J Intell Robot Syst (2011) 64:277–298
DOI 10.1007/s10846-010-9532-6

Aircraft Collision Avoidance Using Monte Carlo Real-Time Belief Space Search

Travis B. Wolf · Mykel J. Kochenderfer

Received: 22 July 2009 / Accepted: 21 December 2010 / Published online: 13 January 2011

Abstract The aircraft collision avoidance problem can be formulated using a decision-theoretic planning framework where the optimal behavior requires balancing the competing objectives of avoiding collision and adhering to a flight plan. Due to noise in the sensor measurements and the stochasticity of intruder state trajectories, a natural representation of the problem is as a partially-observable Markov decision process (POMDP), where the underlying state of the system is Markovian and the observations depend probabilistically on the state. Many algorithms for finding approximate solutions to POMDPs exist in the literature, but they typically require discretization of the state and observation spaces. This paper investigates the introduction of a sample-based representation of state uncertainty to an existing algorithm called Real-Time Belief Space Search (RTBSS), which leverages branch-and-bound pruning to make searching the belief space for the optimal action more efficient. The resulting algorithm, called Monte Carlo Real-Time Belief Space Search (MC-RTBSS), is demonstrated on encounter scenarios in simulation using a beacon-based surveillance system and a probabilistic intruder model derived from recorded radar data.

Keywords POMDP algorithms · Aircraft collision avoidance

This work is sponsored by the Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

T. B. Wolf · M. J. Kochenderfer (✉)
Lincoln Laboratory, Massachusetts Institute of Technology,
244 Wood Street, Lexington, MA 02420, USA
e-mail: mykelk@ll.mit.edu

T. B. Wolf
e-mail: tbwolf@mit.edu

1 Introduction

Without special prior authorization, unmanned aircraft are prohibited from flying in civil airspace. A necessary step towards routine airspace access is the ability to sense and avoid other aircraft. Because of latency and reliability issues with communicating to a ground pilot, an automated airborne collision avoidance system appears to be an important risk-mitigation measure. Unfortunately, collision avoidance systems for unmanned aircraft have not demonstrated the high level of safety required by civil aviation certification authorities.

Currently, the Traffic Alert and Collision Avoidance System (TCAS) is the only widely-deployed aircraft collision avoidance system. TCAS is required on all large transport aircraft in the world. The system estimates intruder position using radar beacon surveillance and uses linear extrapolation to predict the future state of the intruder. If the system predicts that the intruder will penetrate a predefined safety buffer, the system will issue a resolution advisory to the pilot to adjust the vertical speed of the aircraft [12].

Another collision avoidance system, the Autonomous Airborne Collision Avoidance System (Auto-ACAS), is currently used on some high-performance military aircraft. This system requires a dedicated datalink between aircraft and only engages during the last few seconds prior to a potential collision [16]. Other research in autonomous aircraft collision avoidance has investigated potential field methods [7], rapidly-expanding random trees (RRTs) [2], and mixed-integer linear programming (MILP) formulations [13].

A collision avoidance system for unmanned aircraft will require sensors, such as active radar or electro-optical/infrared cameras, capable of detecting and tracking intruders without transponders. It is also likely that the system would need a more sophisticated, probabilistic intruder projection model instead of linear projection. A better model can reduce false alerts and unnecessary flight-plan deviation while making the system more robust to maneuvering intruders.

This paper pursues a decision-theoretic planning framework that leverages knowledge of sensor characteristics and an intruder model to produce behavior that optimally balances the competing objectives of avoiding collision and adhering to a flight plan. The problem of collision avoidance is formulated as a partially-observable Markov decision process (POMDP). In a POMDP, the underlying state dynamics is Markovian and the observations depend probabilistically on the state. For our application, we represent the probabilistic model of intruder trajectories as a dynamic Bayesian network whose parameters are based on a large collection of aviation surveillance radar data. In our experiments, we use a TCAS sensor model, but other sensor models could be used just as easily. Because the underlying state is not directly observable, the system must rely upon the sensor and dynamic models to track the current belief state, which is a distribution over underlying states. A POMDP solution specifies which action to execute from the current belief state in order to maximize expected performance according to some metric.

Many algorithms for finding approximate POMDP solutions exist in the literature, but they typically require discretization of the state and observation spaces. Discretization is impractical because of the dimensionality of our collision avoidance problem, so we investigated a sample-based method to represent the uncertainty in the state, similar to what is used in particle filters. This sample-based representation

was used to extend an existing algorithm called Real-Time Belief Space Search (RTBSS) [9, 11], which uses branch-and-bound pruning to make searching the belief space for the optimal action more efficient. This paper demonstrates the resulting algorithm, called Monte Carlo Real-Time Belief Space Search (MC-RTBSS), on simulated encounter scenarios.

Section 2 provides an overview of POMDPs and introduces some approximation methods, including RTBSS. Section 3 presents MC-RTBSS and explains the algorithm. Section 4 describes an implementation in the aircraft collision avoidance problem domain. Section 5 presents simulation results. Section 6 summarizes these results and offers suggestions for further work.

2 Partially Observable Markov Decision Processes (POMDP)

A POMDP models an autonomous agent interacting with the world. The world is only "partially observable," so the agent must rely upon noisy and incomplete measurements of the state and information about its own actions to infer the state of the world and accomplish its task. Formally, a POMDP is defined by the tuple $\langle S, A, T, R, \Omega, O\rangle$, where S is the set of states of the world, A is the set of possible agent actions, T is the state-transition function, R is the reward function, Ω is the set of possible observations the agent can receive, and O is the observation function [3]. The state-transition function, $T(s, a, s')$, gives the probability that the agent will end in state s', given that it starts in state s and takes action a. The reward function, $R(s, a)$, gives the expected immediate reward the agent receives for taking action a from state s. The observation function, $O(s', a, o)$, gives the probability that the agent receives observation o after taking action a and ending in state s'.

The objective is to choose actions that maximize the expected discounted return,

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 \ldots, \tag{1}$$

where r_t is the reward at time t and $\gamma \in [0, 1)$ is the discount factor. This section explains how the agent estimates the current state from noisy sensor measurements and how to use the state estimates to choose actions that maximize expected discounted return.

2.1 State Estimation

A probability distribution b, called a belief state, is used to represent the uncertainty the agent has about the state of the world. Each time the agent takes an action and receives an observation, it updates the belief state:

$$b'(s') = P(s' \mid o, a, b) \tag{2}$$

$$= \frac{P(o \mid s', a, b)\, P(s' \mid a, b)}{P(o \mid a, b)} \tag{3}$$

$$= \frac{P(o \mid s', a) \sum_{s \in S} P(s' \mid a, b, s)\, P(s \mid a, b)}{P(o \mid a, b)} \tag{4}$$

$$= \frac{O(s', a, o) \sum_{s \in S} T(s, a, s') b(s)}{P(o \mid a, b)}. \tag{5}$$

The denominator, $P(o \mid a, b)$, serves as a normalizing factor, ensuring that b' sums to unity. The current belief state accounts for the past history of observations and the initial belief state [3].

