Online Flight Planner with Dynamic Obstacles for Urban Air Mobility

Joshua R. Bertram Xuxi Yang Marc Brittain Peng Wei
Iowa State University
Ames, IA 50011
{bertram1, xuxiyang, marcb, pwei}@iastate.edu

Urban Air Mobility and Unmanned Aerial Vehicles will result in numerous aircraft which will need to generate flight plans in response to a dynamically changing airspace due to unforeseen emergencies and contingencies in order to safely reach their destinations while avoid collisions. We propose an online computational guidance algorithm for flight planning using Markov Decision Processes (MDP) that is capable of computing flight plans for resource constrained embedded computers found on Unmanned Aerial Vehicles which we anticipate will operate in UAS Traffic Management (UTM) and Urban Air Mobility (UAM) contexts. We propose a novel algorithm for efficiently solving a certain class of MDPs and show how UAV flight planning can be formulated within this framework. We show that as obstacles change dynamically, the algorithm can quickly compute new trajectories making it suitable for an online planner.

NASA, Uber and Airbus have been exploring the concept of Urban Air Mobility (UAM) [1–5] where vertical takeoff and landing (VTOL) aircraft may be either human piloted or autonomous for passenger transport in personal commuting or on-demand air taxiing. UAM operations are expected to fundamentally change cities and people's lives by reducing commute times and stress. Development of efficient algorithms for vehicle technology and airspace operation will be critical for the success of UAM. A critical question is whether structured air space management is required or whether a more loosely controlled "Free Flight" model is possible. Due to the computational complexity of free flight, most research is focusing on a structured approach. In this paper we propose a online computational guidance algorithm which could be used on board an aircraft with limited computing power to support a Free Flight paradigm, or to provide backup planning capability on board the aircraft to enable safe and efficient flight operations in on-demand urban air transportation.

The concept of "Free Flight" was proposed primarily for future air transportation applications because it has the potential to cope with the ongoing congestion of the current ATC system. It was shown in previous work [6, 7] that free flight with airborne separation is able to handle a higher traffic density. Besides, free flight can also bring fuel and time efficiency [8]. In a free flight framework, it is implied that aircraft will be responsible for their own separation assurance and conflict resolution. The loss of an airway structure may make the process of detecting and resolving conflicts between aircraft more complex. However, previous study [9] shows that free flight is potentially feasible because of enabling technologies such as Global Positioning Systems (GPS), data link communications like Automatic Dependence Surveillance-Broadcast (ADSB) [10], Traffic Alert and Collision Avoidance Systems (TCAS) [11], and powerful on board computation. Also, automated conflict detection and resolution tools [12] will be required to aid pilots and/or ground controllers in ensuring traffic separation and conflict resolution.

In this paper, a computational guidance algorithm with collision avoidance capability is proposed using Markov Decision Processes (MDPs), where the input of this algorithm is the position of other obstacles such as aircraft, and the position of one or more destinations. Through on board sensed information of other obstacles or aircraft, the algorithm will perform online sequential decision making to select actions in real-time with on board avionics. The series of actions will generate a trajectory which can guide the aircraft to quickly reach its goal and avoid potential conflicts. The algorithm operates efficiently and can fully recompute its guidance to support online replanning in the presence of dynamically changing obstacles. The proposed algorithm provides a potential solution framework to enable autonomous on-demand free flight operations in urban air mobility.

I. Related Work

There have been many important contributions to the topic of guidance algorithms with collision avoidance capability for small unmanned aerial aircraft which can be roughly categorized based on the following criteria:

- Centralized/Decentralized [13]: whether the problem is solved by a central supervising controller (centralized) or by each aircraft individually (decentralized).
- Planning/Reacting [14]: The planned approach generates feasible paths ahead of time; whereas the reactive approach typically uses an online collision avoidance system to respond to dangerous situations.
- Cooperative/Non-cooperative: whether there exists online communication between aircraft or between aircraft and the central controller.

In centralized methods, the conflicts between aircraft are resolved by a central supervising controller. Under such scenario, the state of each aircraft, the obstacle information, the trajectory constraint as well as the terminal condition are known to the central controller (thus centralized methods are always cooperative), and the central controller in return designs the individual whole trajectory for all aircraft before the flight, typically by formulating it to an optimal control problem. These methods can be based on semidefinite programming [15], nonlinear programming [16, 17], mixed integer linear programming [18–21], mixed integer quadratic programming [22], sequential convex programming [23, 24], second-order cone programming [25], evolutionary techniques [26, 27], and particle swarm optimization [28]. Besides formulating this problem using optimal control framework, roadmap methods such as visibility graph [29] and Voronoi diagrams [30] can also handle the path planning problem for aircraft. However, calculating the exact solutions will become impractical when the state space becomes large or high-dimensional. To address this issue, sample-based planning algorithms are proposed, such as probabilistic roadmaps [31], RRT [32], and RRT* [33]. These centralized methods often pursue the global optimality of the solution. However, as the number of aircraft grows, the computation time of these methods typically scales exponentially. Moreover, these centralized planning approaches typically need to be re-run, as new information in the environment is updated (e.g. a new aircraft enters the airspace).

On the other hand, decentralized methods scale better with respect to the number of agents and are more robust since they do not possess a single point of failure [34]. In decentralized methods, conflicts are resolved by each aircraft individually. Decentralized methods can be cooperative and non-cooperative. Researchers have proposed several algorithms under the case where the communication between aircraft can be successfully established (cooperative) [35]. Algorithms in [36, 37] are based on message-passing schemes, which resolve local (e.g. pairwise) conflicts without needing to form a joint optimization problem between all members of the team. In [13], every agent is allotted a time slot in which to compute a dynamically feasible and guaranteed collision-free path using MILP. In [38], the author recast the global optimization problem as several local problems, which are then iteratively solved by the agents in a decentralized way. In Decentralized Model Predictive Control approach [39], the aircraft solve their own sub-problem one after the other and send the action to other subsystems through communication.

