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

by

Travis Benjamin Wolf

B.S., Aerospace Engineering (2007)
United States Naval Academy

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2009

Author .....
Department of Aeronautics and Astronautics
May 22, 2009

Certified by .....
James K. Kuchar
Associate Group Leader, Lincoln Laboratory
Thesis Supervisor

Certified by .....
John E. Keesee
Senior Lecturer
Thesis Supervisor

Accepted by .....
David L. Darmofal
Professor of Aeronautics and Astronautics
Associate Department Head
Chair, Committee on Graduate Students

Aircraft Collision Avoidance

Using Monte Carlo Real-Time Belief Space Search

by

Travis Benjamin Wolf

Submitted to the Department of Aeronautics and Astronautics
on May 22, 2009, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

This thesis presents the Monte Carlo Real-Time Belief Space Search (MC-RTBSS) algorithm, a novel, online planning algorithm for partially observable Markov decision processes (POMDPs). MC-RTBSS combines a sample-based belief state representation with a branch and bound pruning method to search through the belief space for the optimal policy. The algorithm is applied to the problem of aircraft collision avoidance and its performance is compared to the Traffic Alert and Collision Avoidance System (TCAS) in simulated encounter scenarios. The simulations are generated using an encounter model formulated as a dynamic Bayesian network that is based on radar feeds covering U.S. airspace. MC-RTBSS leverages statistical information from the airspace model to predict future intruder behavior and inform its maneuvers. Use of the POMDP formulation permits the inclusion of different sensor suites and aircraft dynamic models.

The behavior of MC-RTBSS is demonstrated using encounters generated from an airspace model and comparing the results to TCAS simulation results. In the simulations, both MC-RTBSS and TCAS measure intruder range, bearing, and relative altitude with the same noise parameters. Increasing the penalty of a Near Mid-Air Collision (NMAC) in the MC-RTBSS reward function reduces the number of NMACs, although the algorithm is limited by the number of particles used for belief state projections. Increasing the number of particles and observations used during belief state projection increases performance. Increasing these parameter values also increases computation time, which needs to be mitigated using a more efficient implementation of MC-RTBSS to permit real-time use.

Thesis Supervisor: James K. Kuchar
Title: Associate Group Leader, Lincoln Laboratory

Thesis Supervisor: John E. Keesee
Title: Senior Lecturer

Acknowledgments

This work was supported under Air Force contract number FA8721-05-C-0002. Interpretations, opinions, and conclusions are those of the authors and do not reflect the official position of the United States Government. This thesis leverages airspace encounter models that were jointly sponsored by the U.S. Federal Aviation Administration, the U.S. Air Force, and the U.S. Department of Homeland Security.

I would like to thank MIT Lincoln Laboratory Division 4, led by Dr. Bob Shin, for supporting my research assistantship, and in particular Group 42, led by Jim Flavin, for providing a productive and supportive work environment.

Words cannot express my gratitude for Dr. Jim Kuchar, Assistant Group Leader of Group 43 and my Lincoln Laboratory thesis supervisor, for providing me the opportunity to pursue graduate studies and for his time, effort, and support in this project. Despite his schedule, Dr. Kuchar provided timely feedback and guidance that ensured my success in this endeavor.

I extend my sincerest thanks to Mykel Kochenderfer of Group 42 for his time and tireless efforts on this project. He provided extensive support and guidance during all phases of this work and never hesitated to rush to assist whenever requested.

I would like to thank Dan Griffith, Leo Espindle, Matt Edwards, Wes Olsen, and James Chryssanthacopoulos of Group 42 for their technical and moral support during my time at Lincoln. My appreciation also goes to Vito Cavallo, Mike Carpenter, and Loren Wood of Group 42 for their invaluable support and expertise.

I would also like to acknowledge COL John Keesee, USAF (Ret.), my academic advisor, for his help in ensuring my smooth progression through MIT.

I would like to thank my parents, Jess, Nicky, and Jorge for their support and encouragement. Thanks also to my friends and partners at Houghton Management for ensuring that my MIT education was complete, quite literally "mens et manus." A special thanks goes to Philip "Flip" Johnson, without whose early assistance this would not have been possible. Last, I would like to thank Leland Schulz, Vincent Bove, and my other shipmates for their service in harm's way.

Contents

1 Introduction **15**

1.1 Aircraft Collision Avoidance Systems 16

1.1.1 Traffic Alert and Collision Avoidance System 16

1.1.2 Autonomous Collision Avoidance 17

1.2 Challenges 18

1.3 POMDP Approach 19

1.3.1 POMDP Solution Methods 21

1.3.2 Online Solution Methods 22

1.4 Proposed Solution 23

1.5 Thesis Outline 24

2 Partially-Observable Markov Decision Processes **27**

2.1 POMDP Framework 28

2.2 Representing Uncertainty 28

2.3 The Optimal Policy 29

2.4 Offline Algorithms: Discrete POMDP Solvers 31

2.5 Online Algorithms: Real-Value POMDP Solvers 32

2.5.1 Real-Time Belief Space Search 32

2.5.2 Monte Carlo POMDPs 33

3 Monte Carlo Real-Time Belief Space Search **35**

3.1 Belief State Valuation 35

3.2 Implementation 37

3.2.1 Belief State Projection 38
3.2.2 Particle Filtering 39
3.2.3 MC-RTBSS Recursion 39
3.2.4 Pseudocode 40
3.2.5 Complexity 42
3.3 Collision Avoidance Application Domain 43
3.3.1 Encounter Models 43
3.3.2 Simulation Environment 47
3.3.3 MC-RTBSS Parameters 54

4 Simulation Results **57**
4.1 Simple Encounter Cases 57
4.2 Single Encounter Simulation Results 60
4.2.1 TCAS 61
4.2.2 Varying Number of Particles, N_p 61
4.2.3 Varying Number of Observations, N_o 63
4.2.4 Varying NMAC Penalty, λ 65
4.2.5 Varying Maximum Search Depth, D 67
4.2.6 Varying Number of Particles in the Action Sort Function, N_{sort} 67
4.2.7 Discussion 71
4.3 Large Scale Simulation Results 72
4.3.1 Cost of NMAC, λ 73
4.3.2 Number of Particles, N_p 77
4.3.3 Number of Observations, N_o 81
4.3.4 Discussion 85

5 Conclusion **87**
5.1 Summary 87
5.2 Contributions 88
5.3 Further Work 89

List of Figures

2-1 POMDP model. 28

2-2 Example of a POMDP search tree. 33

3-1 Belief state projection example. 38

3-2 EXPAND example. 41

3-3 Dynamic Bayesian network framework for the uncorrelated encounter model. 46

4-1 Single encounters, no CAS. 59

4-2 Example intruder trajectories from the encounter model, 20 second duration, offset scenario, no CAS. 59

4-3 Single encounter, TCAS. 61

4-4 Varying N_p, single encounter. 62

4-5 Varying N_o, single encounter. 64

4-6 Varying λ, single encounter. 66

4-7 Varying D, single encounter. 68

4-8 Varying N_{sort}, single encounter. 69

4-9 Examples of generated encounters. 74

4-10 Total number of NMACs, varying λ. 75

4-11 Mean miss distance, varying λ. 76

4-12 Average deviation, varying λ. 77

4-13 Miss distance comparison, varying λ. 78

4-14 Average deviation comparison, varying λ. 78

4-15 Total number of NMACs, varying N_p. 79

4-16 Mean miss distance, varying N_p. 80
4-17 Average deviation, varying N_p. 81
4-18 Miss distance comparison, varying N_p. 82
4-19 Average deviation comparison, varying N_p. 82
4-20 Total number of NMACs, varying N_o. 83
4-21 Mean miss distance, varying N_o. 84
4-22 Average deviation, varying N_o. 85
4-23 Miss distance comparison, varying N_o. 86
4-24 Average deviation comparison, varying N_o. 86

List of Tables

2.1 POMDP framework 29

3.1 State variables 48
3.2 Observation variables 48
3.3 Action climb rates 49
3.4 Observation noise model 51
3.5 Tunable parameters 54

4.1 Single encounter initial conditions 58
4.2 Single encounter MC-RTBSS parameter settings 60
4.3 Nodes expanded 70
4.4 Pruning results 71
4.5 λ sweep values 73
4.6 N_p sweep values 77
4.7 N_o sweep values 81

Nomenclature

General

A	Set of actions
a	Action
a_i	ith action
b	Belief state
b_0	Initial belief state
b_t	Belief state at time t
$b_{a,o}$	Belief state given action a and observation o
$b'_{a,o}$	Future belief state given action a and observation o
D	Maximum search depth
d	Current search depth
L	Lower bound of V^*
L_T	Global lower bound within MC-RTBSS instantiation
N_o	Number of observations
N_p	Number of belief state projection particles
N_{PF}	Number of particle filter particles
N_{sort}	Number of sorting function particles
O	Observation model
o	Observation
o_i	ith observation
SE	State estimator
S	Set of states
S'_a	Future set of states given action a
$S'_{a,o,a'}$	Future set of states given action a, observation o, and action a'
s	State
s_i	ith state
s'	Future state

T	Transition function
t	Time
t_{max}	Maximum time in simulation
U	Utility function
V	Value function
V_π	Value function for policy π
V^*	Optimal value function
W	Set of belief state particle weights
γ	Discount factor
π	Action policy
π^*	Optimal policy

Domain Specific

E_1	Own ship East displacement
E_2	Intruder East displacement
$\dot{E_1}$	Own ship East velocity component
$\dot{E_2}$	Intruder East velocity component
h_1	Own ship altitude
h_2	Intruder altitude
$\dot{h_1}$	Own ship vertical rate
$\dot{h_2}$	Intruder vertical rate
N_1	Own ship North displacement
N_2	Intruder North displacement
$\dot{N_1}$	Own ship North velocity component
$\dot{N_2}$	Intruder North velocity component
r	Intruder range
v_1	Own ship airspeed
v_2	Intruder airspeed
$\dot{v_1}$	Own ship acceleration
$\dot{v_2}$	Intruder acceleration

$v_{horizontal}$	Horizontal velocity component
w_i	ith particle weight
β	Intruder bearing
θ_1	Own ship pitch angle
θ_2	Intruder pitch angle
λ	Penalty for NMAC
ϕ_1	Own ship bank angle
ϕ_2	Intruder bank angle
ψ_1	Own ship heading
ψ_2	Intruder heading
$\dot{\psi}_1$	Own ship turn rate
$\dot{\psi}_2$	Intruder turn rate

Chapter 1

Introduction

The Federal Aviation Administration (FAA) Federal Aviation Regulations (FAR) Section 91.113b states:

> [R]egardless of whether an operation is conducted under instrument flight rules or visual flight rules, vigilance shall be maintained by each person operating an aircraft so as to see and avoid other aircraft.