2.2 Policy Optimization

When following a policy π, the expected discounted return from a given state s is given by the value function

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \sum_{o \in \Omega} O(s', \pi(s), o) V_\pi(s'). \tag{6}$$

An optimal policy π^* maximizes the expected discounted return from every state and has a value function denoted $V^*(s)$:

$$\pi^*(s) = \arg\max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, \pi^*(s), s') \sum_{o \in \Omega} O(s', \pi^*(s), o) V^*(s') \right] \tag{7}$$

$$V^*(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, \pi^*(s), s') \sum_{o \in \Omega} O(s', \pi^*(s), o) V^*(s') \right]. \tag{8}$$

Because the agent does not know s exactly, the reward function and value function must be evaluated over a belief state, not simply at a single state. The reward associated with a particular belief state is the expected value of the reward function:

$$R(b, a) = \sum_{s \in S} b(s) R(s, a). \tag{9}$$

The value of a belief state is then

$$V_\pi(b) = R(b, \pi(b)) + \gamma \sum_{o \in \Omega} P(o \mid b, \pi(b)) V_\pi(b'_o), \tag{10}$$

where b'_o is the future belief state weighted according to observation o. A POMDP policy, $\pi(b)$, specifies an action to take while in a particular belief state b. The solution to a POMDP is the optimal policy, π^*, which chooses the action that maximizes the value function in each belief state:

$$V^*(b) = \max_{a \in A} \left[R(b, a) + \gamma \sum_{o \in \Omega} P(o \mid b, a) V^*(b'_o) \right] \tag{11}$$

$$\pi^*(b) = \arg\max_{a \in A} \left[R(b, a) + \gamma \sum_{o \in \Omega} P(o \mid b, a) V^*(b'_o) \right]. \tag{12}$$

Finding an exact solution for π^* is often impractical for even moderately-sized POMDPs, so we typically must rely upon approximation methods.

2.3 Approximation Methods

There are generally two types of POMDP solution methods. Offline POMDP solution methods solve for the entire state space, approximating the optimal policy

for every possible belief state. Online methods solve only for the subspaces reachable from the current belief state, greatly reducing the size of the problem.

Many offline methods, such as Heuristic Search Value Iteration (HSVI) [14], Point-based Value Iteration (PBVI) [10], and Successive Approximation of the Reachable Space under Optimal Policies (SARSOP) [6] require discrete, finite state and action spaces. These methods approximate the value function as a convex, piecewise-linear function of the belief state. The algorithm iteratively tightens upper and lower bounds on the function until the approximation converges to within some predefined bounds of the optimal value function. Once a value function approximation is obtained, the optimal policy may then be followed by choosing the action that maximizes the function in the current belief state.

Online, real-time POMDP approaches have been applied to aircraft collision avoidance in the past. Winder [18] applied the POMDP framework to the aircraft collision avoidance problem, where he assumed Gaussian intruder process noise and used intruder behavioral modes for belief state compression. These modes described the overall behavior of the intruder, such as climbing, level, or descending, to differentiate significant trends in intruder action from noise. Winder showed that a POMDP-based collision avoidance system can have an acceptable probability of unnecessary alerts compared to other methods.

Real-Time Belief Space Search (RTBSS) [9, 11] is another online POMDP approximation method. RTBSS uses a discrete POMDP formulation and searches the reachable belief space for the optimal action to execute from the current belief state. The reachable belief space can be represented as a tree, as in Fig. 1, with the current belief state b_0 as the root node. The branches correspond to the possible actions a_i observations o_j. Each action-observation combination results in a new belief state at the next depth level of the tree. The algorithm searches the tree

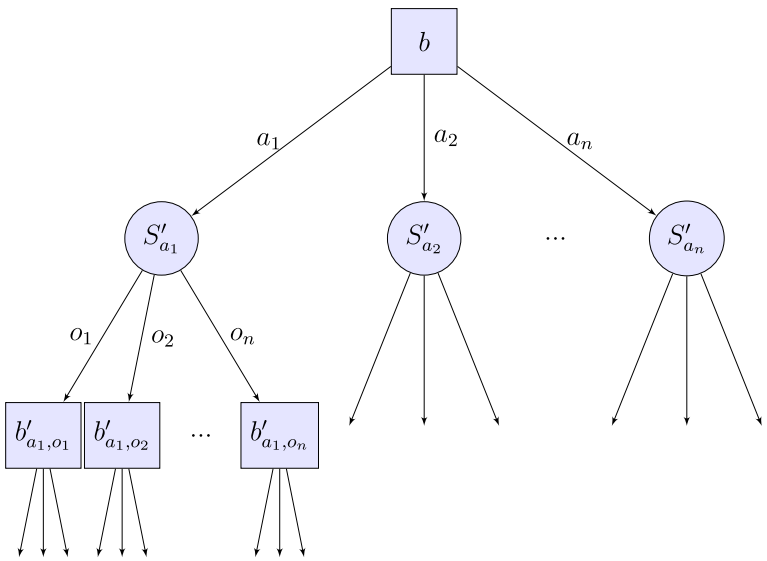


Fig. 1 Example of a POMDP search tree [9]

depth first to some maximum depth to determine the optimal action in the current belief state. RTBSS uses a branch and bound method to prune subtrees and reduce computation time.

One of the challenges of a problem with a continuous state space is representing the belief state. Thrun suggests a sample-based representation of the belief state, which permits the use of non-linear, non-Gaussian transition models. He uses belief state projection to generate a posterior distribution, or the future belief state, for a particular belief state-action pair [17]. Such a sample-based representation can be applied to the aircraft collision avoidance problem.

3 Monte Carlo Real-Time Belief Space Search (MC-RTBSS)

The Monte Carlo Real-Time Belief Space Search (MC-RTBSS) algorithm combines RTBSS with belief state projection to compute the optimal action from the current belief state. MC-RTBSS is designed for continuous state and observation spaces and a finite action space. The algorithm takes a sample-based belief state representation and chooses the action that maximizes the expected future discounted return. MC-RTBSS uses a branch and bound method, combined with an action sorting procedure, to prune sub-optimal subtrees.

The execution of MC-RTBSS is shown in Algorithm 1. The algorithm uses EXPAND to recursively search for the optimal action, which the agent executes before perceiving the next observation and updating the belief state. The initial belief state is represented by b_0.

Algorithm 1: ONLINEPOMDPALGORITHM

Function ONLINEPOMDPALGORITHM()
  Static: b: The current belief state.
  D: The maximum search depth.
  action: The best action.
$b \leftarrow b_0$
while *simulation is running* **do**
  EXPAND(b, D)
  $a \leftarrow$ *action*
  Execute a
  Perceive new observation o
  $b \leftarrow$ PARTICLEFILTER(b, a, o)

3.1 Particle Filtering

In order to handle continuous state spaces, the MC-RTBSS algorithm represents the belief state using weighted particles. The belief state, $b = \langle W, S \rangle$ is a set of state samples, $S = \{s_1, ..., s_{N_p}\}$, and a set of associated weights, $W = \{w_1, ..., w_{N_p}\}$, where N_p is the number of particles maintained in the belief state. The weights are normalized to sum to unity.

Belief state updates are accomplished using a particle filter (Algorithm 2). The algorithm takes a belief state, an action, and an observation as its argument and returns a new belief state for the next time step.