Model Predictive Control [40, 41] can be used to solve collision avoidance problem but the computation load is relatively high. Potential field method [42, 43] is computationally fast, but in general they provide no guarantees of collision avoidance. Machine learning and reinforcement learning based algorithms [44–47] have promising performance, but usually need a lot of time to train. Monte Carlo Tree Search algorithm [48] does not need time to train before the flight and it can finish in any predefined computation time, but the aircraft can only adopt several discretized actions at each time step. A geometric approach [49–52] can be also applied for the collision avoidance problem and the computation time only grows linearly as the number of aircraft increases. DAIDALUS (Detect and Avoid Alerting Logic for Unmanned Systems) [53] is another geometric approach developed by NASA. The core logic of DAIDALUS consists of: (1) definition of self-separation threshold (SST) and well-clear violation volume (WCV), (2) algorithms for determining if there exists potential conflict between aircraft pairs within a given lookahead time, and (3) maneuver guidance and alerting logic. The drawback of these geometric approaches is that it can not look ahead for more than one step (it only pays attention to the current action and does not take account of the effect of subsequent actions) and the outcome can be local optimal in the view of the global trajectory.

Our proposed algorithm is an alternative to previous work [48], where instead of using a Monte Carlo Tree Search algorithm, we propose a novel online method for solving a subclass of MDPs with very efficient performance for problems with sparse rewards.

There are many well known methods for solving MDPs including value iteration and policy iteration, which are iterative methods based on the dynamic programming approach proposed by Bellman [54]. These algorithms use a table-based approach to represent the state-action space exactly and iteratively converge to the optimal policy π^* and corresponding value function V^*. These table-based methods have a well known disadvantage that they quickly become

intractable. As the number of states and actions increases in number or dimension, the number of entries in the table increases exponentially. Many real world problems quickly exhaust the resources of even high performing computers due to the well known 'curse of dimensionality' [54].

Many attempts have been made to allow MDPs to scale to larger problems. Factored MDPs [55, 56] attempt to alleviate the problem of state space explosion by identifying subsets of the MDP that can be broken into smaller problems. Approximation methods under the general umbrella of Approximate Dynamic Programming have been used as a compromise to obtain reasonable approximations of the underlying true value function in cases where the state-action space (or the transition matrix T) is too large to represent with traditional exact methods, which are summarized by the [57, 58]. Notably, linear function approximation methods such as GradientTD methods [59–61], statistical evaluation methods such as Monte Carlo Tree Search [62], and non-linear function approximation methods such as TDC with non-linear function approximation [63] and DQN [64] are good examples of some of the approaches taken using approximation.

Against this backdrop of extensive research, our contribution is as follows:

1) Definition of MDP as a graph and mathematical description of reward propagation through the graph from a single isolated reward source.
2) Mathematical description of how positive reward sources interact and interfere with one another while propagating through the graph.
3) Mathematical description of how to construct the final value function from the individual positive reward sources' value functions.
4) A novel algorithm to determine the solution of this MDP that has no dependency on the size of the state space.
5) Discussion of negative reward sources and the difficulty they pose in this solution framework.
6) A proposed method to model obstacles as negative rewards that allows positive and negative rewards to be separated and solved individually, along with a method to combine them back into the full MDP.
7) Demonstration of how this approach can be used to solve the real world problem of UAV path planning in an Urban Air Mobility context.

The organization of the paper is as follows. Section II provides a background of MDPs for readers unfamiliar with the topic. Section III describes an efficient method used to solve MDPs with positive rewards and Section IV lists the corresponding algorithm. Section V describes a way to use the algorithm for negative rewards as well. Section VI shows how to model the UAV path planning problem using this framework and provides the performance results.

II. Background

Markov Decision Processes (MDPs) are a framework for decision making with broad applications to finance, robotics, operations research and many other domains [65]. MDPs are formulated as the tuple (s_t, a_t, r_t, 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 is the reward received by the agent as a result of taking the action a_t from s_t and arriving at s_{t+1}, and $T(s_t, a, s_{t+1})$ is a transition function that describes the dynamics of the environment and capture the probability $p(s_{t+1}|s_t, a_t)$ of transitioning to a state s_{t+1} given the action a_t taken from state s_t.

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

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

Though normally not used in the literature, we refer to the path taken through the state space as a result of following the optimal policy as the *optimal trajectory*. We define the UAV planning problem as finding the optimal trajectory through the space such that the UAV maximizes its future expected reward.

III. Method

We first describe a very efficient method for calculating an MDP containing only positive rewards. We then describe how negative rewards can be incorporated into this method to yield an efficient method that can solve MDPs containing both positive and negative rewards.

A. Positive Reward

Our method relies on an interpretation of an MDP as a graph, rather than as a tree as is normally considered. A *transition graph* for a deterministic MDP can be defined where each node of the graph is a state $s \in S$ and each edge of the graph is a possible action $a \in A$. The transition graph is a directed graph which may be cyclic. (Note that this is just a graphical representation of the information contained in the transition function T.) This interpretation was known at least as far back as 1987 in [66] but is not usually used in the MDP literature.

We consider a special case of MDPs where the reward function is very sparse compared to the number of states in the state space. In the case of UAVs or UAM, these sparse rewards may correspond to reaching a waypoint (positive reward) or colliding with an obstacle (negative reward). We assume that the number of states is very large because either (1) the area covered by the state space is very large, or (2) the granularity of how the state space maps to the geography is very fine (perhaps even continuous).

To reduce ambiguity, when we refer to a sequence of states in time but do not care what states in particular we are referring to (as in, "from the first to the n-th state") we will use a subscript as in $\{s_1, s_2, \cdots, s_n\}$. When we refer to a specific state located somewhere in the state space (as in, "from the start state $s^{(i)}$ to the goal state $s^{(g)}$"), we will use letters and a superscript $\{s^{(i)}, \cdots, s^{(g)}\}$. Though we will try to avoid it, we could show an entire path with both time and location with $\{s_1^{(a)}, s_2^{(b)}, s_3^{(c)}\}$ which would indicate that we progressed from $s^{(a)}$ to $s^{(b)}$ and finally to $s^{(c)}$ where $s^{(a)}, s^{(b)}, s^{(c)} \in S$. When the superscript and subscripts are dropped, then the state refers to a location state generically as in $s \in S$. We will use similar notation for actions and rewards as needed.

Definition 1. *For a deterministic transition graph, the* distance δ *of two states $s^{(i)}$ and $s^{(k)}$ is defined as the minimum positive number of actions (or transitions) needed to move from a given state $s^{(i)}$ to a desired state $s^{(k)}$:*

$$\delta(s^{(i)}, s^{(k)}) = \min_t \{t | T(s^{(i)}, a_1, a_2, \cdots, a_t) = s^{(k)}\}. \tag{1}$$

Note that in this definition, $T(s^{(i)}, a_1, a_2, \cdots, a_t)$ is defined to be the final state we reach starting from state $s^{(i)}$ while sequentially taking actions $a_1, a_2, \cdots, a_t$.