This "see and avoid" requirement is of particular concern in the case of unmanned aircraft. While the use of existing collision avoidance systems (CAS) in manned aircraft have been proven to increase the level of safety of flight operations, the use of these systems requires a pilot who can independently verify the correctness of alerts and visually acquire aircraft that the CAS may miss. The lack of an actual pilot in the cockpit to see and avoid presents unique challenges to the unmanned aircraft collision avoidance problem.

Collision avoidance systems rely on noisy, incomplete observations to estimate intruder aircraft state and use models of the system dynamics to predict the future trajectories of intruders. However, current collision avoidance systems rely on relatively naive predictions of intruder behavior, typically extrapolating the position of aircraft along a straight line. Most methods also use heuristically chosen safety buffers to ensure that the system will act conservatively in close situations. These assumptions can lead to poor performance including excessive false alarms and flight

path deviation.

More sophisticated models of aircraft behavior exist. For example, an airspace encounter model is a statistical representation of how aircraft encounter each other, describing the geometry and aircraft states during close encounters. Airspace encounter models provide useful information for tracking and future trajectory prediction that has not been incorporated into current collision avoidance systems.

This thesis presents an algorithm called Monte Carlo Real-Time Belief Space Search (MC-RTBSS) that can be applied to the problem of aircraft collision avoidance. The algorithm uses a partially-observable Markov decision process (POMDP) formulation with a sample-based belief state representation and may be feasible for online implementation. The general POMDP formulation permits the integration of different aircraft dynamic and sensor models, the utilization of airspace encounter models in a probabilistic transition function, and the tailoring of reward functions to meet competing objectives. This work demonstrates the effect of various parameters on algorithm behavior and performance in simulation.

1.1 Aircraft Collision Avoidance Systems

The Traffic Alert and Collision Avoidance System (TCAS) is the only collision avoidance system currently in widespread use. TCAS was mandated for large commercial cargo and passenger aircraft (over 5700 kg maximum takeoff weight or 19 passenger seats) worldwide in 2005 (Kuchar and Drumm, 2007). While TCAS is designed to be an aid to pilots, some preliminary work has commenced on developing automatic collision avoidance systems for manned aircraft and collision avoidance systems designed specifically for autonomous unmanned aircraft.

1.1.1 Traffic Alert and Collision Avoidance System

TCAS is an advisory system used to help pilots detect and avoid nearby aircraft. The system uses aircraft radar beacon surveillance to estimate the range, bearing, relative-altitude, range rate, and relative-altitude rate of nearby aircraft (RTCA, 1997). The

system's threat-detection algorithms project the relative position of intruders into the future using linear extrapolation, using the estimates of the intruder range-rate and relative-altitude rate. The system also uses a safety buffer to protect against intruder deviations from the nominal projected path. The algorithm declares the intruder as a threat if the intruder is projected to come within certain vertical and horizontal separation limits. If the intruder is deemed to be a threat and the estimated time until the projected closest point of approach (CPA) is between 20 and 48 seconds (depending on the altitude), then TCAS issues a traffic advisory (TA) in the cockpit to aid the pilot in visually acquiring the intruder aircraft. If the intruder is deemed to be a more immediate threat (CPA between 15 and 35 seconds, depending on altitude), then TCAS issues a resolution advisory (RA) to the cockpit, which includes a vertical rate command intended to avoid collision with the intruder aircraft. TCAS commands include specific actions, such as to climb or descend at specific rates, as well as vertical rate limits, which may command not to climb or descend above or below specified rates.

The TCAS threat resolution algorithm uses certain assumptions about the execution of RA commands. The algorithm assumes a 5 second delay between the issuance of the RA and the execution of the command, and that the pilot will apply a 0.25 g vertical acceleration to reach the commanded vertical rate. The algorithm also assumes that the intruder aircraft will continue along its projected linear path. However, as the encounter progresses, TCAS may alter the RA to accommodate the changing situation, even reversing the RA (from climb to descend, for example) if necessary. If the intruder aircraft is also equipped with TCAS, then the RA is coordinated with the other aircraft using the Mode S data link. A coordinated RA ensures that the aircraft are not advised to command the same sense (climb or descend) (Kuchar and Drumm, 2007).

1.1.2 Autonomous Collision Avoidance

Some high-performance military aircraft are equipped with the Autonomous Airborne Collision Avoidance System (Auto-ACAS), which causes an Auto-ACAS-equipped

aircraft to automatically execute coordinated avoidance maneuvers just prior to (i.e. the last few seconds before) midair collision (Sundqvist, 2005). Autonomous collision avoidance is an active area of research. One method explored in the literature uses predefined maneuvers, or maneuver automata, to reduce the complexity of having to synthesize avoidance maneuvers online (Frazzoli et al., 2004). The autonomous agent chooses the best automaton using rapidly-expanding random trees (RRTs). Maneuver automata have been used in mixed-integer linear programming (MILP) formulations of the problem, in which the best automaton is that which minimizes some objective function. A MILP formulation has also been used for receding horizon control (Schouwenaars et al., 2004). This method solves for an optimal policy in a given state, assuming some predicted future sequence of states, and chooses the current optimal action. As the agent moves to the next state, the process is repeated, in case some unexpected event occurs in the future. While all of these methods use some sort of dynamic model of the world, the models do not incorporate encounter model data, typically assuming a worst case scenario or simply holding the intruder velocity, vertical rate, and turn rate constant (Kuchar and Yang, 2000). MC-RTBSS incorporates some of the concepts used by many of these methods, such as the use of predefined maneuvers and a finite planning horizon.

1.2 Challenges

While the widespread use of TCAS has increased the safety of air travel (Kuchar and Drumm, 2007), it has some limitations. First, TCAS is ineffective if an intruder is not equipped with a functioning transponder, because it relies upon beacon surveillance. Second, TCAS was designed for use in the cockpit of a manned vehicle, in which there is a pilot who can utilize TCAS alerts to also "see-and-avoid" intruder aircraft; an unmanned aircraft with an automated version of TCAS would rely solely on limited TCAS surveillance for collision avoidance commands. This reliance has several problems, in addition to the possibility of encountering an intruder without a transponder. The information available to TCAS includes coarse altitude discretizations and par-

ticularly noisy bearing observations. In addition, the underlying assumptions of the TCAS algorithms (e.g. linear extrapolated future trajectories) do not necessarily reflect the reality of the airspace. A better CAS would be able to utilize all information available to verify that the Mode C reported altitude is correct and to provide a better estimate of the current and future intruder states. Such a CAS would allow for the easy integration of different sensors, such as Electro-Optical/Infra-Red (EO/IR) sensors, Global Positioning System (GPS), or radar, in order to provide the best estimate of the intruder state as possible. In addition, the CAS would utilize the information available in an airspace encounter model to achieve better predictions of the future state of the intruder. Modeling the aircraft collision avoidance problem as a POMDP addresses many of these issues.

1.3 POMDP Approach

A POMDP is a decision-theoretic planning framework that assumes the state is only partially observable and hence, must account for the uncertainty inherent in noisy observations and stochastic state transitions (Kaelbling et al., 1998). A POMDP is primarily composed of four parts: an observation or sensor model, a transition model, a reward function, and a set of possible actions. An agent working under a POMDP framework tries to act in such a way as to maximize the accumulation of future rewards according to the reward function. Beginning with some initial belief state (a probability distribution over the underlying state space), an agent acts and then receives observations. Using models of the underlying dynamics and of its sensors, the agent updates its belief state at each time step and chooses the best action. The solution to a POMDP is the optimal policy, which is a mapping from belief states to actions that maximize the expected future return.

POMDPs have been applied to a wide range of problems, from dynamic pricing of grid computer computation time (Vengerov, 2008) to spoken dialog systems (Williams and Young, 2007). In general, POMDPs are used for planning under uncertainty, which is particularly important in robotics. For example, a POMDP formulation was

used in the flight control system of rotorcraft-based unmanned aerial vehicles (RUAVs) (Kim and Shim, 2003). A POMDP formulation has also been used for aircraft collision avoidance, resulting in a lower probability of unnecessary alerts (when the CAS issues avoidance maneuvers when no NMAC would occur otherwise) compared to other CAS logic methods (Winder, 2004).

For a POMDP formulation of the aircraft collision avoidance problem, the state space consists of the variables describing the own aircraft position, orientation, rates, and accelerations as well as those of the intruder. The aircraft may receive observations from various sensors related to these variables, such as its own location via a GPS and, in the case of TCAS, the intruder range, bearing, and altitude from the Mode S interrogation responses. The system also has knowledge of the uncertainty associated with these observations; this knowledge is represented in an observation model. The aircraft has an associated set of possible maneuvers, and aircraft dynamic models specify the effect of each maneuver on the aircraft state variables. Use of an airspace encounter model provides the distribution of intruder maneuvers. Both of these pieces constitute the transition model. The reward function is used to score performance, which may involve competing objectives such as avoiding collision and minimizing deviation from the planned path.

In addition to providing an alternative approach to the aircraft collision avoidance problem, the POMDP framework offers advantages compared to previous approaches. First, the use of a reward function facilitates the explicit specification of objectives. The algorithm optimizes performance relative to these objectives. Undesirable events can be penalized in the reward function, while the potentially complex or unknown conditions that lead to the event do not have to be explicitly addressed or even understood; the optimal policy will tend to avoid the events regardless. In addition, the POMDP framework leverages all available information. The use of an explicit transition model allows for the application of airspace encounter models to the underlying CAS logic. Last, the POMDP framework is very general; a POMDP-based CAS is not tailor-made for a particular sensor system or aircraft platform. Such a CAS could be used on a variety of aircraft with different sensor systems. New sensor suites

can be integrated into or removed from the observation model, which is particularly attractive when an aircraft is equipped with multiple sensors, such as EO/IR sensors, radar, and GPS or undergoes frequent upgrades.

1.3.1 POMDP Solution Methods

POMDP solution methods may be divided into two groups, involving offline and online POMDP solvers. While exact solution methods exist (Cassandra et al., 1994), many methods only approximate the optimal policy.

Offline solvers require large computation time up front to compute the optimal policy for the full belief space. The agent then consults this policy online to choose actions while progressing through the state space. Offline solvers typically require discrete POMDP formulations. These discrete algorithms take advantage of the structure of the value function (the metric to be maximized) in order to efficiently approximate the value function within some error bound. This type of solution method has several drawbacks. First of all, the method is insensitive to changes in the environment because the policy is determined ahead of time. Second, the state space of many problems is too rich to adequately represent as a finite set of enumerable states (Ross et al., 2008).