Algorithm 2: PARTICLEFILTER

Function $b' = \text{PARTICLEFILTER}(b,a,o)$
  Input: b: The current belief state.
  a: The action.
  o: The observation.
for $n = 1 : N_p$ **do**
  sample s from b
  sample s'_n according to $T(s,a,\cdot)$
  set particle weight: $w_n = O(s'_n,a,o)$
  add $\langle s'_n, w'_n \rangle$ to b'
normalize weights in b'
return b'

3.2 Belief Space Search

The depth-first search in MC-RTBSS, implemented by the recursive function EXPAND (Algorithm 3), is very similar to that of the original RTBSS. Given rough lower and upper bounds on the optimal value function, the routine attempts to prune as much of the search space as possible using branch and bound.

Algorithm 3: EXPAND

Function $L_T(b) = \text{EXPAND}(b,d)$
  Input: b: The current belief state.
  d: The current depth.
  Static: *action*: The best action.
  L: A lower bound on V^*.
  U: An upper bound on V^*.

action $\leftarrow$ *null*
if $d = 0$ **then**
  $L_T(b) \leftarrow L(b)$
else
  Sort actions $\{a_1, a_2, \ldots, a_{|A|}\}$ such that $U(b,a_i) \geq U(b,a_j)$ if $i \leq j$
  $i \leftarrow 1$
  $L_T \leftarrow -\infty$
  while $i \leq |A|$ *and* $U(b,a_i) > L_T(b)$ **do**
    $\{b'_{a_i,o_1}, \ldots, b'_{a_i,o_{N_o}}\} = \text{PARTICLEPROJECT}(b,a_i)$
    $L_T(b,a_i) \leftarrow R(b,a_i) + \gamma \frac{1}{N_o} \sum_{i=1}^{N_o} \text{EXPAND}(b'_{a_i,o_i}, d-1)$
    if $L_T(b,a_i) > L_T(b)$ **then**
      action $\leftarrow a_i$
      $L_T(b) \leftarrow L_T(b,a_i)$
    $i \leftarrow i+1$
return $L_T(b)$

The main difference between MC-RTBSS and RTBSS is how it represents belief states. In order to expand a node in the search tree, MC-RTBSS executes a belief

state projection procedure (Algorithm 4), similar to that suggested by Thrun [17]. When generating a future state for particle projection, the next state s' is unknown and must be sampled from $T(s,a,\cdot)$, where s is sampled from the belief state b. Similarly, when generating an observation in some state s, the observation must be sampled from $O(s,a,\cdot)$.

Algorithm 4: PARTICLEPROJECT

Function $\{b'_{a,o_1},\ldots,b'_{a,o_{N_o}}\} = \text{PARTICLEPROJECT}(b,a)$
Input: b: The belief state to project forward.
a: The action.
for $n = 1 : N_p$ **do**
  sample s from b
  sample s'_n according to $T(s,a,\cdot)$
for $i = 1 : N_o$ **do**
  sample x from b
  sample x' according to $T(x,a,\cdot)$
  sample o_i according to $O(x',a,\cdot)$
  for $n = 1 : N_p$ **do**
    set particle weight: $w_n = O(s'_n,a,o_i)$
    add $\langle s'_n, w'_n\rangle$ to b'_{a,o_i}
  normalize weights in b'_{a,o_i}
return $\{b'_{a,o_1},\ldots,b'_{a,o_{N_o}}\}$

This process is illustrated in Fig. 2. The search begins at the current belief state b, with MC-RTBSS expanding this node with action a_1. The algorithm generates a set of future states by sampling from b and then using the sampled states and a_1 to sample a set of future states, S'_{a_1}, from T. It also generates three observations (o_1, o_2, and o_3), which it uses to weight the samples in S'_{a_1}, yielding the three belief states b'_{a_1,o_1}, b'_{a_1,o_2}, and b'_{a_1,o_3}. These belief states are the children of the node b.

3.3 Complexity

There are $(N_o|A|)^d$ nodes at depth level d in the worst case, where $|A|$ is the number of possible actions. At each node, N_p samples are projected forward, weighted, and

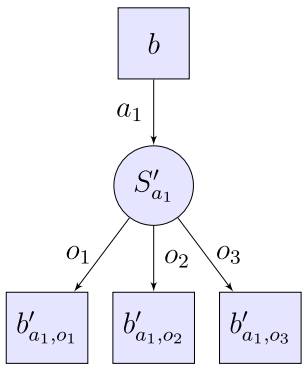


Fig. 2 Belief state projection example

evaluated by the reward function. In addition, a sorting function is used to order the set of actions by decreasing associated upper bounds (of the value function). This procedure propagates N_{sort} particles for each of $|A|$ actions at each node. Accounting for the belief state projection and the sort function at each node, the worst-case complexity for MC-RTBSS is bounded by $O((N_p + N_{\text{sort}}|A|)(N_o|A|)^D)$.

4 Collision Avoidance Application Domain

MC-RTBSS was integrated into a simulation infrastructure. The dynamic simulation operates at 10 Hz, while MC-RTBSS operates at 1 Hz. The simulation generates observations according to a noise model and propagates the aircraft according to a probabilistic dynamic model. The simulation is initialized with a script of commanded maneuvers for both aircraft. The scripted maneuvers are obtained from an encounter model [4].

4.1 Encounter Model

The airspace encounter model uses a Markov process, represented as a dynamic Bayesian network, to describe how the state variables change over time. Figure 3 shows the dynamic Bayesian network used to model aircraft behavior. The arrows between the variables represent direct statistical dependencies between the variables. The vertical rate $\dot{h}$, turn rate $\dot{\psi}$, and linear acceleration $\dot{v}$ vary with time. The airspace class A and altitude layer L are characteristic of each encounter and do not vary with time. The first set of variables (on the top of Fig. 3) represents the variable values at the current time (time t). The second set of variables (on the bottom of Fig. 3) represents the variable values at the next time step (time $t + 1$). Associated with each of the variables in the second set are conditional probability tables. The values of these tables are based on recorded radar data. The dynamic Bayesian network is then used to generate a sequence of scripted maneuvers by sampling from the conditional probability tables [4, 8].

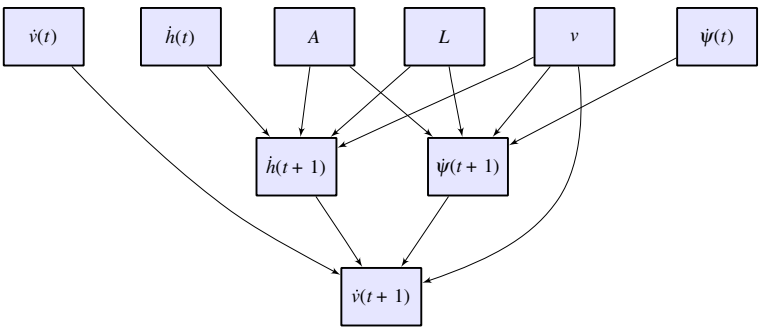


Fig. 3 Dynamic Bayesian network structure for the intruder dynamics [4]

Table 1 State variables

Variable	Definition	Units
v	Airspeed	ft/s
N	North displacement	ft
E	East displacement	ft
h	Altitude	ft
ψ	Heading	rad
θ	Pitch angle	rad
ϕ	Bank angle	rad
$\dot{v}$	Acceleration	ft/s^2
$\dot{h}$	Vertical rate	ft/s
$\dot{\psi}$	Turn rate	rad/s

4.2 States

The MC-RTBSS state is comprised of a vector of 21 state variables: 10 variables for each of the aircraft (the own aircraft and the intruder aircraft) and one variable for the simulation time. This vector is shown below

$$s = \langle v_1, N_1, E_1, h_1, \psi_1, \theta_1, \phi_1, \dot{v}_1, \dot{h}_1, \dot{\psi}_1, v_2, N_2, E_2, h_2, \psi_2, \theta_2, \phi_2, \dot{v}_2, \dot{h}_2, \dot{\psi}_2, t \rangle, \quad (13)$$

where the subscripts 1 and 2 correspond to the own aircraft and the intruder aircraft, respectively. The definition and units for each variable are shown in Table 1.