If we consider an arbitrary value function, it will contain local maximas (which can envisioned as peaks) and local minimas (which can be envisioned as wells) that correspond to the value of being in a particular state. The peaks and wells exist due to the presence of rewards. The magnitude of each peak is driven by the magnitude of the reward, and the shape of the peak is driven by the exponential decay curve defined by the discount factor. Let us consider an MDP with discount factor $0 < \gamma < 1$, initial state s and positive reward $r^{(g)}$ located at state $s^{(g)}$, and distance through the transition graph $\delta(s, s^{(g)})$. If the state $s^{(g)}$ is a terminating state, the value function is:

$$V(s) = \gamma^{\delta(s, s^{(g)})} \cdot r^{(g)}, \quad \forall s \in S. \tag{2}$$

However, if the state $s^{(g)}$ is non-terminating, then we must consider that after collecting the reward $r^{(g)}$, the agent will pass through $s^{(g)}$ and then attempt to "circle back" to collect $r^{(g)}$ again and again. We refer to this process as a *cycle*:

Definition 2. *The* cycle *of a state s, which denoted as $C(s)$, is an ordered sequence of states: $s_1, s_2, \cdots, s_t$ resulting from actions $a_1, a_2, \cdots, a_{t+1}$ that result in returning to the same state s:*

$$T(s, a_1, a_2, \cdots, a_t, a_{t+1}) = s.$$

The length of the cycle, $d\,[C(s)]$ is the number of actions in the sequence $(t+1)$ that causes a return to s.

We refer to the cycle(s) with shortest length as a *minimum cycle(s)*:

Definition 3. *Suppose a state s has p cycles $C^1(s), \cdots, C^p(s)$, where p can be finite or infinite. The* minimum cycle *of state s, which denoted as $C^*(s)$, is a cycle with minimum distance. We denote the distance of the minimum cycle of state s as $\phi(s)$.*

$$C^*(s) = \left\{ C^i(s) \mid d\left[C^i(s)\right] \leq d\left[C^j(s)\right], \ \ \forall j \in \{1, \cdots, p\} \right\}. \tag{3}$$

Given these definitions, we can then compute the value function of a single non-terminating reward as:

$$V(s) = \gamma^{\delta(s, s^{(g)})} \cdot \frac{r^{(g)}}{1 - \gamma^{\phi(s^{(g)})}}, \ \ \forall s \in S \tag{4}$$

When multiple rewards are present, however, this complicates the value function as execution continues after collecting a reward. If it so happens that after collecting a reward, an additional reward is collected then this alters the resulting value function. In short, in certain cases the "peaks" resulting from the different rewards can interact with each other, especially when one reward, a *secondary reward* lies within the minimum cycle of a larger reward, the *primary reward*.

We account for this complexity by defining two types of peaks:

- *Baseline peak:* Value function resulting from collecting non-terminating reward which does not contain any other reward sources within its minimum cycle.
- *Combined peak:* Value function resulting from collecting a non-terminating primary reward which when collected also collects one or more secondary non-terminating reward sources when following the primary reward's minimum cycle.

The baseline peak and combined peak defined above correspond to rewards that are collected "infinitely", meaning that if the agent were to run forever, it would collect the corresponding rewards an infinite number of times.

We further note one additional source of complexity for non-terminating rewards. Some rewards are so large that they cause other rewards to be ignored. This occurs when a large reward is so attractive that it dominates the decision of which action is most valuable to take from the current state. This is a natural byproduct of MDPs balancing long term and short term reward. We then define a third type of peak:

- *Delta peak:* Value function resulting from collecting a non-terminating secondary reward which is collected at most once as the agent follows the optimal trajectory to a primary reward which forms a baseline or combined peak.

Formally, we define a propagation operator $\mathcal{P}$. If we assume that there is a peak value $v^{(g)}$ at state $s^{(g)}$, then we will term the operation of calculating the whole value function from a peak as *propagating reward* and will denote this for reward $r^{(g)}$ at state $s^{(g)}$ generally as:

$$\mathcal{P}_g(s) = \gamma^{\delta(s, s^{(g)})} \cdot v^{(g)}, \ \ \forall s \in S \tag{5}$$

where $\delta(s, s^{(g)})$ is the distance from s to $s^{(g)}$. Note that this simply corresponds to the discounted future reward from state s with respect to reward $r^{(g)}$, but is a convenient notational shorthand.

Definition 4. *We use the term* baseline peak*, $\mathcal{B}^i$, to describe a single reward source $r^{(i)}$ located at state $s^{(i)}$ which is collected infinitely but has no other rewards in its minimum cycle. When the context is clear or we are speaking generally of a baseline peak, we may drop the i and denote the baseline peak as $\mathcal{B}$. The value at the baseline peak is:*

$$\mathcal{B}^i = \frac{r^{(i)}}{1 - \gamma^{\phi(s^{(i)})}}$$

The value function for the baseline peak $\mathcal{B}^i$ is a mapping $\mathcal{P} : \mathcal{S} \to \mathbb{R}$:

$$\mathcal{P}_{\mathcal{B}^i}(s) = \gamma^{\delta(s, s^{(i)})} \cdot \frac{r^{(i)}}{1 - \gamma^{\phi(s^{(i)})}} \tag{6}$$

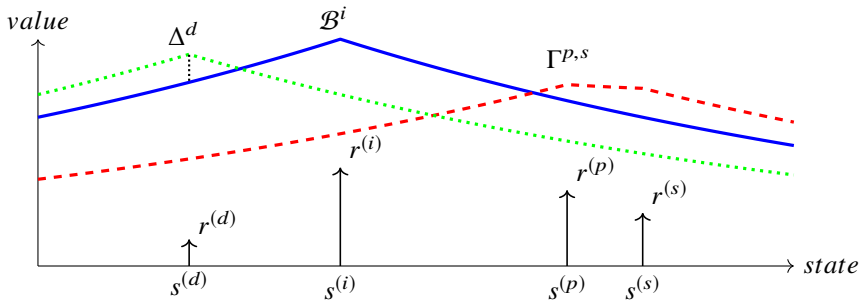


Fig. 1 Illustration of baseline (blue, solid), combined baseline (red,dashed), and delta baseline (green, dotted) in a 1-dimensional state space.