Examples of offline solvers include Point-Based Value Iteration (PBVI), which was applied to *Tag*, a scalable problem in which the agents must find and tag a moving opponent (Pineau et al., 2003), in addition to other well-known scalable problems from the POMDP literature. PBVI and another offline solver, Heuristic Search Value Iteration (HSVI), were applied to *RockSample*, which is a scalable problem in which a rover tries to sample rocks for scientific exploration, in addition to other well-known scalable problems found in the POMDP literature, such as *Tiger-Grid* and *Hallway* (Smith and Simmons, 2004). HSVI was found to be significantly faster than PBVI in large problems. Successive Approximation of the Reachable Space under Optimal Policies (SARSOP) was applied to robotic tasks, such as underwater navigation and robotic arm grasping (Kurniawati et al., 2008). SARSOP performed significantly faster than the other value iteration algorithms. Offline POMDP solution methods

have recently been applied to the aircraft collision avoidance problem (Kochenderfer, 2009). In particular, this work applied the HSVI and SARSOP discrete POMDP algorithms to the problem. These discrete methods require a reduction in the dimensionality of the state space, which inherently results in a loss of information that is useful for the prediction of future states of intruder aircraft.

Online algorithms address the shortcomings of offline methods by only planning for the current belief state. As opposed to planning for all possible situations (as is the case with offline solvers), online algorithms only consider the current situation and a small number of possible plans. Online algorithms are able to account for changes in the environment because they are executed once at each decision point, allowing for updates between these points. In addition, because online algorithms are not solving the complete problem, they do not require a finite state space (as do discrete solvers). Consequently, these algorithms are able to use real-value representations of the state space and are referred to as real-value POMDP solvers. This capability is significant for the aircraft collision avoidance problem because of the large size of the belief space. The use of real values for state variables permits the integration of a wide range of transition functions and sensor models and allows the algorithm to plan for any possible scenario. Attempts to use small enough discretizations to permit the use of such models would cause the problem to be intractable for discrete solution methods.

Paquet's Real-Time Belief Space Search (RTBSS) is an online algorithm that has been compared to HSVI and PBVI on the *Tag* and *RockSample* problems (Paquet et al., 2005b). RTBSS is shown to outperform (achieve greater reward than) HSVI and PBVI in *Tag* and to perform orders of magnitude faster than these offline algorithms. In *RockSample*, RTBSS performs comparably to HSVI for small problems and outperforms HSVI in large problems.

1.3.2 Online Solution Methods

Two online POMDP solution methods are of particular relevance to this thesis and are mentioned here. First, Real-Time Belief Space Search (RTBSS) (Paquet et al.,

2005b) is an online algorithm that yields an approximately optimal action in a given belief state. Paquet uses a discrete POMDP formulation: a finite set of state variables, each with a finite number of possible values, and a finite set of possible observations and actions. The algorithm essentially generates a search tree, which is formed by propagating the belief state a predetermined depth, D, according to each possible action in each possible reachable belief state. It searches this tree depth-first using a branch and bound method to prune suboptimal subtrees.

Real-time POMDP approaches have been applied to aircraft collision avoidance in the past. Winder (2004) 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 CAS can have an acceptable probability of unnecessary alerts compared to other CAS logic methods.

Thrun (2000) applies a Monte-Carlo approach to POMDP planning that relies upon a sample-based belief state representation. This method permits continuous state and action space representations and non-linear, non-Gaussian transition models.

Monte-Carlo sampling has been used to generate intruder trajectories for aircraft collision avoidance (Yang, 2000). Poisson processes were used to describe the evolution of heading and altitude changes and the resulting probabilistic model of intruder trajectories was used to compute the probability of a future conflict. A CAS could then use this probability to decide whether or not to issue an alert. This process could be executed real-time.

1.4 Proposed Solution

This thesis introduces the Monte Carlo Real-Time Belief Space Search (MC-RTBSS) algorithm. This is a novel, online, continuous-state POMDP approximation algo-

rithm, that may be applied to aircraft collision avoidance. The algorithm combines Paquet's RTBSS with Thrun's belief state projection method for sample-based belief state representations. The result is a depth-first, branch-and-bound search for the sequence of actions that yields the highest discounted future expected rewards, according to some predefined reward function.

To reduce the computation time, the algorithm attempts to prune sub-trees by using the reward function and a heuristic function to maintain a lower bound for use with a branch and bound method. In addition, the algorithm prunes larger subtrees by using a sorting function to attempt to arrange the actions in order of decreasing expected value. The algorithm uses the resulting ordered set of actions to explore the more promising actions (from a value maximizing perspective) first.

The most significant difference between the new MC-RTBSS presented here and the prior RTBSS is the method of belief state representation. RTBSS assumes a finite number of possible state variable values, observations values, and actions. The algorithm then begins to conduct a search of all possible state trajectory-observation pairs. MC-RTBSS, on the other hand, does not assume a finite number of state variable values or observations (though all possible actions are also assumed to be predefined). MC-RTBSS uses a sample-based state representation to represent the belief state as a collection of weighted samples, or particles, where each particle represents a full state consisting of numerous real-valued state values. Each particle is weighted according to an observation noise model. Instead of iterating through each possible future state-observation combination for each state-action pairing, MC-RTBSS generates a specified number of noisy observations, according to an observation model, which is then used to assign weights to the belief state particles. The actual belief state is updated using a particle filter.

1.5 Thesis Outline

This thesis presents MC-RTBSS as a potential real-time algorithm for use in unmanned aircraft collision avoidance systems. This chapter introduced recent methods

of addressing aircraft collision avoidance and discussed methods of CAS analysis. The POMDP framework was suggested as a viable solution to the aircraft avoidance problem for unmanned aircraft. This chapter also provided motivation for using MC-RTBSS instead of other POMDP solution methods.

Chapter 2 presents a formal overview of the POMDP framework and a more detailed description of POMDP solution methods. The chapter also discusses Monte-Carlo POMDPs and RTBSS.

Chapter 3 explains the MC-RTBSS algorithm and discusses its implementation in the aircraft collision avoidance application domain. The chapter describes the notation and equations used in the MC-RTBSS transition and observation models as well as the metrics used in the reward function.

Chapter 4 presents results of single encounter scenario simulations with varying parameter settings in addition to the results of parameter sweeps on a collections of encounter simulations.

Chapter 5 presents a summary of this work, conclusions, and suggested further work.

Chapter 2

Partially-Observable Markov Decision Processes

A partially-observable Markov decision process (POMDP) models an autonomous agent interacting with the world. The agent receives observations from the world and uses these observations to choose an action, which in turn affects the world. The world is only "partially observable," meaning that the agent does not receive explicit knowledge of the state; it must use noisy and incomplete measurements of the state and knowledge of its own actions to infer the state of the world. In order to accomplish this task, the agent consists of two parts, as shown in Figure 2-1: a state estimator (SE) and a policy (π). The state estimator takes as input the most recent estimate of the state of the world, called the belief state, b; the most recent observation; and the most recent action and updates the belief state. The belief state is a probability distribution over the state space that expresses the agent's uncertainty as to the true state of the world. The agent then chooses an action according to the policy, which maps belief states to actions. After each time step, the agent receives a reward, determined by the state of the world and the agent's action. The agent's ultimate goal is to maximize the sum of expected discounted future rewards (Kaelbling et al., 1998). The state progression of the world is assumed to be Markovian. That is, the probability of transitioning to some next state depends only on the current state (and action); the distribution is independent of all previous states, as shown in Equation

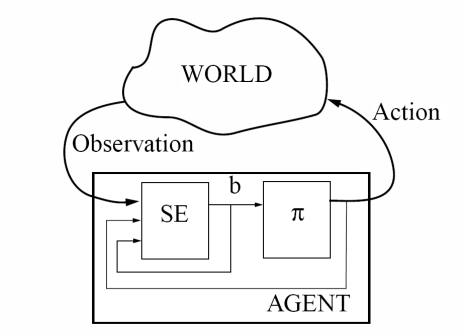


Figure 2-1: POMDP model (Kaelbling et al., 1998).

2.1.

$$P(s_{t+1} \mid a_t, s_t, a_{t-1}, s_{t-1}, ..., a_1, s_1) = P(s_{t+1} \mid a_t, s_t) \tag{2.1}$$

2.1 POMDP Framework

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 (Kaelbling et al., 1998). The state-transition function, $T(s, a, s')$, yields a probability distribution over states representing the probability that the agent will end in state s', given that it starts in state s and takes action a. The reward function, $R(s, a)$, yields the expected immediate reward the agent receives for taking action a from state s. The observation function, $O(s', a, o)$, yields a probability distribution over observations representing the probability that the agent receives observation o after taking action a and ending up in state s'. The framework is summarized in Table 2.1.

2.2 Representing Uncertainty

The agent's belief state, b, represents the agent's uncertainty about the state of the world. The belief state is a distribution over the states that yields the probability that the agent is in state s. At time t, the belief that the system state s_t is in state

Table 2.1: POMDP framework

Notation	Meaning
S	set of states
A	set of actions
$T(s, a, s')$	state-transition function, $S \times A \rightarrow \Pi(S)$
$R(s, a)$	reward function, $\mathcal{S} \times A \rightarrow \Re$
Ω	set of possible observations
$O(s', a, o)$	observation function, $S \times A \rightarrow \Pi(\Omega)$

s is given by

$$b_t(s) = P(s_t = s) \tag{2.2}$$

The belief state is updated each time the agent takes an action and each time the agent receives an observation, yielding the new belief state, b', calculated as:

$$\begin{aligned} b'(s') &= P(s' \mid o, a, b) && (2.3)\\ &= \frac{P(o \mid s', a, b)P(s' \mid a, b)}{P(o \mid a, b)} && (2.4)\\ &= \frac{P(o \mid s', a)\sum_{s \in S} P(s' \mid a, b, s)P(s \mid a, b)}{P(o \mid a, b)} && (2.5)\\ &= \frac{O(s', a, o)\sum_{s \in S} T(s, a, s')b(s)}{P(o \mid a, b)} && (2.6) \end{aligned}$$

The denominator, $P(o \mid a, b)$, serves as a normalizing factor, ensuring that b' sums to unity. Consequently, the belief state accounts for all of the agent's past history and its initial belief state (Kaelbling et al., 1998).