4.3 Observations

An observation o is composed of four elements, $\langle r, \beta, h_1, h_2 \rangle$, defined in Table 2. It is assumed that the noise is Gaussian with zero bias and that there is no correlation in the noise between variables. This noise model is similar to the model used in previous TCAS studies [5].

4.3.1 Actions

At each decision point, there are six actions from which the agent may choose. Each action lasts 5 s. One of the six candidate actions is always the scripted set of commands for that particular 5 s of time as generated by the encounter model. Every other candidate action has the same acceleration and turn rate as the scripted commands, but the vertical rate is different. The vertical rates for all of the actions are described in Table 3.

Table 2 Observation variables

Variable	Definition	Noise (σ)	Units
r	Slant range	50	ft
β	Bearing	0.1745	rad
h_1	Own altitude	50	ft
h_2	Intruder altitude	50	ft

Table 3 Vertical rates

Action	Vertical rate (ft/min)
a_1	2,000
a_2	1,500
a_3	0
a_4	−1,500
a_5	−2,000
a_6	Scripted

4.4 Reward Function

The reward function, $R(s, a)$, penalizes the agent for both near mid-air collisions (NMACs) and deviation from the nominal flight path. An NMAC occurs when the intruder comes within 100 ft vertically and 500 ft horizontally. The penalty for an NMAC is denoted λ. The reward function is

$$R(s, a) = \begin{cases} \text{deviation penalty} + \lambda, & \text{if NMAC} \\ \text{deviation penalty}, & \text{otherwise.} \end{cases} \tag{14}$$

The deviation penalty is the mean deviation (using Euclidean distance) from the nominal flight path for the 5 s between decision points.

4.5 Heuristic Utility Function

The heuristic utility function, $U(b, a)$, is an upper bound on the value of taking action a in belief state b. As with the original RTBSS algorithm, $U(b, a)$ is used in the action sorting procedure to increase the likelihood of pruning. In our collision avoidance problem, our utility function samples N_{sort} particles from the current belief state and propagates them forward to the next decision point. The function evaluates the reward function and sets the utility to the mean of these rewards. The output of the utility function will be greater than the computed value of the action choice (and thus serves as an upper bound) because the true value of an action is the discounted return and the reward function only penalizes.

4.6 MC-RTBSS Parameters

The tunable parameters of MC-RTBSS and our collision avoidance problem are summarized in Table 4. This section describes the qualitative effects expected when varying these parameters.

Table 4 Tunable parameters

Parameter	Description
N_p	Number of particles in belief states
N_o	Number of observations to generate for an action
λ	NMAC cost
D	Maximum search depth
N_{sort}	Number of particles used for each action in sort function

4.6.1 Number of Particles, N_p

The number of particles used to represent the belief state affects the accuracy of the belief state approximation. Increasing the number of particles increases the likelihood that the algorithm will adequately reason about rare events, such as the intruder making an aggressive maneuver which might result in an NMAC. The computation time per node in the search tree increases linearly with N_p.

4.6.2 Number of Observations, N_o

Each generated observation affects the weights of the particles representing the belief state. Hence, the number of observations generated when a node is expanded (for each action) determines the number of future sampled belief states. Increasing the number of observations increases the likelihood that rarer observations will be generated. Hence, increasing the number of observations increases the diversity of child belief state distributions, which in turn increases the likelihood that MC-RTBSS will account for rarer future events. Because the computation time grows polynomially with the product of N_o and the number of possible actions, increasing N_o results in an even greater impact on the computation time than increasing N_p.

4.6.3 NMAC Penalty, λ

The cost of an NMAC affects both how likely an NMAC must be to initiate an avoidance maneuver and how much deviation is allowed to avoid collision. The NMAC cost has no direct effect on computation time.

4.6.4 Maximum Search Depth, D

The maximum search depth affects the planning horizon of the algorithm. In the collision avoidance problem, MC-RTBSS will not maneuver to avoid an NMAC that occurs beyond the horizon. Without pruning, the computation time grows exponentially with D and results in the greatest increase in computation time of all parameters.

4.6.5 Number of Particles in the Action Sort Function, N_{sort}

The number of particles used in the sort function increases the accuracy of the upper bounds associated with taking each action from the current belief state. In addition, these upper bounds are used to sort the actions to aid in pruning. Increasing N_{sort} tightens the upper bounds used for pruning in the algorithm, which could reduce actual computation time. However, the computation associated with the sort function, $O(N_{\text{sort}}|A|)$, increases linearly with increasing N_{sort} as well.

5 Simulation Results

The performance characteristics of the MC-RTBSS algorithm are evaluated in simulating simple, level encounters. The goal of these simulations is to demonstrate the effect of individual parameters on the behavior of the algorithm in the context of collision avoidance. Two different scenarios are used to demonstrate different aspects of MC-RTBSS behavior. For both cases, the encounter begins at $t = 0$ s,

Table 5 Single encounter initial conditions

Variable	Head-on case	Offset case
$v_1 = v_2$	338 ft/s	338 ft/s
$h_1 = h_2$	4,500 ft	4,500 ft
N_1	0 ft	0 ft
N_2	13,520 ft	13,520 ft
E_1	0 ft	−900 ft
E_2	0 ft	0 ft
ψ_1	0°	0°
ψ_2	180°	180°

the duration is 30 s, and the closest point of approach (CPA) occurs 20 s into the encounter. In both scenarios, the own ship and intruder aircraft are initialized at the same altitude on straight and level trajectories. The own ship is heading north while the intruder is heading south. In the offset case, the own ship is offset 900 ft west of the intruder. No NMAC occurs in this case. In the head-on case, the aircraft are set to result in an NMAC. The initial conditions for both cases are shown in Table 5.

The purpose of the offset case is to evaluate the ability of the algorithm to identify and respond to future potential (less likely) safety events. For example, the intruder aircraft could maneuver at any time before CPA and eventually cause an NMAC. Figure 4 shows an example of 30 intruder trajectories generated from the encounter model for the first 20 s of the offset encounter. While most of the trajectories tend to head south, one trajectory veers to the west toward the own ship flight path, resulting in an NMAC with a vertical miss distance (VMD) of 69 ft and horizontal miss distance (HMD) of 4 ft at $t = 20$ s. This scenario is designed to test whether MC-RTBSS can identify such possible trajectories and maneuver to ensure that an NMAC is unlikely to occur, regardless of the intruder's maneuvers. This scenario is used for the N_p, N_o, and λ trials. The head-on case is used to determine the effect of varying parameters on the avoidance maneuver choice and timing when an NMAC is likely. This scenario is used for the D and N_{sort} trials.