Definition 5. *We use the term* combined peak *to describe a primary reward source at state $s^{(p)}$ which is collected infinitely and has a secondary reward source at state $s^{(s)}$ within the primary state's minimum cycle. We denote the value of the combined peak at $s^{(p)}$ as $\Gamma^{p,s}$ (or Γ^p or even Γ when the context is clear.) The value function for a combined peak is:*

$$\mathcal{P}_{\Gamma^{p,s}}(s) = \mathcal{P}_{\mathcal{B}^p}(s) + \mathcal{P}_{\mathcal{B}^s}(s) \tag{7}$$

Definition 6. *A delta peak for a reward $r^{(i)}$ is calculated by adding the reward $r^{(i)}$ at state $s^{(i)}$ to some pre-existing value function $\mathcal{P}^j(s)$ formed by propagation. At $s^{(i)}$, the value of the delta peak is $\Delta^i = r^{(i)} + \mathcal{P}^j(s^{(i)})$. The value function for the delta peak is formed by propagation:*

$$\mathcal{P}_{\Delta^i}(s) = \gamma^{\delta(s,s^{(i)})} \cdot \Delta^i \tag{8}$$

With those definitions, we return now to constructing the value function from the peaks. The optimal value function in an MDP can be understood as the maximum possible value at each state. To construct the value function we need only take the *max* over all of the value functions formed by the baseline, combined, and delta peaks. While this set explodes combinatorially with the number of reward sources, we can efficiently prune this set of possibilities by ordering the peaks by value and processing them in order from greatest to smallest. Note that in order to efficiently process the delta functions, we also compute only the subset of them that are realizable at each iteration of the algorithm.

See [67] for proofs and expanded explanation.

IV. Algorithm

We present a variant of [67], `Memoryless` , which computes the value function from a list of peaks that have been processed at previous iterations. `Exact` from [67] is faster but has a dependency on the number of states $|S|$, whereas `Memoryless` is somewhat slower but frees the algorithm from any dependency on the size of the state space $|S|$.

The `Memoryless` algorithm computes the neighboring state values on demand from a list of the peaks sorted by order in which they were processed by the algorithm. During each iteration of the algorithm, the value of any required states are calculated as needed; the algorithm therefore need not compute the entire intermediate value function. This results in slower overall performance especially as the number of reward source grows, but if the number of reward sources is kept manageable, then the solution to the MDP can be computed efficiently and exactly without having a time or space dependence on the number of states $|S|$.

We now discuss the methodology for calculating the distance between two states. For an arbitrary graph, computing the shortest path through the graph is $O(V \log V + E)$ where V is the number of vertices and E is the number of edges. However, in our flight planning problem formulation (i.e., when the dimensions of the state space S map to an underlying metric space) we can use special knowledge of the state space to directly compute distances with a metric such as Euclidean distance to compute shortest paths in constant time. Because of this special circumstance, we permit ourselves to omit the cost of searching a general graph from the run time of the algorithm and note that it increases by a factor of $O(V \log V + E)$ if a general graph search with Dijkstra's algorithm is used. Note that all complexity factors shown below assume this constant time Euclidean distance metric.

The algorithm uses a heap-based priority queue which takes $O(\log N)$ for insertion and deletion which we use to

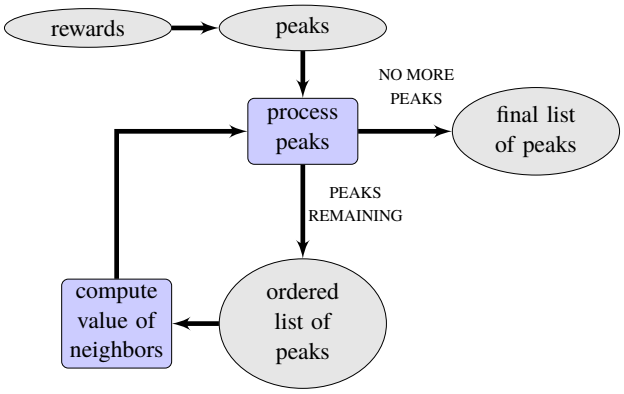
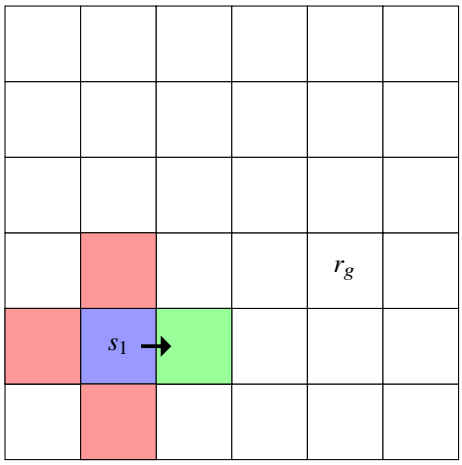
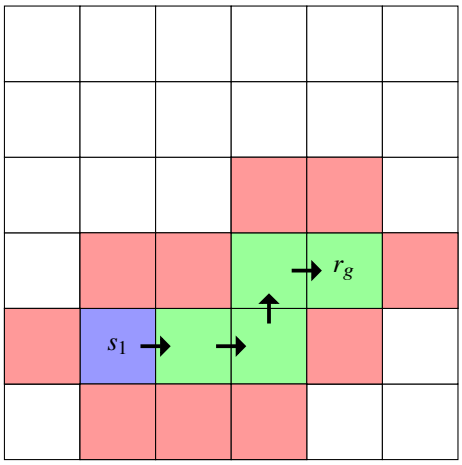


Fig. 2 **`Memoryless`** **continues processing until no more peaks remain to be processed and arrives at a data structure that can be used to determine the value function of the MDP. In the** **`Memoryless`** **algorithm, we maintain a list of the peaks and compute the value of states on demand. The** **`Exact`** **algorithm from [67] therefore has a similar limitation to value iteration in that the entire state space must fit into memory. The** **`Memoryless`** **algorithm has no such dependency and can in theory represent even a continuous state space (with infinite states).**



(a) Neighbors' values from initial state.



(b) Neighbors' values along entire path.

Fig. 3 **Illustration of** **`Memoryless`** **algorithm calculating neighboring states on-demand as it follows the optimal policy. The optimal neighbor is shown in green, and the sub-optimal neighbors are shown in red. The initial state is shown in blue and labeled s_1. State containing reward labeled r_g. The optimal policy is shown with arrows. The optimal path is followed by computing the value of only a subset states, where un-colored states are not computed at all. When the number of states $|S|$ is very large, the number of on-demand computations can be very small compared to the total number of states.**

keep lists of peaks which are of order $|R|$. Given that we assume a small number for $|R|$, we assume the insertion and deletion times are negligible compared to the run time of the algorithm.

