., 2019.

[3] David P. Thipphavong and Rafael Apaza and Bryan Barmore and Vernol Battiste and Barbara Burian and Quang Dao and Michael Feary and Susie Go and Kenneth H. Goodrich and Jeffrey Homola and Husni R. Idris and Parimal H. Kopardekar and Joel B. Lachter and Natasha A. Neogi and Hok Kwan Ng and Rosa M. Oseguera-Lohr and Michael D. Patterson and Savita A. Verma, "Urban air mobility airspace integration concepts and considerations," in *In Proceedings of Aviation Technology, Integration, and Operations Conference*, Atlanta, GA, 2018. [Online]. Available: doi:10.2514/6.2018-3676

[4] The Federal Aviation Administration, "Urban Air Mobility (UAM) Conceopt of Operations v1.0," Tech. Rep., 2020. [Online]. Available: https://nari.arc.nasa.gov/sites/default/files/attachments/UAM_ConOps_v1.0.pdf

[5] National Academies of Sciences, Engineering, and Medicine, "Advancing Aerial Mobility: A National Blueprint," The National Academies Press, Washington, DC, Tech. Rep., 2020. [Online]. Available: https://doi.org/10.17226/25646

[6] National Aeronautics and Space Administration, "Advanced Air Mobility," Tech. Rep., 2020. [Online]. Available: https://www.nasa.gov/aam

[7] T. Gaska, C. Watkin, and Y. Chen, "Integrated modular avionics - past, present, and future," *IEEE Aerospace and Electronic Systems Magazine*, vol. 30, no. 9, pp. 12–23, 2015.

[8] Y. Yeh, "Safety critical avionics for the 777 primary flight controls system," in *20th DASC. 20th Digital Avionics Systems Conference (Cat. No.01CH37219)*, vol. 1, 2001, pp. 1C2/1–1C2/11 vol.1.

[9] H. F. Grip, J. Lam, D. S. Bayard, D. T. Conway, G. Singh, R. Brockers, J. H. Delaune, L. H. Matthies, C. Malpica, T. L. Brown, A. Jain, A. M. S. Martin, and G. B. Merewether, *Flight Control System for NASA's Mars Helicopter*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2019-1289

[10] R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[11] M. J. Kochenderfer and J. Chryssanthacopoulos, "Robust airborne collision avoidance through dynamic programming," *Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371*, 2011.

[12] J. P. Chryssanthacopoulos and M. J. Kochenderfer, "Decomposition methods for optimized collision avoidance with multiple threats," in *2011 IEEE/AIAA 30th Digital Avionics Systems Conference*, 2011, pp. 1D2–1–1D2–11.

[13] W. H. Harman, "Tcas- a system for preventing midair collisions," *The Lincoln Laboratory Journal*, vol. 2, no. 3, pp. 437–457, 1989.

[14] C. Munoz, A. Narkawicz, and J. Chamberlain, *A TCAS-II Resolution Advisory Detection Algorithm*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2013-4622

[15] J. K. Kuchar and L. C. Yang, "A review of conflict detection and resolution modeling methods," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, pp. 179–189, 2000.

[16] H. Y. Ong and M. J. Kochenderfer, "Markov decision process-based distributed conflict resolution for drone air traffic management," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 1, pp. 69–80, 2017. [Online]. Available: https://doi.org/10.2514/1.G001822

[17] E. R. Mueller and M. Kochenderfer, *Multi-Rotor Aircraft Collision Avoidance using Partially Observable Markov Decision Processes*. AIAA Modeling and Simulation Technologies Conference, 2016. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2016-3673

[18] S. Ragi and E. K. P. Chong, "Uav path planning in a dynamic environment via partially observable markov decision process," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 4, pp. 2397–2412, 2013.

[19] V. Sezer, "Intelligent decision making for overtaking maneuver using mixed observable markov decision process," *Journal of Intelligent Transportation Systems*, vol. 22, no. 3, pp. 201–217, 2018. [Online]. Available: https://doi.org/10.1080/15472450.2017.1334558

[20] R. P. Anderson, E. Bakolas, D. Milutinović, and P. Tsiotras, "Optimal feedback guidance of a small aerial vehicle in a stochastic wind,"

Journal of Guidance, Control, and Dynamics, vol. 36, no. 4, pp. 975–985, 2013. [Online]. Available: https://doi.org/10.2514/1.59512

[21] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19.

[22] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Computers and Games*, H. J. van den Herik, P. Ciancarini, and H. H. L. M. J. Donkers, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–83.

[23] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[24] X. Yang and P. Wei, "Autonomous free flight operations in urban air mobility with computational guidance and collision avoidance," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–14, 2021.