Both TCAS-equipped and MC-RTBSS-equipped aircraft with different parameter settings are simulated in the offset encounter for comparison. The MC-RTBSS

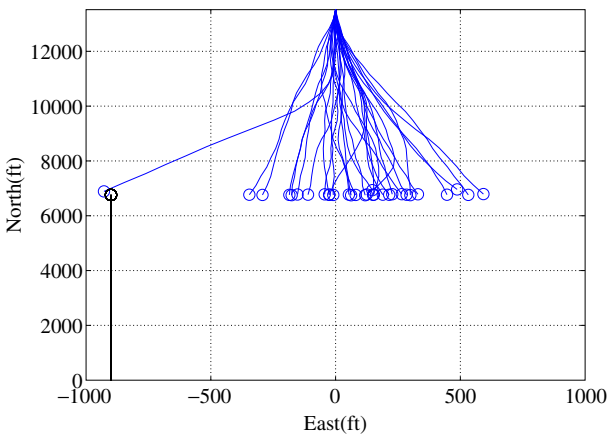


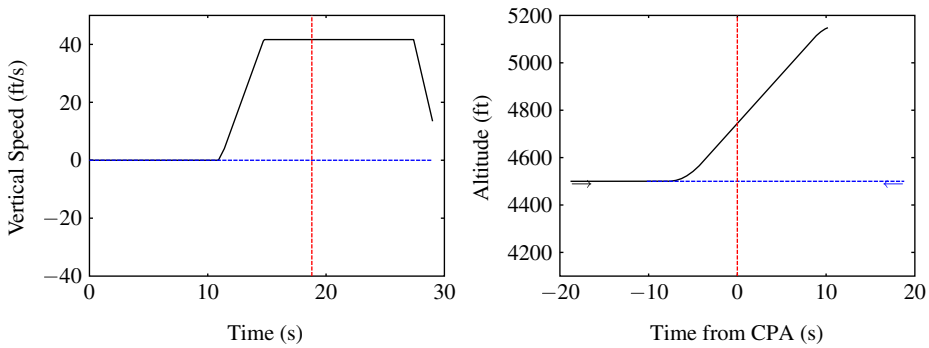
Fig. 4 Example intruder trajectories from the encounter model, 20 s duration, offset scenario, no collision avoidance system

Table 6 Single encounter MC-RTBSS parameter settings

Experiment	N_p	N_o	λ	D	N_{sort}	Normalized complexity
Small N_p	**10**	3	10^{15}	3	10	10
Large N_p	**3,000**	3	10^{15}	3	10	458
Small N_o	100	**1**	10^{15}	3	10	1
Large N_o	100	**10**	10^{15}	3	10	851
Small λ	100	3	$\mathbf{10^3}$	3	10	24
Large λ	100	3	$\mathbf{10^{15}}$	3	10	24
Small D	100	3	10^{15}	**3**	10	24
Large D	100	3	10^{15}	**3**	10	408
Small N_{sort}	100	3	10^{15}	3	**10**	24
Large N_{sort}	100	3	10^{15}	3	**1,000**	912

parameter settings tested are in Table 6. The last column lists a normalized worst-case estimate on the upper bounds of complexity with each parameter setting. These numbers are the result of applying the parameter settings to the worst-case complexity discussed in Section 3.3, $O((N_p + N_{\text{sort}}|A|)(N_o|A|)^D)$, for $|A| = 6$ and normalizing by the lowest value, which is associated with the the Small N_o case. Pruning reduces this complexity. Computation time is approximately proportional to these bounds. The normalized complexity is used to express the worst-case computation bounds between the simulations as they relate to each other. For example, the Small D simulation is estimated to require approximately 24 times as much computation as the Small N_o simulation in the worst case of each.

The result of the offset encounter simulation with a TCAS-equipped own aircraft is shown in Fig. 5. In this encounter, at $t = 11$ s TCAS issues a climb RA at 41.7 ft/s and at $t = 27$ s TCAS issues clear of conflict and the aircraft begins to level off. The resulting miss distances are 294 ft vertically and 900 ft horizontally. TCAS issues the climb RA because the intruder is projected to violate the TCAS range and relative altitude thresholds at CPA.



(a) Vertical Speed (b) Altitude

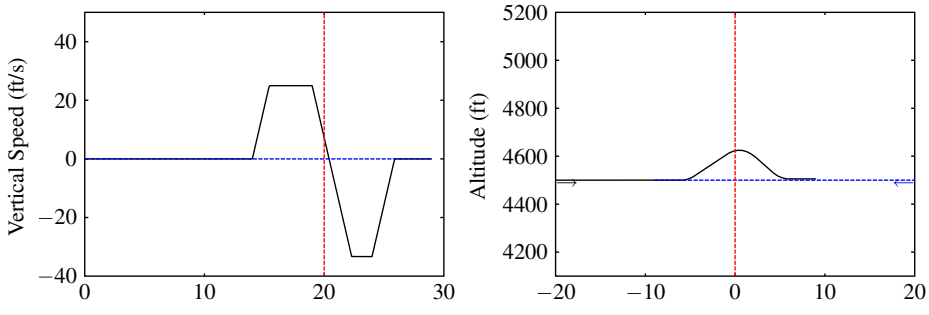
Fig. 5 Single encounter, TCAS (the *black line* represents own ship data while the *blue line* represents intruder data, and the *vertical red line* denotes CPA)

5.1 Varying Number of Particles, N_p

The results of an MC-RTBSS-equipped aircraft with $N_p = 3{,}000$ are shown in Fig. 6. With $N_p = 10$, MC-RTBSS does not command an avoidance maneuver and VMD is 0 ft and HMD is 900 ft. With $N_p = 3{,}000$, at $t = 14$ s MC-RTBSS commands a climb at 25 ft/s; at $t = 19$ s it commands a descent at 33.3 ft/s; and at $t = 24$ s it commands a level off. The resulting VMD is 123 ft and HMD is 900 ft. The algorithm is exhibiting three interesting behaviors in this scenario. First, MC-RTBSS identifies the risk of an NMAC near CPA and commands a climb at $t = 14$ s. The larger number of samples used (3,000 instead of 10) allows the the algorithm to generate and consequently account for less likely events, such as the induction of an NMAC. In other words, a larger number of particles allows the belief state to more effectively span the actual belief space, including unlikely events. When $N_p = 10$, if none of the 10 samples capture these events, the events are impossible from the perspective of the algorithm. A large number of particles is needed to accurately approximate the true distribution. As the aircraft approach CPA at $t = 20$ s, MC-RTBSS recognizes that an NMAC is no longer likely. The penalty associated with the risk of NMAC no longer outweighs the penalty of deviating from the nominal path and MC-RTBSS shifts its priority to minimizing deviation as quickly as possible. As a result, the algorithm commands the largest descent rate possible (33.3 ft/s) at $t = 19$ s, before leveling off just prior to regaining track at $t = 24$ s. Because of the low resolution of the action options available to MC-RTBSS in this implementation (one action every 5 s), the algorithm realizes that choosing to descend further will result in even greater deviation below track and chooses to level off instead.