Algorithm 1 `Memoryless`

```
1: procedure MEMORYLESS (rewardSources)
2:     processedPeaks ← empty list
3:     sortedPeaks ← PrecomputePeaks( rewardSources )
4:     while sortedPeaks is not empty do
5:         deltaPeaks ← ComputeDeltas( processedPeaks )
6:         sortedPeaks ← PruneInvalidPeaks( processedPeaks)
7:         maxPeak ← max( [ sortedPeaks, deltaPeaks ] )
8:         sortedPeaks ← RemoveAffectedPeaks( maxPeak )
       return processedPeaks *
```

$O(|R|^3 \times |A|^2)$ *time,* $O(|R|x|A|)$ *space complexity.*

```
1: procedure PRECOMPUTEPEAKS(rewardSources)
2:     list ← empty SortedList
3:     for all rewardSources do
4:         list.add( baseline peak for reward source )
5:     for all rewardSources do
6:         nbr ← find neighboring state with highest reward
7:         if nbr is not empty then
8:             list.add( cycle peak for reward source )
       return list
```

$O(|R| \times |A|)$ *time. Note returned list in pathological worst case is* $O(|R| \times |A|)$ *in length.*

```
1: procedure COMPUTEDELTAS(processedPeaks *)
2:     list ← empty SortedList
3:     for all reward sources do
4:         currentValue = ValueOnDemand( processedPeaks )
5:         compute delta of reward and currentValue
6:         nbr ← find neighboring state with highest value *
7:                 using ValueOnDemand *
8:         list.add(max(deltapeak, neighborvalue))
```

$O(|R|^2 \times |A|)$ *time complexity.*

```
1: procedure REMOVEAFFECTEDPEAKS(list, state)
2:     for all remaining peaks do
3:         if peak is affected by state then
4:             list.remove( peak )
```

$O(|R|)$ *time complexity.*

```
1: procedure PRUNEINVALIDPEAKS
2:     for all remaining peaks do
3:         nbr ← find neighboring state with highest value *
4:                 using ValueOnDemand *
5:         if nbr > peak then
6:             list.remove( peak )
```

$O(|R|^2 \times |A|)$ *time complexity.*

```
1: procedure VALUEONDEMAND(previousPeaks, desiredState)
2:     maxValue ← MIN_FLOAT
3:     for all previousPeaks do
4:         priValue ← pri_value × γ^φ(desiredState,priState)
5:         secValue ← sec_value × γ^φ(desiredState,secState)
6:         maxValue ← max(maxValue, priValue, secValue)
       return maxValue
```

$O(|R|)$ *time complexity.*

The `Memoryless` function is $O(|R|^3 \times |A|^2)$. Memory complexity for the algorithm is $O(|R| \times |A|)$. Note here there is no dependence upon the size of the state space $|S|$.

A. Extracting Optimal Trajectory

It is trivial to follow the optimal policy of a solved MDP. Given the current state s, we use the value function to determine which action is most valuable and then take that action. If the optimal policy is computed, it will always follow the optimal trajectory. This means that for our problem, aircraft does not need to compute a full trajectory at each time step; instead, it need only compute the next step it must take at each time step and over time this will result in the optimal trajectory.

If however, we wish to compute the trajectory, say to visualize the flight plan or to submit it to a central flight planning authority such as UTM, we present a simple algorithm that allows the trajectory to be extracted from the policy. This relies on an $ExecuteAction$ module that can simulate one time step forward into the future. Note again that this need not be on board the aircraft for the aircraft to follow the optimal policy and safely execute its flight.

Algorithm 2 FollowLocalPolicy

```
1: procedure FollowLocalPolicy(processedPeaks, initialState)
2:     currState ← initialState
3:     while True do
4:         neighbor ← FindMaxNeighbor( processedPeaks, currState )
5:         action ← DetermineAction( currState, neighbor)
6:         currState ← ExecuteAction( action )
```

V. Negative Reward

For the problem of UAV collision avoidance, intuitively we were looking for a method to model the obstacles as risks that should be avoided. Being close to an obstacle should be riskier than being further away from an obstacle, and beyond a certain threshold, an obstacle presents effectively no risk and can be safely ignored.

Through our investigation of negative rewards in an MDP formulation, we found that modeling negative rewards as a single point was ineffective as this only materially affected states in the very immediate vicinity of the negative reward. We found that to model our obstacles as a risk that decreased with distance to the obstacle, we had to manually construct a reward function with many negative rewards that explicitly encoded the risk. This explicit construction of the risk may take hundred or thousands of negative rewards in the MDP, which would not lead to good performance.

We instead came up with a method to obtain the desired negative reward shape in a way which allowed us to reuse the efficient `Memoryless` algorithm.

A. Standard Positive Form

Given the new understanding of how value functions are composed from reward, we determined that is is possible to separate out the positive and negative rewards into two separate MDPs, solve them independently, and then aggregate the results together to recover a very close approximation of the original MDP's value function.

With the negative rewards isolated into a separate MDP, all of the rewards can be negated so that they are all positive. We will refer to an MDP with negative rewards that has been converted to this positive form as *standard positive form*. The `Memoryless` algorithm can then compute the resulting value function for the MDP in standard positive form. If the resulting MDP's value function is then negated, then this very closely approximates the result obtained by value iteration for the negative rewards. If the resulting value functions are added together, we then arrive back at a close approximation of the original value iteration solution obtained from all of the positive and negative rewards together.

Furthermore, given any set of rewards in standard positive form, we can decompose them as desired into subsets and solve these subsets individually. After solving them, we can take the max over the resulting value functions to obtain the same value function as if we had solved the MDP with all of these rewards in standard positive form. This ability to freely decompose the MDP into smaller subsets has important computational advantages in the problem formulation below.

VI. UAV path planning and obstacle avoidance

Returning now to our UAV path planning and obstacle avoidance problem, we will now demonstrate how this all comes together to form an efficient but effective path planner.

For our UAV problem in this paper, we will assume that our UAVs operate effectively in a 2D plane which will maximize potential conflicts and require all corrections to be performed laterally.