2.3 The Optimal Policy

As stated earlier, the agent's ultimate goal is to maximize the expected sum of discounted future rewards. The agent accomplishes this by choosing the action that maximizes the value of its current state. The value of a state is defined as the sum of the immediate reward of being in that state and taking a particular action and the discounted expected value of following some policy, π, thereafter, as shown in

Equation 2.7, in which the information inherent in the observation and observation model is utilized:

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \sum_{o \in \Omega} O(s', \pi(s), o) V_\pi(s') \tag{2.7}$$

where π is a policy, and $\pi(s)$ represents the specified action while in state s (i.e. $\pi(s)$ is a mapping of the current state to an action choice). The optimal policy, π^* maximizes $V_\pi(s)$ at every step, yielding $V^*(s)$:

$$\pi^*(s) = \arg\max_{a \in A} [R(s, a) + \gamma \sum_{s' \in S} T(s, \pi^*(s), s') \sum_{o \in \Omega} O(s', \pi^*(s), o) V^*(s')] \tag{2.8}$$

$$V^*(s) = \max_{a \in A} [R(s, a) + \gamma \sum_{s' \in S} T(s, \pi^*(s), s') \sum_{o \in \Omega} O(s', \pi^*(s), o) V^*(s')] \tag{2.9}$$

However, in most problems the value function is unknown (if it were known, then the problem would be solved). In the case of a POMDP, the agent never knows exactly which state it is in; the agent only has knowledge of its belief state, and hence, 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, weighted by the belief state, as shown in Equation 2.10.

$$R(b, a) = \sum_{s \in S} b(s) R(s, a) \tag{2.10}$$

The value of a belief state is then computed:

$$V_\pi(b) = R(b, \pi(b)) + \gamma \sum_{o \in \Omega} P(o \mid b, \pi(b)) V_\pi(b'_o) \tag{2.11}$$

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, shown in Equation 2.13.

$$V^*(b) = \max_{a \in A}[R(b,a) + \gamma \sum_{o \in \Omega} P(o \mid b,a) V^*(b'_o)] \tag{2.12}$$

$$\pi^*(b) = \arg\max_{a \in A}[R(b,a) + \gamma \sum_{o \in \Omega} P(o \mid b,a) V^*(b'_o)] \tag{2.13}$$

This solution could be reached in finite problems by iterating through every possible combination of actions, future belief states, and observations until some stop point, and then by determining the optimal policy after all of the rewards and probabilities have been determined. However, this process would be unfeasible in even moderately sized POMDPs. Consequently, other solution methods have been devised.

2.4 Offline Algorithms: Discrete POMDP Solvers

Discrete POMDP solution methods, such as Heuristic Search Value Iteration (HSVI) (Smith and Simmons, 2004), Point-based Value Iteration (PBVI) (Pineau et al., 2003), and Successive Approximation of the Reachable Space under Optimal Policies (SARSOP) (Kurniawati et al., 2008) require finite state and action spaces. These methods enumerate each unique state (i.e. each possible combination of state variable values) and each unique observation. This discretization allows the transition, observation, and reward functions to be represented as matrices. For example, the transition matrix for a given action maps a pair of states to a probability (e.g., the i, jth entry represents the probability of transitioning from the ith state to the jth state). The observation probabilities and reward values can be expressed similarly. 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.

2.5 Online Algorithms: Real-Value POMDP Solvers

One significant limitation of a discrete POMDP representation is that the state space increases exponentially with the number of state variables, which gives rise to two significant issues for the aircraft collision avoidance problem: the compact representation of the belief state in a large state space and the real-time approximation of a POMDP solution. Paquet's Real-Time Belief Space Search (RTBSS) is an online lookahead algorithm designed to compute a POMDP solution. However, the problem of adequate belief state representation still remains. Because of the nature of the Bayesian encounter model and its utilization, the belief state may not necessarily be represented merely by a mean and variance, which would allow for the use of a Kalman filter for belief state update. Thrun's sample-based belief state representation allows for the adequate representation of the full richness of the belief state in the aircraft collision avoidance problem, while his particle projection algorithm provides a method to predict future sample-based belief states.

2.5.1 Real-Time Belief Space Search

Paquet's RTBSS is an online algorithm that yields an approximately optimal action in a given belief state. Paquet assumes a discrete POMDP formulation: a finite set of state variables, each with a finite number of possible values, and a finite set of possible observations and actions. In order to reduce the computation time of traditional POMDP approximation methods (e.g. discrete solvers), Paquet uses a factored representation of the belief state. He assumes that the all of the state variables are independent, which allows him to represent the belief state by assigning a probability to each possible value of each variable, where the the probabilities of the possible values for any variable sum to unity. This factorization provides for the ready identification of subspaces which cannot exist (i.e. they exist with probability zero). RTBSS uses this formulation to more efficiently search the belief space by not exploring those states which cannot possibly exist from the start of the search. The belief space can be represented as a tree, as in Figure 2-2, by starting at the current belief state, b_0, and

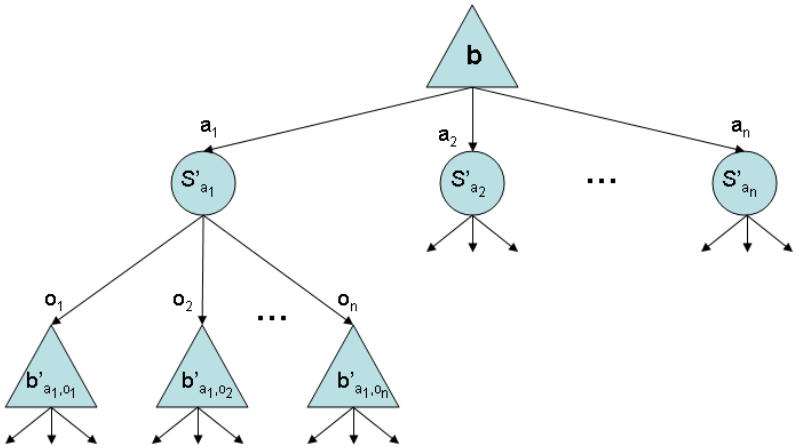


Figure 2-2: Example of a POMDP search tree (Paquet et al., 2005a).

considering each action choice, a_i, and the possible observations, o_j, which may be perceived afterward. Each action-observation combination will result in a new belief state at the next depth level of the tree. The process is then repeated at each node, resulting in a tree that spans the entire belief space up to some maximum depth level, D. The algorithm searches the tree to determine the optimal policy in the current belief state. At each depth level, RTBSS explores each possible belief state, which depends on (and is calculated according to) the possible perceived observations at that level. The algorithm uses the transition and observation models to determine the belief state variable value probabilities. The goal of the search is to determine which action (taken from the current belief state) yields the highest value. RTBSS uses a branch and bound method to prune subtrees and reduce computation time.

2.5.2 Monte Carlo POMDPs

Thrun's work with Monte Carlo POMDPs addresses the problem of POMDPs with a continuous state space and continuous action space. A sample-based representation of the belief state is particularly amenable to non-linear, non-Gaussian transition models. Thrun uses such a representation for model-based reinforcement learning in belief space. While MC-RTBSS is not a reinforcement learning algorithm because the observation, transition, and reward models are assumed to be known, Thrun's

particle projection algorithm is of particular interest. Particle projection is a method of generating a posterior distribution, or the future belief state, for a particular belief state-action pair. The original particle projection algorithm is modified slightly in MC-RTBSS to reduce computation time. Thrun's original procedure projects a new set of future states for each generated observation. The modified version is shown as Algorithm 1, used to propagate a current belief state forward to the next belief state, given some action. The algorithm projects N_p states, sampled from the current belief state, generates N_o random observations, and then weights the set of future states according to each observation, yielding N_o new belief states. The weights are normalized according to

$$w_n = \frac{p(s'_n)}{\sum_{n=1}^{N_p} p(s'_n)} \tag{2.14}$$

where the probabilities $p(s'_n)$ are calculated in line 12 of the particle projection algorithm.

```
1  Function {b'_{a,o_1}, ..., b'_{a,o_{N_o}}} = ParticleProject(b, a)
     Input:  b: The belief state to project forward.
2            a: The action.
3  for n = 1 : N_p do
4      sample s from b
5      sample s'_n according to T(s, a, ·)
6  end
7  for i = 1 : N_o do
8      sample x from b
9      sample x' according to T(x, a, ·)
10     sample o_i according to O(x', a, ·)
11     for n = 1 : N_p do
12         set particle weight: w_n = O(s'_n, a, o_i)
13         add ⟨s'_n, w'_n⟩ to b'_{a,o_i}
14     end
15     normalize weights in b'_{a,o_i}
16 end
17 return {b'_{a,o_1}, ..., b'_{a,o_{N_o}}}
```

Algorithm 1: ParticleProject

Chapter 3

Monte Carlo Real-Time Belief Space Search

This chapter discusses the Monte Carlo Real-Time Belief Space Search (MC-RTBSS) algorithm. The algorithm computes the approximately 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 and permit a real-time implementation.

Section 3.1 presents the belief state representation used by MC-RTBSS. Section 3.2 presents the MC-RTBSS pseudocode and subroutines. Section 3.3 describes the implementation of MC-RTBSS in the aircraft collision avoidance application domain.

3.1 Belief State Valuation

The belief state is a probability distribution over the current state of the system. 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 sum to

unity. At each time step, the agent must choose an action a from a set of possible actions, $A = \{a_1, ..., a_{|A|}\}$. The transition function, $T(s, a, s') = P(s' \mid s, a)$, specifies the probability of transitioning from the current state, s, to some state, s', when taking action a. At each time step, the agent receives an observation, o, based on the current state. The MC-RTBSS particle filter uses a known observation model, $O(s', a, o) = P(o \mid s, a)$, to assign weights to the particles in the belief state. In general, a reward function, $R(s, a)$, specifies the immediate reward the agent receives for being in a particular state s and taking an action a. MC-RTBSS uses its particle representation to approximate the immediate reward associated with a particular belief state:

$$R(b, a) = \sum_{i=1}^{N_p} w_i R(s_i, a) \tag{3.1}$$

The value of taking some action a_i in a belief state b, $V(b, a_i)$, is the expected discounted sum of immediate rewards when following the optimal policy from the belief state. If there are a finite number of possible observations, N_o, $V(b, a_i)$ may be expressed as:

$$V(b, a) = R(b, a) + \gamma \max_{a' \in A} \sum_{o \in \Omega} P(o \mid b, a) V(b'_o, a') \tag{3.2}$$

where N_o is the number of generated observations, b'_o is the next belief state associated with the observation o, a' is the action taken at the next decision point, and γ is a discount factor (generally less than 1). MC-RTBSS is designed to be used with continuous observation spaces that cannot be explicitly enumerated, and so relies upon sampling and a small collection of observations from the distribution $P(o \mid b, a)$. Equation 3.2 is then approximated by:

$$V(b, a) = R(b, a) + \gamma \frac{1}{N_o} \max_{a' \in A} \sum_{o \in \Omega} V(b'_o, a') \tag{3.3}$$

Finally, a heuristic utility function, $U(b, a)$, provides an upper bound on the value of being in a particular belief state b and taking a particular action a. As discussed later, MC-RTBSS uses $U(b, a)$ for pruning in its branch and bound method.