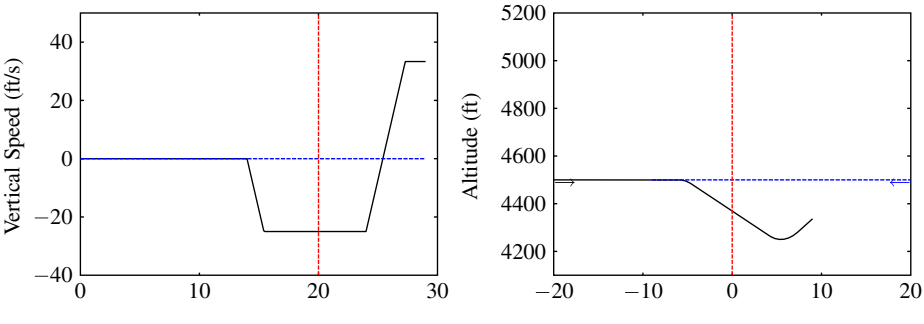
5.2 Varying Number of Observations, N_o

The results of an MC-RTBSS-equipped aircraft with $N_o = 10$ are shown in Fig. 7. With $N_o = 1$, MC-RTBSS does not command an avoidance maneuver with these parameter settings and VMD is 0 ft and HMD is 900 ft. For $N_o = 10$ s, at $t = 14$ s MC-RTBSS commands a descent at 25 ft/s to avoid a potential NMAC, and at $t = 24$ s it commands a climb at 33.3 ft/s to regain track. The resulting VMD is −131 ft and



(a) Vertical Speed, $N_p = 3000$. (b) Altitude, $N_p = 3000$.

Fig. 6 Varying N_p, single encounter



(a) Vertical Speed, $N_o = 10$ (b) Altitude, $N_o = 10$

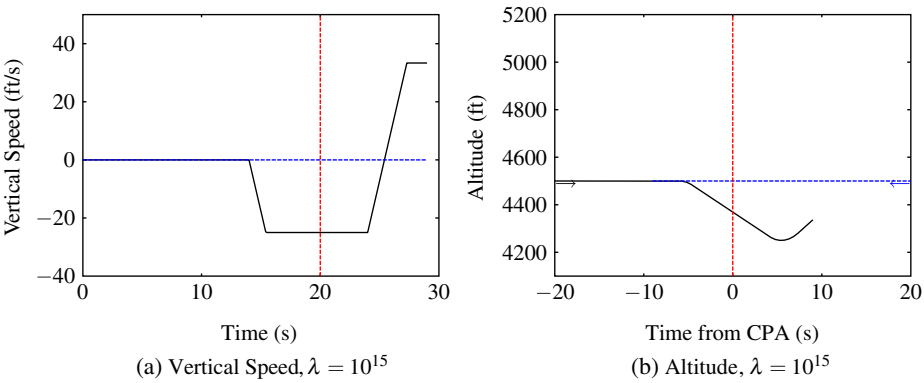
Fig. 7 Varying N_o, single encounter

HMD is 900 ft. The increase in the number of observations increases the likelihood that MC-RTBSS will sample a less likely observation and more heavily weight a different portion of the particles than it would with a more likely observation. When MC-RTBSS expands a node weighted in such a way, more of the heavily weighted particles will tend to be sampled and propagated. The consequence of sampling more observations is a search tree of greater breadth, allowing the algorithm to search subtrees that more fully span the belief space, allowing MC-RTBSS to account for rare events. With fewer observations, the search tree is narrow and MC-RTBSS may only account for a small subset of the belief space. Due to the stochastic nature of the observation generating process, in the case of few observations, this subset may include rare events, though it tends to include only the most likely ones.

There is a discrepancy between the avoidance maneuvers commanded in the $N_p = 3{,}000$ case, which results in a climb command, and the $N_o = 10$ case, which results in a descend command. MC-RTBSS commands different avoidance maneuvers despite the fact that the encounter scenario is the same in each case. This difference in commanded maneuver highlights the effect of the inherent randomness of a sample-based algorithm, particularly with small numbers of samples. Unless the intruder is significantly more likely to maneuver in a particular sense (climb or descent), both a climb and descent are equally effective avoidance maneuvers in this situation. If either option will yield the same rewards, then MC-RTBSS will choose whichever comes first in the sorted list of actions. If the utility of taking either action is arbitrarily close, then the order of the two in the list will be arbitrary as well.

5.3 Varying NMAC Penalty, λ

The results of an MC-RTBSS-equipped aircraft with $\lambda = 10^{15}$ are shown in Fig. 8. With $\lambda = 1{,}000$, MC-RTBSS does not command an avoidance maneuver and the resulting VMD is 0 ft and HMD is 900 ft. With $\lambda = 10^{15}$, at $t = 14$ s MC-RTBSS commands a descent of 25 ft/s, and at $t = 24$ s a climb at 33.3 ft/s to begin to regain track. The reason for the different command with a larger λ is that in this scenario, the likelihood that the intruder aircraft will maneuver in such a way as to induce



(a) Vertical Speed, $\lambda = 10^{15}$ (b) Altitude, $\lambda = 10^{15}$

Fig. 8 Varying λ, single encounter

an NMAC is very small. Consequently, particles containing NMACs are not likely to be weighted very heavily and, hence, do not contribute to the value function significantly with a small NMAC penalty. However, if the penalty for NMAC is large enough, the expected reward associated with samples containing these events will contribute more to the value function, significantly reducing the value associated with taking actions that lead to these situations. In this case, MC-RTBSS decides that an avoidance maneuver maximizes the value function and that staying level does not. After the threat of NMAC has passed, MC-RTBSS commands a climb to begin to regain track and minimize the deviation penalty.

5.4 Varying Maximum Search Depth, D

For the maximum search depth experiments, the head-on scenario is used to investigate how varying D affects behavior in the presence of an imminent NMAC. The results of an MC-RTBSS-equipped aircraft with $D = 3$ and $D = 4$ are shown in Fig. 9. When $D = 3$, at $t = 15$ s MC-RTBSS commands a 33.3 ft/s climb, and at $t = 20$ s it commands a reduction in climb rate to 25 ft/s. The VMD is 143 ft and HMD is 7 ft. When $D = 4$, at $t = 15$ s the algorithm commands a 25 ft/s descent and the VMD is −114 ft and HMD is 5 ft. MC-RTBSS commands an avoidance maneuver with both settings. This result is expected, because in either case, MC-RTBSS is optimizing the reward function. The optimal path in this situation would just skirt the edge of the NMAC cylinder around the intruder, minimizing deviation while avoiding an NMAC. The algorithm would not be expected to command an avoidance maneuver sooner with the current reward function despite the longer planning horizon.