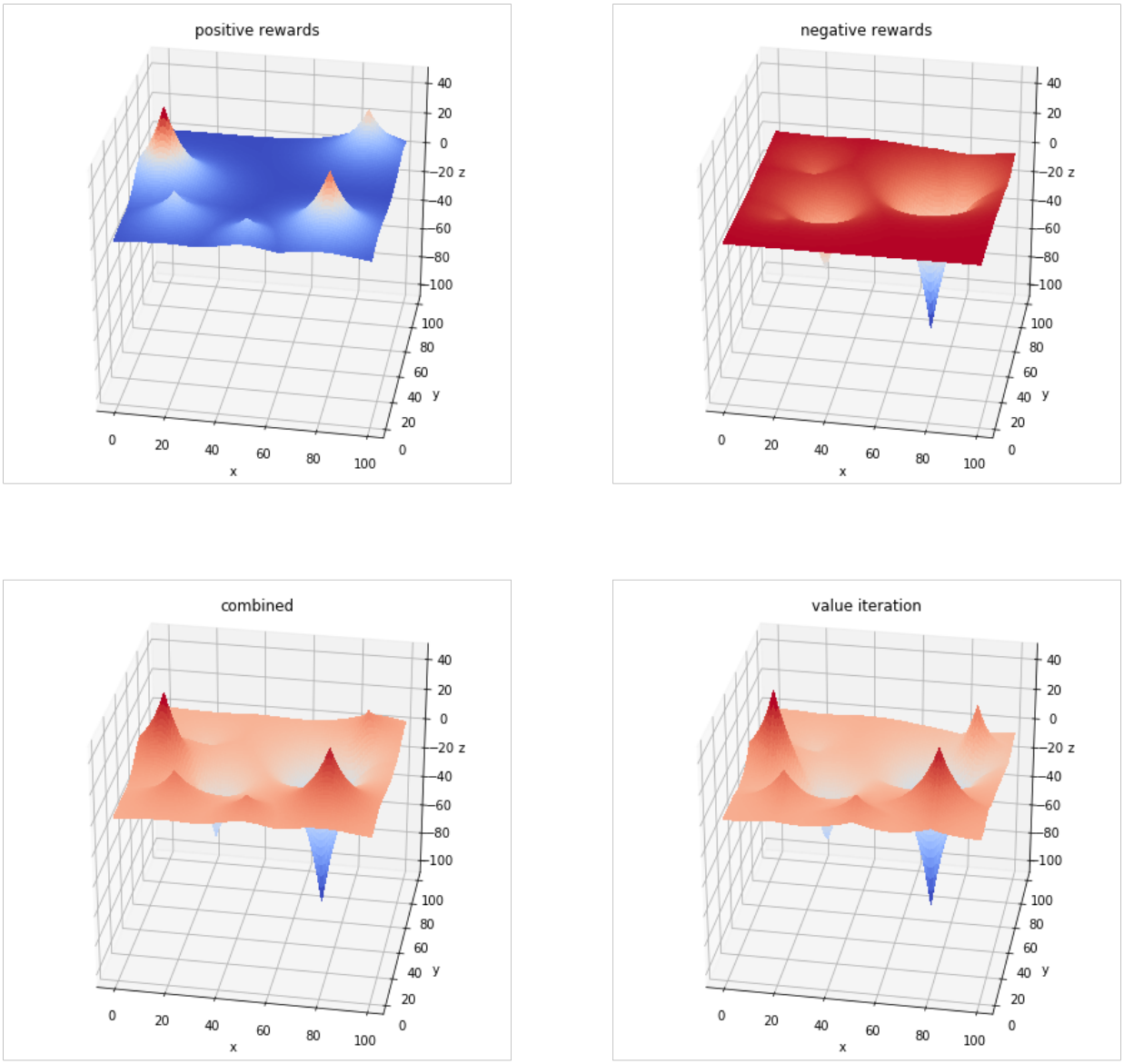


Fig. 4 Separating MDPs into positive and negative rewards into sub problems, reassembling the results, and comparison to value iteration results.

A. MDP Formulation

1. State Space

We define the environment in which the UAV operates as a $24km \times 24km$ square area in which there is a goal and a configurable number of intruders. We discretize the MDP state space into an 800x800 grid of states.

The state includes all the information the ownship needs for its decision making: the position, heading, and velocity of the ownship, the goal position, and each intruder(s) position, heading and velocity. The ownship position (o_x, o_y), heading o_θ and velocity o_{v_x}, o_{v_y}, the goal position (g_x, g_y), and for each intruder $\forall k \in K$, the position $(i_{k,x}, i_{k,y})$, heading $i_{k,\theta}$, and velocity i_{k,v_x}, i_{k,v_y} are all concatenated into one long vector.

$$s = [o_x, o_y, o_\theta, o_{v_x}, o_{v_y}, g_x, g_y, i_{1,x}, i_{1,y}, i_{1,\theta}, i_{1,v_x}, i_{1,v_y}, \cdots, i_{m,y}, i_{m,\theta}, i_{m,v_x}, i_{m,v_y}], \tag{9}$$

where m represents the number of intruders.

2. Action Space

The set of possible actions that can be taken are heading commands from $0, \cdots, 2\pi$ in steps of $\frac{\pi}{12}$.

$$A = \{0, \frac{\pi}{12}, \frac{2\pi}{12}, \frac{3\pi}{12}, \cdots, \frac{23\pi}{12}\}. \tag{10}$$

3. Dynamic Model

The ownship kinematic model is:

$$\dot{x} = v \cos\theta \tag{11}$$

$$\dot{y} = v \sin\theta, \tag{12}$$

where $v = \sqrt{v_x^2 + v_y^2}$ is the speed of the aircraft.

The ownship speed v is fixed at $50m/s$. At each step the ownship is restricted to performing a change in heading of $\dot{\theta}$ of up to $\pm 15°$.

4. Reward Function

We model the goal as a positive reward of 100. To model the UAV risk, we define a “risk well" as a negative reward of -500 which decays at a rate of 0.96 for up to 1500 meters from the center of the well, after which there is no negative reward.