3.2 Implementation

MC-RTBSS is implemented in Algorithm 2. The algorithm uses EXPAND to approximate the optimal action (line 5), which the agent executes before perceiving the next observation and updating the belief state. The initial belief state is represented by b_0.

```
1  Function OnlinePOMDPAlgorithm()
     Static:   b:       The current belief state.
               D:       The maximum search depth.
2              action:  The best action.
3  b ← b_0
4  while simulation is running do
5      Expand(b, D)
6      a ← action
7      Execute a
8      Perceive new observation o
9      b ← ParticleFilter(b, a, o)
10 end
```

Algorithm 2: OnlinePOMDPAlgorithm

MC-RTBSS is implemented as a recursive function that searches for the optimal action at each depth level of the search. The temporal significance (with respect to the environment) of each depth level is determined by the temporal length of an action, because searching one level deeper in the tree corresponds to looking ahead to the next decision point, which occurs when the action is scheduled to be complete. Consequently, the temporal planning horizon of the search is limited to:

$$t_{max} = t_0 + D \times actionLength \tag{3.4}$$

where t_0 is the time at the initial decision point and D is the maximum depth of the search, which is usually limited by computational constraints.

The belief state at the current decision point is the root node of the aforementioned search tree. MC-RTBSS keeps track of its depth in the search with a counter, d, which is initialized to D and decremented as the depth increases. Thus, at the top of the

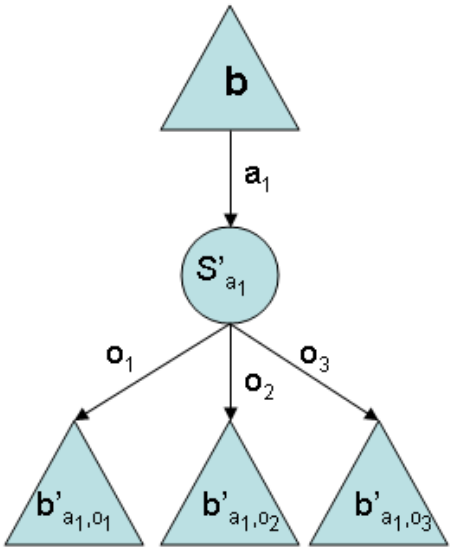


Figure 3-1: Belief state projection example.

search tree $d = D$ and at each leaf node $d = 0$.

3.2.1 Belief State Projection

In order to expand a node, MC-RTBSS executes a modified version of Thrun's belief state projection procedure. The MC-RTBSS belief state projection pseudocode is shown in Algorithm 1, in Section 2.5.2. 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 $O(s, a, \cdot)$.

This process is illustrated in Figure 3-1. In this example, the search begins at the current belief state b, with MC-RTBSS expanding this node with trial 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.

3.2.2 Particle Filtering

Belief state update is accomplished using a particle filter. The particle filter algorithm used in this work, ParticleFilter is shown in Algorithm 3. 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. The particle filter is similar to the ParticleProject, except that it weights the particles according to the perceived observation. N_{PF} is the number of particles used in the particle filter.

```
1  Function b' = ParticleFilter(b, a, o)
     Input:  b:  The current belief state.
             a:  The action.
2            o:  The observation.
3  for n = 1 : N_PF do
4      sample s from b
5      sample s'_n according to T(s, a, ·)
6      set particle weight: w_n = O(s'_n, a, o)
7      add ⟨s'_n, w'_n⟩ to b'_{a,o}
8  end
9  normalize weights in b'_{a,o}
10 return b'
```

Algorithm 3: ParticleFilter

3.2.3 MC-RTBSS Recursion

As mentioned earlier, MC-RTBSS is implemented as a recursive function, called Expand, that terminates at some specified depth. The function takes a belief state and a value for its depth, d, as arguments. The function returns both a lower bound on the value of an action choice and an action. After the initial function call, the algorithm is only concerned with the returned value for a given belief state at some depth value $(d-1)$, which it uses to compute the value of the belief state at the previous depth level (d), according to:

$$V(b,a) = R(b,a) + \gamma \frac{1}{N_o} \sum_{i=1}^{N_o} \textsc{Expand}(b'_{a,o_i}, d-1) \tag{3.5}$$

3.2.4 Pseudocode

The full pseudocode of the MC-RTBSS algorithm is introduced in Algorithm 4.

```
1  Function L_T(b) = EXPAND(b, d)
     Input:   b:       The current belief state.
2            d:       The current depth.
     Static:  action:  The best action.
              L:       A lower bound on V*.
              U:       An upper bound on V*.
3  action ← null
4  if d = 0 then
5      L_T(b) ← L(b)
6  else
7      Sort actions {a_1, a_2, ..., a_|A|} such that U(b, a_i) ≥ U(b, a_j) if i ≤ j
8      i ← 1
9      L_T ← −∞
10     while i ≤ |A| and U(b, a_i) > L_T(b) do
11         {b'_{a_i,o_1}, ..., b'_{a_i,o_{N_o}}} =PARTICLEPROJECT(b, a_i)
12         L_T(b, a_i) ← R(b, a_i) + γ (1/N_o) Σ_{i=1}^{N_o} EXPAND(b'_{a_i,o_i}, d − 1)
13         if L_T(b, a_i) > L_T(b) then
14             action ← a_i
15             L_T(b) ← L_T(b, a_i)
16         end
17         i ← i + 1
18     end
19 end
20 return L_T(b)
```

Algorithm 4: EXPAND

In the first D iterations, MC-RTBSS recurs down the first branch of the search tree in a similar manner to a depth-first search. However, when it reaches the first leaf node (when $d = 0$, node c in Figure 3-2), the algorithm cannot expand any further. As seen in line 5 of EXPAND, at this point MC-RTBSS calculates a lower bound of the belief state at the leaf node ($R(C)$ in Figure 3-2) and sets it as the lower bound of the search, L_T. The algorithm then returns this value to its instantiation at the previous depth level ($d = 1$).

The algorithm repeats this process for all of the leaf nodes that are children of the node at depth $d = 1$ (node b'_{a_1,o_1} in Figure 3-2), keeping track of all of these values,

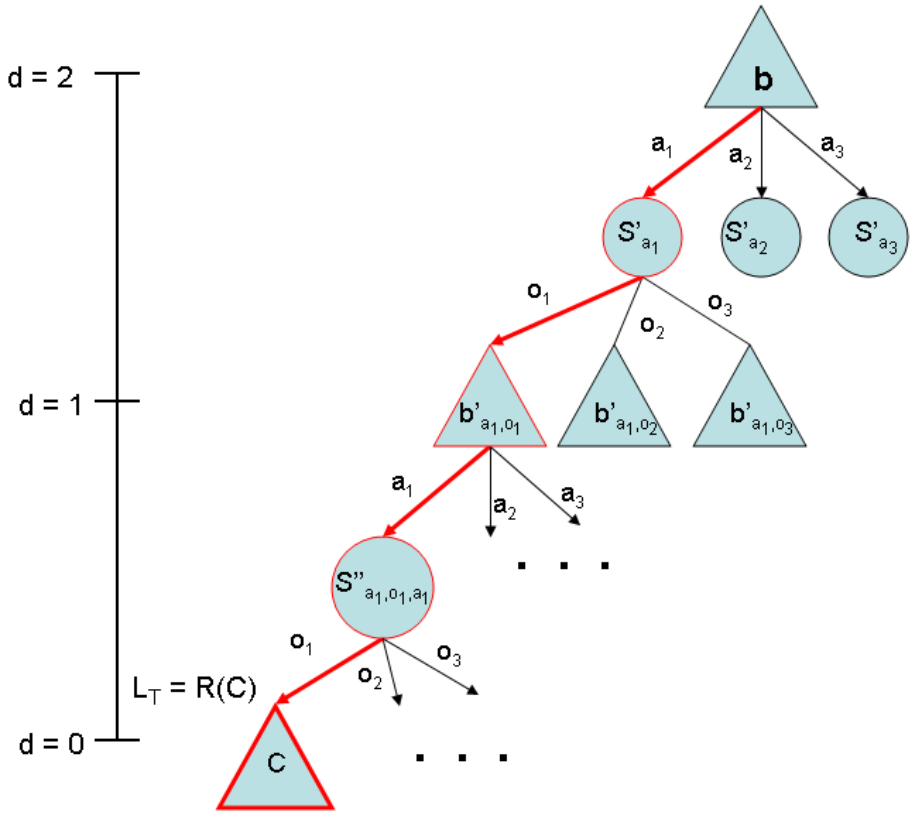


Figure 3-2: EXPAND example.

before it reaches line 12, at which point it is able to compute the value associated with taking action a_1 in b'_{a_1,o_1}.

Now that the global lower bound, L_T has been updated to something greater than $-\infty$, the reason for using an upper bound of the actual value of a belief state for the utility function becomes apparent. The result of this evaluation the next time around ($U(b, a_2)$, in the case of the example) is compared against the global lower bound for pruning. Wasting time exploring a suboptimal policy is preferable to pruning the optimal policy, thus it is safer (from an optimality standpoint) to overestimate the value of a leaf node than to underestimate it. The search then iterates through the remaining candidate actions, repeating a similar process unless $U(b, a_i) \not> L_T$, in which case the node is not expanded for the ith action, effectively pruning the subtree associated with that action. If an action is not pruned and the lower bound obtained from choosing that action is greater than the global bound, then the new action is recorded as the best action, and the global lower bound is updated (line 13).

Until now, the assumption has been that the utility has been computed and that the candidate actions are iterated through in some arbitrary order. However, a sorting procedure is used to compute the utilities of choosing each a while in a given b, and then to order the list of candidate actions in order of decreasing utility. The idea here is that if $U(b, a_i) > U(b, a_j)$, then a_i is more likely to yield a larger value than a_j (i.e. a_i is more likely to be part of the optimal policy). If this is indeed the case and $V(b_{a_i}) > U(b, a_j)$, then the subtree associated with a_j and all other a_k's (where $k > j$) will be pruned because U yields an upper bound on $V(b_a)$ and $U(b, a_j) \geq U(b, a_k)$ (by definition of the sorting function), potentially saving computation time.

After completing the search, the initial instantiation of the search (at $d = D$) returns both the value approximation of the initial belief state b and the optimal action a.

3.2.5 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

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 $O((N_p + N_{sort}|A|)(N_o|A|)^D)$.

3.3 Collision Avoidance Application Domain

This section explains how MC-RTBSS was applied to the problem of collision avoidance for unmanned aircraft.

3.3.1 Encounter Models

One of the strengths of the approach to aircraft collision avoidance pursued in this thesis is the leveraging of airspace encounter models, constructed from a large collection of radar data, to predict the future state of intruder aircraft. These encounter models are also used in this thesis to evaluate the performance of the algorithms. This section provides some background on encounter models and their construction.