5.5 Varying Number of Particles in the Action Sort Function, N_{sort}

The head-on encounter is used for the N_{sort} experiments as well. The results of an MC-RTBSS-equipped aircraft with $N_{\text{sort}} = 10$ and $N_{\text{sort}} = 1{,}000$ are shown in Fig. 10. With $N_{\text{sort}} = 10$, at $t = 14$ s MC-RTBSS commands a climb at 25 ft/s and at $t = 24$ s,

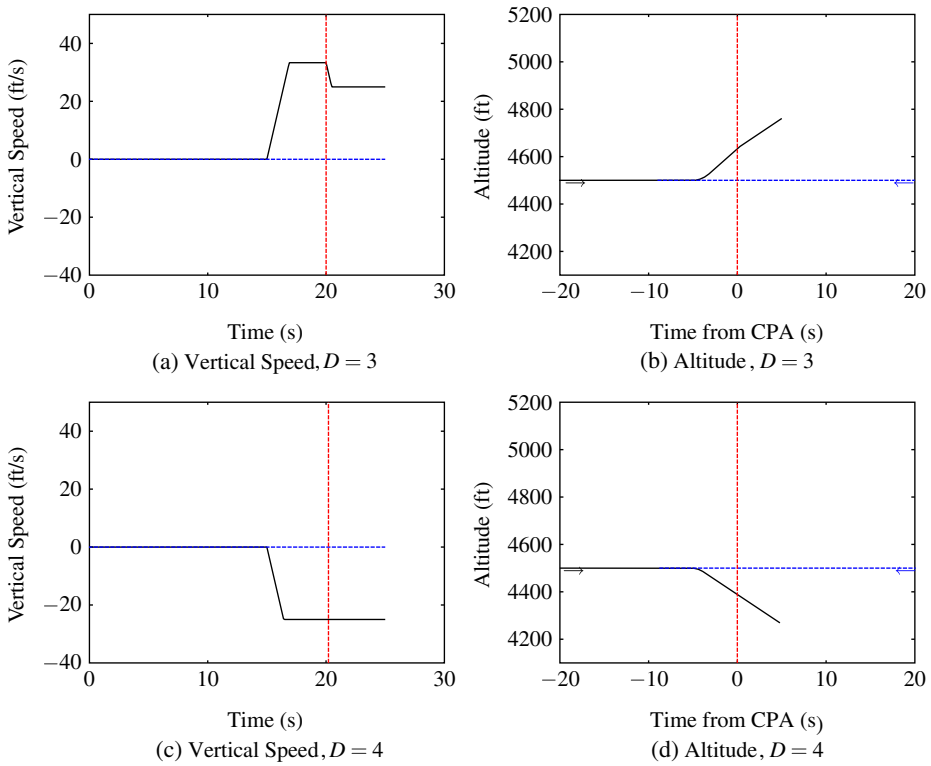
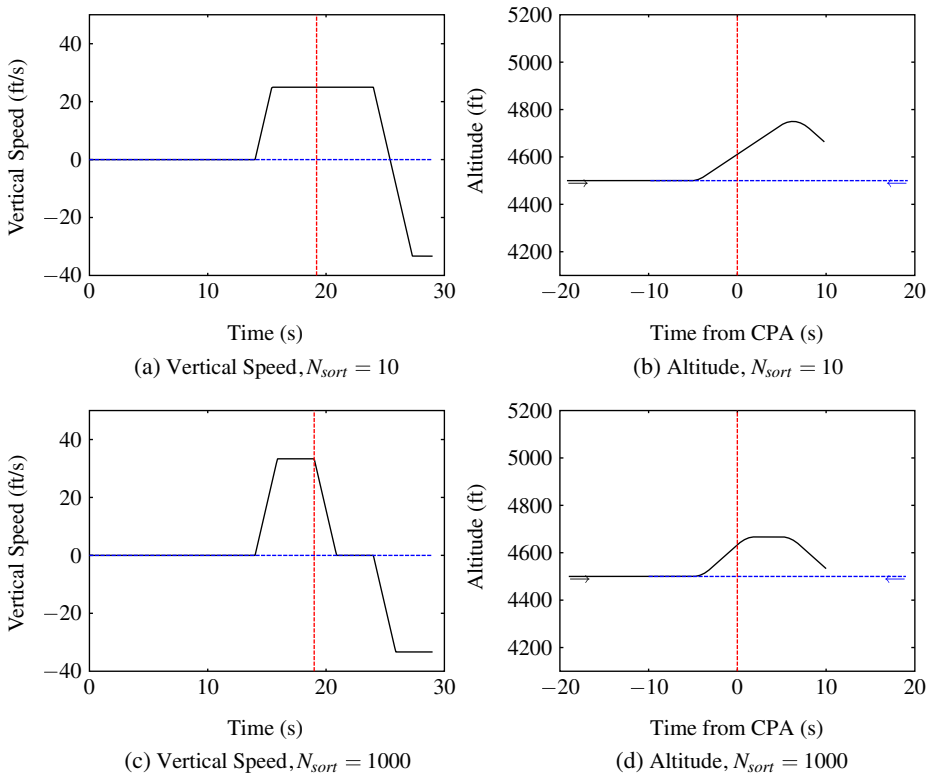


Fig. 9 Varying D, single encounter

it commands a descent at −33.3 ft/s to regain track. The resulting VMD is 114 ft and HMD is 5 ft. With $N_{\text{sort}} = 1{,}000$, at $t = 14$ s MC-RTBSS commands a climb at 33.3 ft/s; at $t = 19$ s it commands a level off; and at $t = 24$ s it commands a descent at 33.3 ft/s to regain track. MC-RTBSS commands an avoidance maneuver with both settings. This result is expected because N_{sort} does not directly affect the algorithm's reasoning. This parameter most directly affects the accuracy of the heuristic function, U, which would most significantly affect pruning of subtrees. A more accurate heuristic in the sort function should result in more pruning.

While pruning and the associated reduction in computation time is desirable, the stochastic nature of the sort function has undesirable consequences, because the algorithm is not guaranteed to prune only suboptimal subtrees. The optimal subtree could be pruned, resulting in a suboptimal action choice. This appears to have happened in the case with $N_{\text{sort}} = 10$. In the $N_{\text{sort}} = 10$ case, the algorithm does not begin to descend until $t = 24$ s, resulting in a slight increase in mean deviation for this encounter (compared to the $N_{\text{sort}} = 1{,}000$ case). This could be the result of pruning the optimal action (descending) at $t = 19$ s in the $N_{\text{sort}} = 10$ case, because the algorithm did more pruning with a smaller N_{sort}. This suboptimal action choice could also be the result of particularly noisy observations, leading the algorithm to the belief that the intruder is closer, or a combination of both effects.



(a) Vertical Speed, $N_{sort} = 10$ (b) Altitude, $N_{sort} = 10$

(c) Vertical Speed, $N_{sort} = 1000$ (d) Altitude, $N_{sort} = 1000$

Fig. 10 Varying N_{sort}, single encounter

The average total number of nodes expanded at each decision point in each simulation for all single encounter simulations is compared to the theoretical worst-case number of nodes in Fig. 7. In all cases, the number of nodes expanded is less than the worst-case total number of nodes, which indicates that MC-RTBSS is pruning a significant portion of the search tree. As expected (and seen in Table 7), increasing

Table 7 Pruning results

Experiment	N_p	N_o	λ	D	N_{sort}	Worst case	Avg. nodes expanded	Percent pruned
Small N_p	**10**	3	10^{15}	3	10	6,175	139	98
Large N_p	**3,000**	3	10^{15}	3	10	6,175	325	95
Small N_o	100	**1**	10^{15}	3	10	259	5	98
Large N_o	100	**10**	10^{15}	3	10	219,661	21,568	90
Small λ	100	3	$\mathbf{10^3}$	3	10	6,175	134	98
Large λ	100	3	$\mathbf{10^{15}}$	3	10	6,175	342	94
Small D	100	3	10^{15}	**3**	10	6,175	702	89
Large D	100	3	10^{15}	**4**	10	346,201	10,404	97
Small N_{sort}	100	3	10^{15}	3	**10**	6,175	823	87
Large N_{sort}	100	3	10^{15}	3	**1,000**	6,175	752	88

the N_{sort} does result in a decrease in the number of nodes expanded, from 832 with $N_{\text{sort}} = 10$, to 752 with $N_{\text{sort}} = 1{,}000$. The last column of Table 7 shows the average percentage of nodes pruned by the branch and bound method. The Small N_p case simulates the offset encounter and prunes 98% of the nodes whereas the Small N_{sort} case simulates the head-on encounter and prunes only 87%. The significant difference between the percent pruned in the Small N_p case and the Small N_{sort} case (both of which have the same parameter settings), highlights the effect of different scenarios on pruning and consequently computation time.