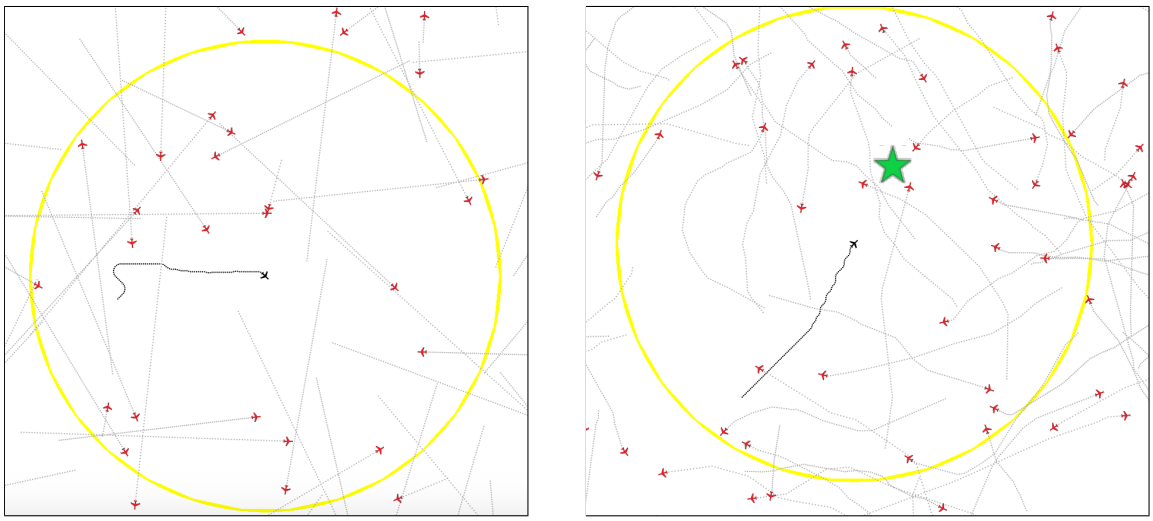
For each intruder, we place a risk well at the location the intruder will be in 2 seconds and a second risk well at the location the intruder will be in 4 seconds. This assumes that the intruders will maintain a constant heading and velocity and is used as a way to model the risk over the next 4 seconds.

For the overall MDP containing all rewards, we use a discount factor of 0.999. This provides a strong attraction to the goal globally over the state space. We found in early experiments that in an MDP the impact of negative rewards remains relatively isolated to the state where the negative reward occurs. Thus the negative rewards we place in the space are largely unaffected by the discount factor.

With a normal MDP formulation, we would need to insert many hundreds or thousands of individual negative rewards to model the risk wells for each intruder. With the algorithm we present in this paper, we instead construct an MDP in standard positive form and represent the risk wells as a single negative reward of 500 with a discount factor of 0.96. Each intruder receives its own MDP, and as there are two risk wells per intruder, there are two rewards of 500 in each intruder's MDP.

B. Computing Value Function

We define a radius of $6km$ around the ownship that defines the radius of consideration of intruders. Only intruders within this radius of the ownship will be modeled in the problem and all other intruders will be ignored.



(a) Deterministic intruders **(b) Stochastic intruders**

Fig. 5 Experimental results showing deterministic and stochastic intruders. Deterministic intruders are spawned in random locations with random heading and velocities (within predefined limits), but during flight they maintain constant heading and airspeed. Stochastic intruders are spawned identically, but there is a small probability that they will change their heading by up to ±25° at each time step making it very difficult to predict their future position with any certainty. Ownship is in black, intruders are in red, and goal is a green star. Light shaded paths are intruder past trajectories, and the dark shaded path is ownship past trajectory. The yellow circle illustrates the boundary beyond which intruders will be ignored.

To compute the resulting value at any state, we follow the following procedure to compute the value at a state s:

1) Solve the MDP for the goal
2) Solve the MDPs for each of the intruders in standard positive form
3) Extract the value from the positive reward MDP:

$$v_g = V_g^*(s) \tag{13}$$

where V_g^* indicates the value at state s calculated from the solution of the MDP.

4) Compute the worst case penalty from the intruders in standard positive form:

$$p_p = \max_{i \in I} V_i^*(s) \tag{14}$$

where V_i^* indicates the value at state s calculated from the solution of the MDP corresponding to intruder i.

5) Convert the penalty in standard positive form back to a negative value.

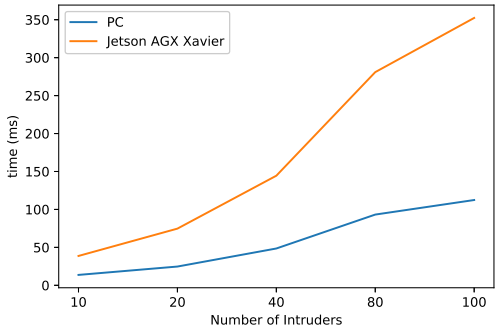
$$p_n = -p_p \tag{15}$$

6) Add the positive and negative penalties together to obtain the true value of the value function.

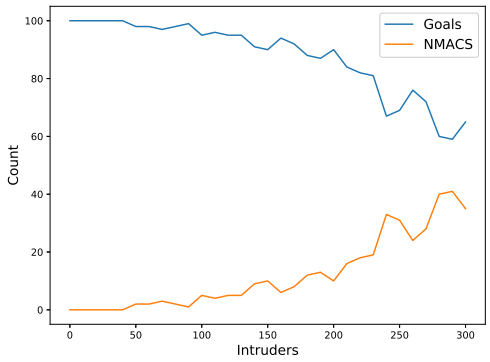
$$v_t = v_g + p_n \tag{16}$$

We demonstrate this planner in a 2D aircraft simulation showing an overhead view of the ownship, the goal, and the intruders as shown in Figure 5.

The intruders are driven by a simple policy, which may either be deterministic or stochastic during an experiment. Intruders are spawned randomly in the space. Deterministic intruders maintain heading and airspeed during their flight. Stochastic intruders have a small probability of randomly changing their heading up to ±25° at each time step. In either case, if an intruder reaches the boundary it is re-spawned in a new random location. A new MDP is created and solved at each time step allowing for dynamically changing obstacles.