Encounter Modeling Background

Historically, encounter models have been used by organizations such as the FAA and International Civil Aviation Organization (ICAO) to test CAS effectiveness across a wide range of encounter situations. The encounters generated by the models represent the behavior of aircraft during the final minute or so before a potential collision. The models assume that prior airspace safety layers, such as air traffic control advisories, have failed. The encounters are defined by a set of initial conditions for each aircraft and a scripted sequence of maneuvers to occur during simulation. The initial conditions consist of the initial positions, velocities, and attitudes of the aircraft, and the maneuvers (known as controls or events) specify accelerations and turn rates that are scheduled to occur at specific times during the simulation. The distributions from which these initial conditions and controls are drawn are based on radar data

(Kochenderfer et al., 2008c). A simulation then applies sensor and CAS algorithm models to the aircraft trajectories, allowing the CAS to issue avoidance maneuvers if appropriate, and propagates the aircraft states accordingly. Large numbers of encounters are generated and run in simulation to evaluate performance metrics such as the probability of a Near Mid-Air Collision (NMAC) or the risk ratio, which compares the probability of NMAC of different systems. In the collision avoidance community, an NMAC is an incident in which two aircraft have less than 500 ft horizontal separation and less than 100 ft vertical separation (Kuchar and Drumm, 2007).

Prior Encounter Models

MITRE initially developed an encounter model of the U.S. airspace in the early 1980's for the development and certification of TCAS in support of the U.S. mandate to equip large transport aircraft with the system (The MITRE Corporation, 1983). This two-dimensional model was used to simulate aircraft vertical motion. The ICAO and Eurocontrol then completed a three-dimensional aircraft model in 2001, which allowed for a single period of acceleration during each encounter. This model was used in support of the worldwide TCAS mandates (Aveneau and Bonnemaison, 2001). Beginning in 2006, new U.S. encounter models were developed for use in the evaluation of TCAS and future CAS for both manned and unmanned aircraft. These models involved collecting and processing data from 130 radars in the U.S. (Kochenderfer et al., 2008a,b,c; Edwards et al., 2009).

Correlated and Uncorrelated Encounters

The availability and presence of air traffic control (ATC) services, such as flight-following services, have an impact on aircraft behavior during encounters. For example, when both aircraft in an encounter each have a transponder and at least one aircraft is in contact with ATC, then ATC is likely tracking both aircraft and at least one of the aircraft will probably receive notification about nearby traffic. This aircraft may then begin to act accordingly in order to avoid the traffic conflict before any CAS becomes involved. The ATC intervention in the encounter likely re-

sults in some correlation between the two aircraft trajectories. An airspace encounter model that captures such statistically related behavior is called a correlated encounter model. Conversely, a different, uncorrelated encounter model captures the behavior of aircraft in encounters in which there is no prior ATC intervention. This type of encounter includes situations where two aircraft are flying under visual flight rules (VFR) without ATC flight-following or where one of the aircraft is not equipped with a transponder. In these encounters, pilots must visually acquire the other aircraft at close range or use some other CAS in order to avoid collision, generally resulting in uncorrelated trajectories. The uncorrelated encounter model captures this type of behavior by randomly propagating the aircraft trajectories based solely on the statistical characteristics of the individual aircraft (Kochenderfer et al., 2008c).

Encounter Model Construction and Implementation

Encounter models are used to specify the values of certain state variables during simulation. Both the correlated and uncorrelated encounter models describe the true airspeeds, airspeed accelerations, vertical rates, and turn rates of each aircraft involved in an encounter. The models also include environmental variables for aircraft altitude layer and airspace class. The correlated model includes the approach angle, horizontal miss distance, and vertical distance at the time of closest approach (Kochenderfer et al., 2008a).

The encounter models use Markov processes to describe how the state variables change over time. A Markov process assumes that the probability of a specific future state only depends on the current state. In short, the process assumes the future is independent of the past. Dynamic Bayesian networks are then used to express the statistical interdependencies between variables (Neapolitan, 2004). As an example, the structure of the dynamic Bayesian network used for the uncorrelated encounter model is shown in Figure 3-3.

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

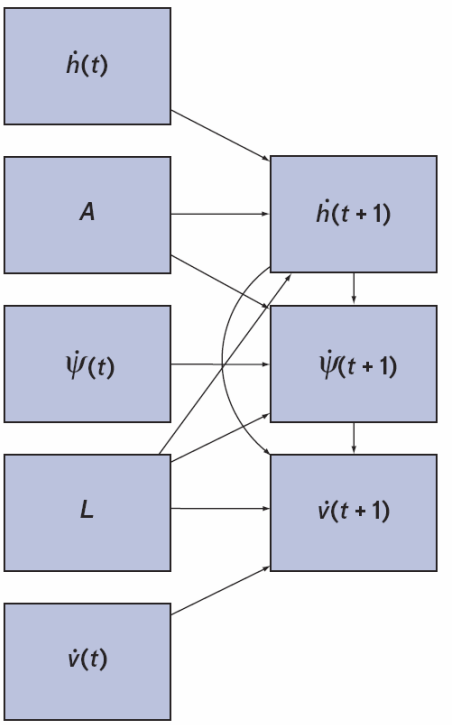


Figure 3-3: Dynamic Bayesian network framework for the uncorrelated encounter model (Kochenderfer et al., 2008b).

encounter and do not vary with time. The first set of variables (on the left of Figure 3-3) represents the variable values at the current time (time t). The second set of variables (on the right of Figure 3-3) represents the variable values at the next time step (time $t + 1$). Each of the variables in the second set (the variable values at the next time step) have an associated conditional probability table. The values of these tables are determined by statistics derived from collected radar data. The dynamic Bayesian network is then used to generate encounter trajectories by sampling from the network according to the conditional probability tables (in accordance with the previous variable values) and projecting the aircraft states forward accordingly. The particular structure used for the encounter models was not chosen haphazardly. The structure was chosen and optimized according to a quantitative metric, the Bayesian scoring criterion, which is related to the likelihood that the set of radar data would be generated from the network (Kochenderfer et al., 2008b; Neapolitan, 2004).

3.3.2 Simulation Environment

MC-RTBSS was integrated into existing fast-time simulation infrastructure, MIT Lincoln Laboratory's Collision Avoidance System Safety Assessment Tool (CASSATT). CASSATT is implemented in the Simulink environment, permitting the easy application of specific aircraft dynamics models and limits as well as the incorporation of different sensor models and CAS algorithms. Up to millions of encounters are generated using an encounter model and simulated in CASSATT using a parallel computing cluster to test the performance of various collision avoidance algorithms (Kochenderfer et al., 2008c). The dynamic simulation operates at 10 Hz, while MC-RTBSS operates at 1 Hz, beginning at $t = 4$ s. The first 3 seconds are used to generate an initial belief state, described later, which is fed into the particle filter to be updated using the latest observation. The resulting belief state is then fed into the MC-RTBSS algorithm. The uncorrelated airspace encounter model represents encounters in which aircraft must rely solely on the ability to "see-and-avoid" intruders. These situations present one of the greatest challenges to integrating unmanned aircraft into the U.S. airspace.

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

$$s = \langle v_1, N_1, E_1, h_1, \psi_1, \theta_1, \phi_1, \dot{v}_1, \dot{h}_1, \dot{\psi}_1, v_2, N_2, E_2, h_2, \psi_2, \theta_2, \phi_2, \dot{v}_2, \dot{h}_2, \dot{\psi}_2, t \rangle \quad (3.6)$$

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 3.1.

Table 3.1: 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

Observations

An observation is comprised of the following four elements:

$$o = \langle r, \beta, h_1, h_2 \rangle \tag{3.7}$$

The definition and units for each are shown in Table 3.2.

Table 3.2: Observation variables

Variable	Definition	Units
r	slant range	ft
β	bearing	rad
h_1	own altitude	ft
h_2	intruder altitude	ft

Bearing is measured clockwise from the heading of the own aircraft. The range and bearing can be determined from the state variables using the following operations:

$$r = \sqrt{(N_2 - N_1)^2 + (E_2 - E_1)^2 + (h_2 - h_1)^2} \tag{3.8}$$

$$\beta = \text{arctan2}(E_2 - E_1, N_2 - N_1) - \psi_1 \tag{3.9}$$

where arctan2 is the four-quadrant inverse tangent, whose range is the interval $[-\pi,\pi]$.

Actions

The set of all possible actions, A, contains 6 actions at any given decision point. Each action, in turn, consists of a sequence of 5 commands, each of which is meant to be executed each second, beginning at a decision point and finishing 5 seconds later, should that action be chosen. In the simulations conducted in this project, one of the 6 candidate actions is always the nominal set of commands for that 5 seconds of time in the simulation (i.e. the 5 entries describing the aircraft's default behavior, sampled from the encounter model). Every other candidate action has the same acceleration and turn rate ($\dot{v}$ and $\dot{\psi}$, respectively) as the nominal commands, but the vertical rate ($\dot{h}$) is different. The vertical rates for all of the actions are described in Table 3.3.

Table 3.3: Action climb rates

Action	**Climb Rate (ft/min)**
a_1	2000
a_2	1500
a_3	0
a_4	-1500
a_5	-2000
a_6	scripted

Transition Function

The transition function, T, takes a state and an action as its arguments and deterministically propagates the own ship state variables forward by using simple Euler integrations, shown below, using the chosen action to determine the rates at each time step ($\Delta t = 1\,\mathrm{s}$).

$$v_{n+1} = v_n + \dot{v}_{n+1}\Delta t \tag{3.10}$$

$$h_{n+1} = h_n + \dot{h}_{n+1}\Delta t \tag{3.11}$$

$$\psi_{n+1} = \psi_n + \dot{\psi}_{n+1}\Delta t \tag{3.12}$$

The transition function calculates the flight path angle θ at each instant using the trigonometric relations of other state variables at that instant.

$$\theta = \sin^{-1} \frac{\dot{h}}{v} \tag{3.13}$$

The transition function propagates aircraft location through space by determining the horizontal component of velocity, $v_{horizontal}$, and then breaking this vector into its North and East components ($\dot{N}$ and $\dot{E}$, respectively). These components are then used in an Euler integration to propagate the aircraft in the respective directions. The transition function does not model vertical acceleration ($\ddot{h}$), bank angle (ϕ), angular rates (p,q,r) or accelerations ($\dot{p}, \dot{q}, \dot{r}$) and consequently, the transition function is unable to model aircraft performance limitations on these variables.

$$\begin{aligned} v_{horizontal} &= v \cos\theta & (3.14)\\ \dot{N} &= v_{horizontal} \cos\psi & (3.15)\\ \dot{E} &= v_{horizontal} \sin\psi & (3.16)\\ N_{n+1} &= N_n + \dot{N}\Delta t & (3.17)\\ E_{n+1} &= E_n + \dot{E}\Delta t & (3.18) \end{aligned}$$

The function calls another function, GenerateIntruderAction, which takes the state at a particular time step as its argument and uses the intruder action and altitude at some time t to sample the intruder action at time $t+1$ from the encounter model. The function repeats this procedure for the duration of the own ship action. The transition function returns a state trajectory of duration equal to the input action duration.