5.6 Discussion

The results demonstrate the behavior of MC-RTBSS with various parameter settings. The actual maneuvers the algorithm chooses in each simulation vary slightly, particularly in sense (climb vs. descent) in the co-altitude, head-on scenario. This nondeterministic behavior highlights the effect of a sample-based algorithm with small sample sizes. As N_p and N_o increase, the value function converges to the true value. Similarly, as N_{sort} increases, the action list ordering will converge. Increasing all three parameters sufficiently is expected to ensure consistency in MC-RTBSS behavior. The other two parameters, λ and D, most directly affect the sensitivity of the algorithm to potential threats and the false alarm rate (i.e., probability of issuing an unnecessary avoidance maneuver).

The branch-and-bound method significantly reduces the problem size. In all cases, a significant portion of the tree was pruned (at least 87%), resulting in a significant reduction in computation time compared to the worst case.

6 Conclusion

The results of the single encounter simulations demonstrate how each algorithm parameter affects the behavior of MC-RTBSS. Increasing the number of particles increases the likelihood that MC-RTBSS will be sensitive to unlikely events. Observations affect the relative weighting of such events. Increasing the number of observations, N_o, increases the accuracy of the posterior distributions over future states, allowing the algorithm to make better approximations of the optimal value function. Increasing the cost of NMAC makes the algorithm more conservative, commanding maneuvers to avoid less likely potential safety events and deviating more to avoid them. Increasing the maximum search depth allows the algorithm to look further ahead and consider events further into the future. Increasing the number of particles used in the sorting function improves the accuracy of the heuristic function used in the branch-and-bound method.

An investigation of the nodes of the search tree expanded shows that the branch-and-bound method pruned significant portions of the tree, from 87% to 98% in the scenarios tested. These results indicate that this method effectively reduced the problem to less than one tenth of its original size in most cases, while still permitting the use of a high-dimensional state representation.

This paper focused on the effects parameter settings have on algorithm behavior. In order to gain further insight into the potential performance of MC-RTBSS, as would be required in a safety study necessary for an operational implementation,

Monte Carlo simulation is needed. This simulation would need to use encounters sampled from an airspace encounter model, as was done to certify previous collision avoidance algorithms [1]. In addition, MC-RTBSS would need to use more particles to adequately represent the belief state. Further work is needed to investigate reimplementing MC-RTBSS in a lower-level programming language to reduce computation time. The Simulink implementation (with embedded Matlab functions) required significantly more time than would an implementation in C, for example. Compilation in Real Time Workshop would also reduce computation time significantly.

Another strategy for reducing computation time while enhancing performance is to reduce the number of particles and observations required to represent the belief space and identify potential NMACs. Future work would be needed to use importance sampling in the particle projection procedure to target trajectories that lead to NMACs when sampling from belief states and from the encounter model [15]. The resulting trajectories would then be weighted according to their actual likelihood. While the NMAC trajectories would remain rare events and would be weighted accordingly, this method would ensure that the algorithm is searching these significant portions of the belief space without using prohibitively large numbers of particles and observations.

Acknowledgements Special thanks are due to James K. Kuchar for feedback during the evolution of this project and to J. Daniel Griffith and Matthew Edwards for assistance with the simulation framework.

References

1. Espindle, L.P., Griffith, J.D., Kuchar, J.K.: Safety analysis of upgrading to TCAS version 7.1 using the 2008 U.S. correlated encounter model. Project Report ATC-349, Lincoln Laboratory, Lexington, Mass. (2009)
2. Frazzoli, E., Dahleh, M.A., Feron, E.: Real-time motion planning for agile autonomous vehicles. AIAA Journal of Aerospace Computing, Information, and Communication **25**, 116–129 (2004)
3. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artif. Intell. **101**, 99–134 (1998)
4. Kochenderfer, M., Kuchar, J., Espindle, L., Griffith, J.: Uncorrelated encounter model of the national airspace system version 1.0. Project Report ATC-345, Lincoln Laboratory, Lexington, Mass. (2008)
5. Kuchar, J.K., Drumm, A.C.: The traffic alert and collision avoidance system. Linc. Lab. J. **16**(2), 277–296 (2007)
6. Kurniawati, H., Hsu, D., Lee, W.S.: SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Proceedings in Robotics: Science and Systems (2008)
7. Lawrence, D., Pisano, W.: Lyapunov vector fields for autonomous unmanned aircraft flight control. J. Guid. Control Dyn. **31**(5), 1220–1229 (2008)
8. Neapolitan, R.: Learning Bayesian Networks. Pearson Prentice Hall, Upper Saddle River, NJ (2004)
9. Paquet, S., Tobin, L., Chaib-draa, B.: Real-time decision making for large POMDPs. In: Advances in Artificial Intelligence (LNAI 3501), pp. 450–455 (2005)
10. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for POMDPs. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 1025–1032. Acapulco, Mexico (2003)
11. Ross, S., Pineau, J., Paquet, S., Chaib-draa, B.: Online planning algorithms for POMDPs. J. Artif. Intell. Res. **32**, 663–704 (2008)

12. RTCA: Minimum operational performance standards for traffic alert and collision avoidance system II (TCAS II) airborne equipment. Tech. rep., RTCA/DO-185A, Washington, D.C. (1997)
13. Schouwenaars, T., Mettler, B., Feron, E., How, J.: Hybrid model for trajectory planning of agile autonomous aerial vehicles. Journal of Aerospace Computing, Information, and Communication, Special Issue on Intelligent Systems **1**, 629–651 (2004)
14. Smith, T., Simmons, R.G.: Heuristic search value iteration for POMDPs. In: Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI) (2004)
15. Srinivasan, R.: Importance Sampling: Applications in Communications and Detection. Springer-Verlag, Berlin, Germany (2002)
16. Sundqvist, B.G.: Auto-ACAS—robust nuisance-free collision avoidance. In: Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference, pp. 3961–3963. IEEE (2005)
17. Thrun, S.: Monte Carlo POMDPs. In: Solla, S., Leen, T., Müller, K.R. (eds.) Advances in Neural Information Processing Systems 12, pp. 1064–1070. MIT Press (2000)
18. Winder, L.F.: Hazard avoidance alerting with Markov decision processes. Ph.D. thesis, MIT, Cambridge, Mass. (2004)