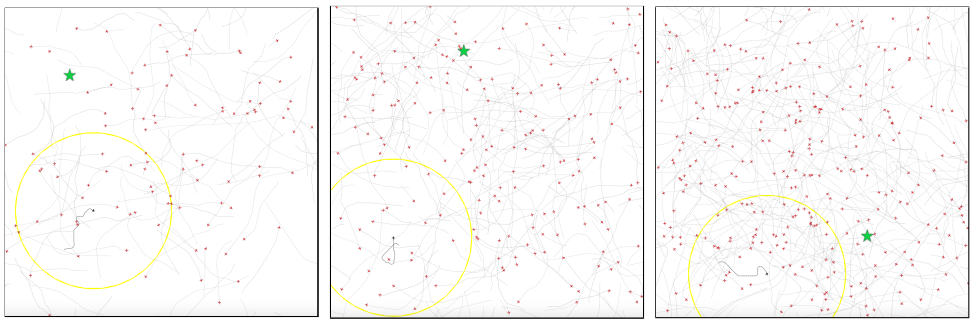


(a) Timing performance as number of intruders increases



(b) Collision avoidance performance as intruder density increases

Fig. 6 Experimental results showing the performance of the algorithm. (a) shows time to compute the solution as the number of intruders increases is roughly $O(m)$ where m is then number of intruders. (b) shows the ability to reach the goal and the number of near midair collisions (NMACs) as the number of randomly turning intruders in the space increases. Note that as the airspace becomes more crowded, at some point it becomes nearly impossible to make it through the waves of intruders. Also, there may be situations where the random position of the intruders leaves no feasible path for collision avoidance.



(a) 100 intruders **(b) 200 intruders** **(c) 300 intruders**

Fig. 7 Visualization of different number of intruders to illustrate the difficulty of the collision avoidance problem.

As shown in Figure 6a, decomposing the problem into very small MDPs with a fixed number of rewards makes each of the sub-MDPs a constant-time operation yielding an overall linear $O(m)$ performance where m is the number of intruder aircraft. For Figure 6a the code was run in a mode where it considered only a specific number of aircraft. Low level performance timers were used to record start and stop times of the algorithm's key processing phases: decoding observations, solving all of the MDPs, and computing the action from all of the MDPs' solutions. These times were summed into a value that captures the amount of time the algorithm runs each cycle within the overall simulation. The simulation ran for approx 1000 iterations to account for any variation. The mean of these iterations is plotted in Figure 6a.

Timing tests were run on two computers. First a PC with a 2.8 GHz Intel i7 CPU. The second platform was an ARM based NVIDIA AGX Xavier board running in MAXN (30W + mode) with jetson_clocks.sh run to maximize clock speeds. Both tests were in python, single-threaded with no special hardware assistance such as GPUs or "hidden" computational libraries such as numba. Numpy is used by the algorithm.

In Figure 6b, we instead study the agent's ability to avoid near midair collisions (NMACs) as we increase the number of intruders in the state space. For this measurement, we allow the agent to run for up to 10,000 steps. The state space is 800×800, so this provide ample ability for the agent to reach the goal even with extreme collision avoidance, but also prevents an infinite run which never terminates because it is infeasible to reach the goal. For each number of intruders, we run 100 episodes to determine how many times we reach the goal. We also record how many episodes result in a near midair collision (NMAC), which we define as coming within 150 meters of an intruder at any point during the episode. If an NMAC is detected, then the episode is terminated. Thus we should expect that as the number of NMACs grow, we should also see the number of goals reached reduce by an equal amount. The intruders in this experiment were the stochastic ones which at each time stamp have a small probability of changing their heading by $\pm 25^\circ$. This results in a very unpredictable and challenging environment for the aircraft to maneuver within, especially when the number of intruders increases.

Sample videos showing the algorithm in action are available at: **https://youtu.be/NWI8T-SgHcU**

VII. Conclusion

In this paper, we have presented a novel computational guidance algorithm for flight planning for Unmanned Aerial Mobility (UAM) based on Markov Decision Processes. We present `Memoryless` , an efficient algorithm for solving MDPs that has no dependence on the size of the state space. We show how the algorithm can be used to solve a path planning and collision avoidance problem, and demonstrate that the algorithm's performance is suitable for online processing with real-time constraints. As the algorithm has no dependence on the size of the state space of the MDP, it is suitable for resource constrained embedded computing environments where memory and computation power is severely limited. In future work, we plan to integrate the algorithm into more advanced flight simulators and investigate multi-agent performance.

References

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

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

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

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

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

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

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

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

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

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

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

[12] Krozel, J., and Peters, M., "Conflict detection and resolution for free flight," *Air Traffic Control Quarterly*, Vol. 5, No. 3, 1997, pp. 181–212.

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

[14] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D., *Introduction to autonomous mobile robots*, MIT press, 2011.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[48] Yang, X., and Wei, P., "Autonomous On-Demand Free Flight Operations in Urban Air Mobility using Monte Carlo Tree Search," 2018.

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

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

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

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

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

[54] Bellman, R., *Dynamic programming*, Courier Corporation, 2013.

[55] Schuurmans, D., and Patrascu, R., "Direct value-approximation for factored MDPs," *Advances in Neural Information Processing Systems*, 2002, pp. 1579–1586.

[56] Guestrin, C., Koller, D., Parr, R., and Venkataraman, S., "Efficient solution algorithms for factored MDPs," *Journal of Artificial Intelligence Research*, Vol. 19, 2003, pp. 399–468.

[57] Bertsekas, D. P., *Dynamic programming and optimal control*, Vol. 1, Athena scientific Belmont, MA, 1995.

[58] Powell, W. B., *Approximate Dynamic Programming: Solving the curses of dimensionality*, Vol. 703, John Wiley & Sons, 2007.

[59] Sutton, R. S., Maei, H. R., and Szepesvári, C., "A Convergent O(n) temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation," *Advances in neural information processing systems*, 2009, pp. 1609–1616.

[60] Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E., "Fast gradient-descent methods for temporal-difference learning with linear function approximation," *Proceedings of the 26th Annual International Conference on Machine Learning*, ACM, 2009, pp. 993–1000.

[61] Precup, D., Sutton, R. S., and Dasgupta, S., "Off-policy temporal-difference learning with function approximation," *ICML*, 2001, pp. 417–424.

[62] Kocsis, L., and Szepesvári, C., "Bandit based monte-carlo planning," *European conference on machine learning*, Springer, 2006, pp. 282–293.

[63] Bhatnagar, S., Precup, D., Silver, D., Sutton, R. S., Maei, H. R., and Szepesvári, C., "Convergent temporal-difference learning with arbitrary smooth function approximation," *Advances in Neural Information Processing Systems*, 2009, pp. 1204–1212.

[64] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

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

[66] Papadimitriou, C. H., and Tsitsiklis, J. N., "The complexity of Markov decision processes," *Mathematics of operations research*, Vol. 12, No. 3, 1987, pp. 441–450.

[67] Bertram, J., Yang, X., and Wei, P., "Fast Online Exact Solutions for Deterministic MDPs with Sparse Rewards," *ArXiv preprint arXiv:1805.02785*, 2018.