Observation Model

MC-RTBSS needs to be able to generate observations from the observation model and consult the posterior distributions (of the observation components) to determine weights for the samples that comprise a belief state. In order to achieve these goals,

two procedures were used, an observation sampling function and a probability density evaluation function. The observation sampling function takes a particular state as input and calculates the actual slant range, bearing, and aircraft altitudes. The function then applies a random number generator, using these actual values as means and predefined values for the standard deviations (determined by instrument noise models), to add noise to the mean values, yielding noisy observations. The mean values and standard deviations used in this implementation are shown in Table 3.4. These statistics reflect the TCAS sensor noise model as used in previous TCAS studies (Kuchar and Drumm, 2007).

Table 3.4: Observation noise model

Variable	**Mean Value**	**Standard Deviation**
r	r_{actual}	50 ft
β	β_{actual}	0.1745 rad
h_1	$h_{1,actual}$	50 ft
h_2	$h_{2,actual}$	50 ft

The observation probability density function takes a state sample and an observation as its arguments and outputs a weighting for the sample, which is an approximation of the observation posterior distribution. The function calculates the actual range, bearing, and altitudes of the state and then uses the probability density function to determine the likelihood of each individual component of the observation. The noise model assumes that each of the four components of an observation are independent. Hence, the joint posterior likelihood is calculated as the product of the likelihoods of each component of the observation. This calculation is shown in Equation 3.19, where the subscript o denotes the observation variable value (as opposed to the actual value, which has no o subscript attached). The result is used to weight the state sample and is eventually normalized to ensure that the particle weights for a belief state sum to unity.

$$p(o \mid s) = p(r_o \mid r)p(\beta_o \mid \beta)p(h_{1,o} \mid h_1)p(h_{2,o} \mid h_2) \tag{3.19}$$

Belief State Update and the Initial Belief State

The own ship location (E_1, N_1, h_1) is assumed to be fully observable from the start of the simulation; the intruder variables are partially observable. At every time step, during particle filtering, the own ship location is updated in every particle. Filtering then becomes a problem of tracking the intruder aircraft.

The initial belief state, b_0 is constructed in the first 3 seconds of simulation using the observations and knowledge of the own ship state. After the agent perceives the first observation, the agent estimates the intruder location $(E_2, N_2, \dot{h_2})$ using knowledge of its own location and the geometry of the observation. At this point, the observation is assumed to be noiseless. After the agent perceives the second observation, it again computes the intruder location and computes the intruder velocity, heading, and vertical rate (v_2, ψ_2, and h_2, respectively) from the change in location over time. The velocity and vertical rate are also used to determine the flight path angle, θ_2. After the agent perceives the third observation, the change in velocity is used to determine the intruder acceleration ($\dot{v_2}$).

Once the agent has estimated these variable values, the initial belief state is constructed by assigning these values to the belief state particles, which are weighted equally. The initial belief state is then passed into the particle filter for the first time.

Reward Function

The reward function, $R(s, a)$, penalizes the agent for both deviation from the nominal flight path and NMACs. The penalty for an NMAC is denoted λ. The reward function is shown in Equation 3.20

$$R(s,a) = \begin{cases} \text{deviation penalty} + \lambda & , \text{ if NMAC} \\ \text{deviation penalty} & , \text{ otherwise} \end{cases} \tag{3.20}$$

Although the agent reaches a decision point every 5 seconds during the simulation, MC-RTBSS maintains a record of the trajectories between decision points at one second increments, on the second. Due to the high velocity of aircraft, it is certainly

possible that the reward function could miss an NMAC in a 5 second window. In order to decrease the likelihood of the reward function missing such an event, the reward function evaluates each 5 second trajectory segment up to and including the state at the decision point in question at every 0.1 seconds of the trajectory.

In order to calculate deviation from the nominal flight path, the nominal flight path must be calculated prior to each encounter. This is accomplished by using the encounter model inputs and initial conditions to run a preliminary simulation in CASSATT with no active CAS on either aircraft. The coordinates of the resulting own ship trajectory are extracted from the simulation results and stored. This path is then fed back to the MC-RTBSS algorithm for use in the reward function. The reward function then calculates deviation for a given state by simply calculating the slant range between the own aircraft and where the aircraft ought to be at that time on the nominal flight path at every time step on the 5 second trajectory and then taking the mean of these deviation values over the 5 second trajectory. The deviation penalty for a 5 second trajectory is the average of the instantaneous deviations at each time step. The NMAC penalty is calculated in a similar manner. The horizontal and vertical separation between the two aircraft is checked to see whether they are less than the predefined NMAC thresholds at every time step on the 5 second trajectory. If the separation falls below the thresholds at any point, then a flag is triggered and the NMAC penalty, λ, is added to the total penalty for the trajectory. The total penalty is the reward associated with taking the particular action while in the initial state.

Heuristic Utility Function

The heuristic utility function, $U(b, a)$, provides an upper bound on the value of taking action a in belief state b. The function is used in the sorting procedure to order the set of action choices by descending utility value. The 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 at each of the resulting states 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 sum of all future returns and the reward function only penalizes (it never increases the value). Incorporating the value of future potential belief states will likely only reduce the one step computed value.

3.3.3 MC-RTBSS Parameters

The parameters of MC-RTBSS include the maximum search depth, D, the number of particles used in the internal belief-state representations, N_p, and the number of observations generated for each action-choice in the search tree, N_o. In addition, in the implementation of MC-RTBSS used for collision avoidance, two additional parameters include the NMAC cost, λ, and the number of particles used in the action sorting function, N_{sort}. The parameters are summarized in Table 3.5. This section describes the qualitative effects expected when varying these parameters. Because MC-RTBSS is intended for online use, computation time is of particular importance. Increasing the value of any parameter is associated with a significant trade off between performance and computation. As described earlier, computation time is bounded by $O((N_o|A|)^D(N_p))$.

Table 3.5: 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

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. In the context of collision

avoidance, increasing the number of particles increases the likelihood that MC-RTBSS can account for less likely future events, such as the intruder making an aggressive maneuver which might result in an NMAC, and act accordingly. The computation time per node in the search tree increases linearly with N_p.

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 different 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 (for a given parent belief state-action pair), which in turn increases the likelihood that MC-RTBSS will account for rarer future events in its decision-making. Because (assuming no pruning) the computation time grows exponentially with the product of N_o and the number of possible actions (depending exponentially on D), increasing N_o results in an even greater impact on the computation time than does increasing N_p.

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. MC-RTBSS compares the expected cost of deviating from the nominal path to the expected cost due to NMAC. Because MC-RTBSS makes decisions based on maximizing a function of expected rewards, λ affects how likely a safety event must be in order to significantly impact decision-making. An increase in λ means that particles leading to an NMAC have a larger impact on the reasoning than do other particles with higher weights. The NMAC cost has no direct effect on computation time, although a small λ value could conceivably raise the lower bounds of the value function at nodes with likely NMACs enough to reduce some pruning. However, increasing the value of λ too much could have undesirable effects as well. For a long enough horizon and a large enough

NMAC penalty, MC-RTBSS might be too cautious, resulting in excessive deviations to avoid improbable future NMACs. The value of λ is the only parameter that affects the reward function in this implementation.

Maximum Search Depth, D

The maximum search depth affects the planning horizon of the algorithm. The algorithm can only account for future states within the horizon. In the collision avoidance problem, MC-RTBSS will not maneuver to avoid an NMAC that occurs beyond the horizon. Assuming no pruning, the computation time grows exponentially with D and results in the greatest increase in computation time of all parameters.

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, $O(N_{sort}|A|)$, increases linearly with increasing N_{sort} as well.

Chapter 4

Simulation Results

The performance characteristics of the MC-RTBSS algorithm are evaluated in two ways. First, simple, level encounters, in which the own ship is equipped with MC-RTBSS, are simulated with various parameter settings. 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. Second, a set of 76 encounters, randomly generated from the encounter model, are simulated with no collision avoidance system, with TCAS, and then with MC-RTBSS with various parameter settings. These larger scale simulations demonstrate the effect of each parameter on the overall performance of the algorithm across a wider range of different situations.

Section 4.1 describes the simple scenarios used to demonstrate the effects of varying different algorithm parameters. Section 4.2 presents the results of the single encounter scenario simulations and explains the algorithm behavior. Section 4.3 presents the results of the parameter sweeps on the set of 76 random encounters.

4.1 Simple Encounter Cases

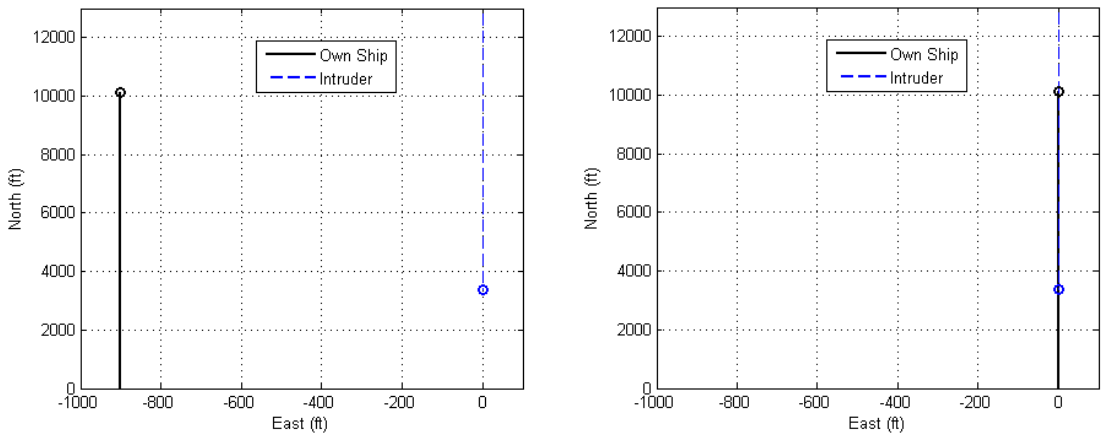
The first set of simulations involves single, co-altitude, level encounters. Two different scenarios are used to demonstrate different aspects of MC-RTBSS behavior. The own ship and intruder trajectories for both cases of this encounter are shown in Figure 4-1, where the circles designate the direction of travel of each aircraft (the circle is at

the last position of each aircraft). For both cases, the encounter begins at $t = 0\,\text{s}$, the duration is 30 seconds, and the closest point of approach (CPA) occurs 20 seconds into the encounter (the time of closest approach (TCA) is at $t = 20\,\text{s}$). The plot in Figure 4-1a shows an offset case, in which the own ship is offset 900 ft west of the intruder. No NMAC occurs in this case. The plot in Figure 4-1b is a head-on case, in which the aircraft are set to result in an NMAC. The initial conditions for both cases are tabulated in Table 4.1. 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 potentially maneuver at any time before CPA and eventually cause an NMAC. Figure 4-2 shows a an example of 30 intruder trajectories generated from the encounter model for the first 20 seconds 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 (vertical miss distance (VMD) is 69 ft and horizontal miss distance (HMD) is 4 ft at $t = 20\,\text{s}$). This scenario is designed to test whether MC-RTBSS can identify such possible trajectories and maneuver to ensure that an NMAC will very likely not 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 highly likely. This scenario is used for the D and N_{sort} trials.