[25] X. Yang and P. Wei, "Scalable multi-agent computational guidance with separation assurance for autonomous urban air mobility," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 8, pp. 1473–1486, 2020. [Online]. Available: https://doi.org/10.2514/1.G005000

[26] J. R. Bertram, "A new solution for markov decision processes and its aerospace applications," *Graduate Theses and Dissertations. 17832.*, 2020. [Online]. Available: https://lib.dr.iastate.edu/etd/17832

[27] J. Bertram, X. Yang, M. W. Brittain, and P. Wei, *Online Flight Planner with Dynamic Obstacles for Urban Air Mobility*. AIAA Aviation 2019 Forum, 2019. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2019-3625

[28] J. Bertram and P. Wei, *Distributed Computational Guidance for High-Density Urban Air Mobility with Cooperative and Non-Cooperative Collision Avoidance*. AIAA Scitech 2020 Forum, 2020. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2020-1371

[29] ——, *An Efficient Algorithm for Self-Organized Terminal Arrival in Urban Air Mobility*. AIAA Scitech 2020 Forum, 2020. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2020-0660

[30] ——, *An Efficient Algorithm for Multiple-Pursuer-Multiple-Evader Pursuit/Evasion Game*. AIAA Scitech 2021 Forum, 2021. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2021-1862

[31] J. Bertram, P. Wei, and J. Zambreno, *Scalable FastMDP for Pre-departure Airspace Reservation and Strategic De-conflict*. AIAA Scitech 2021 Forum, 2021. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2021-0779

[32] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. [Online]. Available: http://www.jstor.org/stable/24900506

[33] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the complexity of solving markov decision problems," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, ser. UAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 394–402.

[34] P. Tseng, "Solving h-horizon, stationary markov decision problems in time proportional to log(h)," *Oper. Res. Lett.*, vol. 9, no. 5, p. 287–297, Sep. 1990. [Online]. Available: https://doi.org/10.1016/0167-6377(90)90022-W

[35] O. Sigaud and O. Buffet, *Markov decision processes in artificial intelligence*. John Wiley & Sons, 2013.

Josh Bertram Josh Bertram is a PhD candidate at Iowa State University and a Machine Learning / Artificial Intelligence researcher at the Johns Hopkins University Applied Physics Lab. Previously, he completed his Masters at Iowa State University and focused on Markov Decision Processes and their applications to aerospace problems. Josh worked at Collins Aerospace for 16 years in real-time embedded systems and as a Machine Learning / Artificial Intelligence engineer. Josh's interests are in the intersection of embedded computing, avionics, and artificial intelligence where size, weight, and power constraints restrict the types of algorithms that can be employed.

Joseph Zambreno Joseph A. Zambreno has been with the Department of Electrical and Computer Engineering at Iowa State University since 2006, where he is currently a Professor, Associate Chair, and director of the Reconfigurable Computing Lab (RCL). Prior to joining ISU he was at Northwestern University in Evanston, IL, where he graduated with his Ph.D. degree in Electrical and Computer Engineering in 2006, his M.S. degree in Electrical and Computer Engineering in 2002, and his B.S. degree summa cum laude in Computer Engineering in 2001. While at Northwestern University, Dr. Zambreno was a recipient of a National Science Foundation (NSF) Graduate Research Fellowship, a Northwestern University Graduate School Fellowship, a Walter P. Murphy Fellowship, and the EECS department Best Dissertation Award for his Ph.D. dissertation titled "Compiler and Architectural Approaches to Software Protection and Security." He is a recent recipient of the NSF CAREER award (2012), as well as the ISU award for Early Achievement in Teaching (2012), the College of Engineering Outstanding Achievement in Teaching Award (2019), and the ECpE department's Warren B. Boast undergraduate teaching award (2009, 2011, 2016).

Peng Wei Peng Wei is an assistant professor in the Department of Mechanical and Aerospace Engineering at the George Washington University, with courtesy appointments at Electrical and Computer Engineering Department and Computer Science Department. By contributing to the intersection of control, optimization, machine learning, and artificial intelligence, he develops autonomy and decision support tools for aeronautics, aviation and aerial robotics. His current focus is on safety, efficiency, and scalability of decision making systems in complex, uncertain and dynamic environments. Recent applications include: Air Traffic Control/Management (ATC/M), Airline Operations, UAS Traffic Management (UTM), eVTOL Urban Air Mobility (UAM) and Autonomous Drone Racing (ADR). He is leading the Intelligent Aerospace Systems Lab (IASL). He is an associate editor for AIAA Journal of Aerospace Information Systems. He received his Ph.D. degree in Aerospace Engineering from Purdue University in 2013.