Table 4.1: Single encounter initial conditions

Variable	**Head-On Case**	**Offset Case**
$v_1 = v_2$	338 ft/s	338 ft/s
$h_1 = h_2$	4500 ft	4500 ft
N_1	0 ft	0 ft
N_2	13520 ft	13520 ft
E_1	0 ft	-900 ft
E_2	0 ft	0 ft
ψ_1	0°	0°
ψ_2	180°	180°

Both TCAS-equipped and MC-RTBSS-equipped aircraft with different parame-



(a) Offset (b) Head-on

Figure 4-1: Single encounters, no CAS.

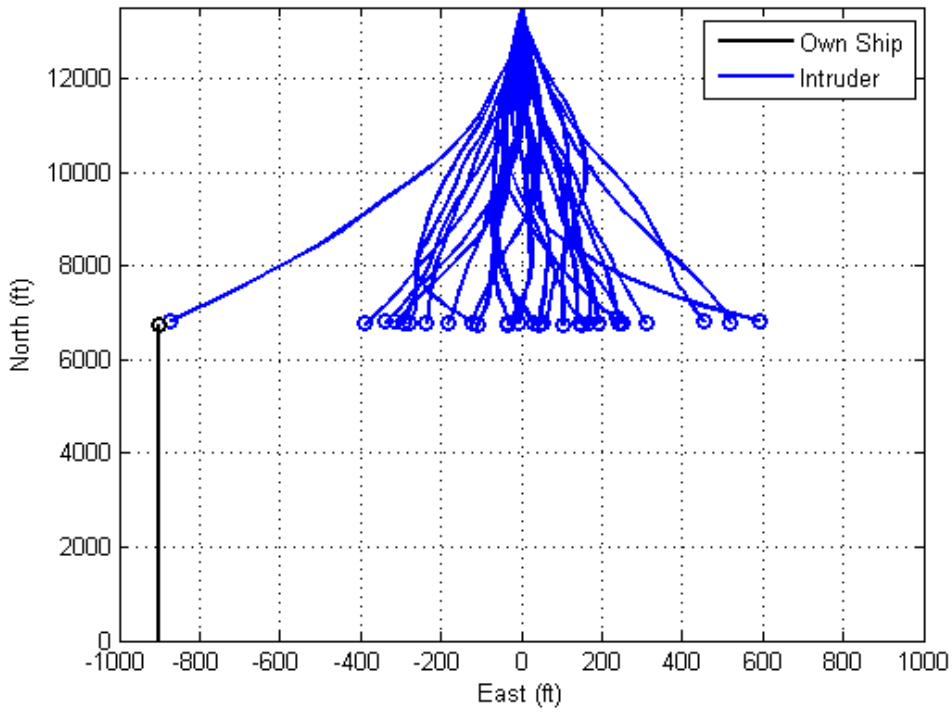


Figure 4-2: Example intruder trajectories from the encounter model, 20 second duration, offset scenario, no CAS.

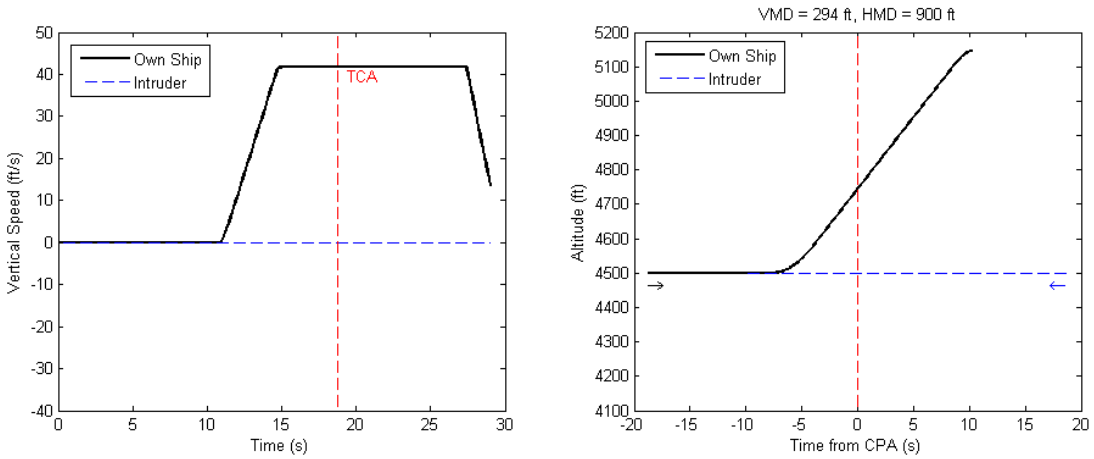
ter settings are simulated in the offset encounter for comparison. The MC-RTBSS parameter settings tested in this scenario are in Table 4.2. The last column lists a normalized worst case estimate on the upper bounds of complexity with each parameter setting. These numbers are the result of applying the parameter settings to the worst case complexity discussed in Section 3.2.5, $O((N_p + N_{sort}|A|)(N_o|A|)^D)$, 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 more computation time than the Small N_o simulation in the worst case of each.

Table 4.2: 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	**3000**	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	**1000**	912

4.2 Single Encounter Simulation Results

This section presents the single encounter simulation results for a TCAS-equipped aircraft and an MC-RTBSS-equipped aircraft with varying parameter settings.



(a) Vertical Speed (b) Altitude

Figure 4-3: Single encounter, TCAS.

4.2.1 TCAS

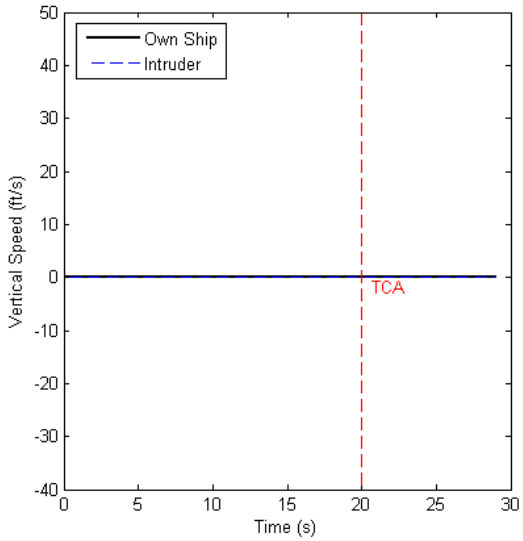
The result of the offset encounter simulation with a TCAS-equipped own aircraft is shown in Figure 4-3.

In this encounter, at $t = 11\,\text{s}$ TCAS issues a climb RA at $41.7\,\text{ft/s}$ and at $t = 27\,\text{s}$ TCAS issues clear of conflict and the aircraft begins to level off. The resulting miss distances are VMD of $294\,\text{ft}$ and HMD of $900\,\text{ft}$. TCAS issues the climb RA because the intruder is projected to violate the TCAS range and relative altitude thresholds at CPA.

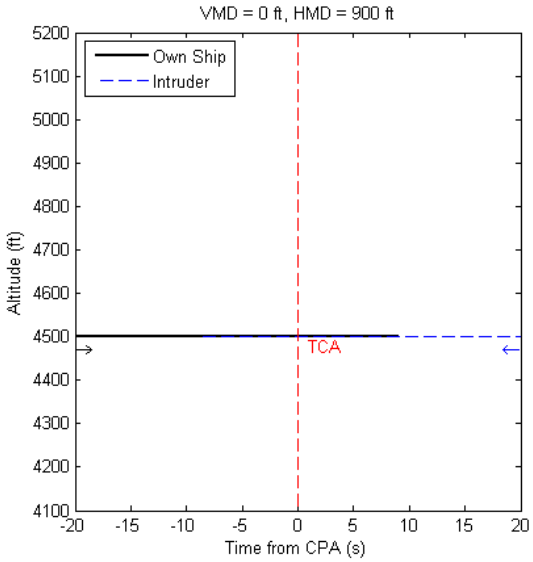
4.2.2 Varying Number of Particles, N_p

The results of an MC-RTBSS equipped aircraft with $N_p = 10$ and $N_p = 3000$ are shown in Figure 4-4.

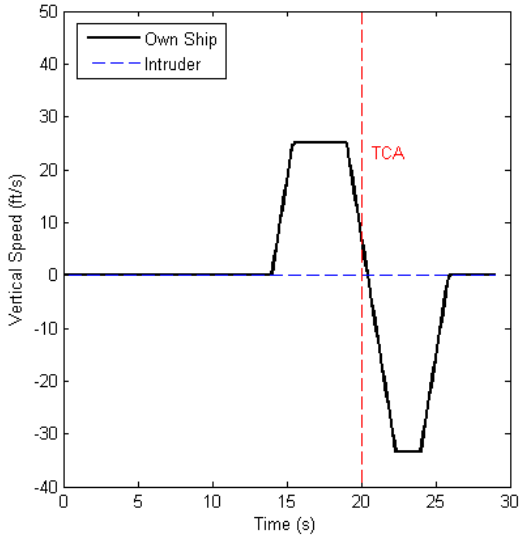
With $N_p = 10$, MC-RTBSS does not command an avoidance maneuver and VMD is $0\,\text{ft}$ and HMD is $900\,\text{ft}$. With $N_p = 3000$, at $t = 14\,\text{s}$ MC-RTBSS commands a climb at $25\,\text{ft/s}$; at $t = 19\,\text{s}$ it commands a descent at $33.3\,\text{ft/s}$; and at $t = 24\,\text{s}$ it commands a level off. The resulting VMD is $123\,\text{ft}$ and HMD is $900\,\text{ft}$. 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 $t = 14\,\text{s}$. The larger



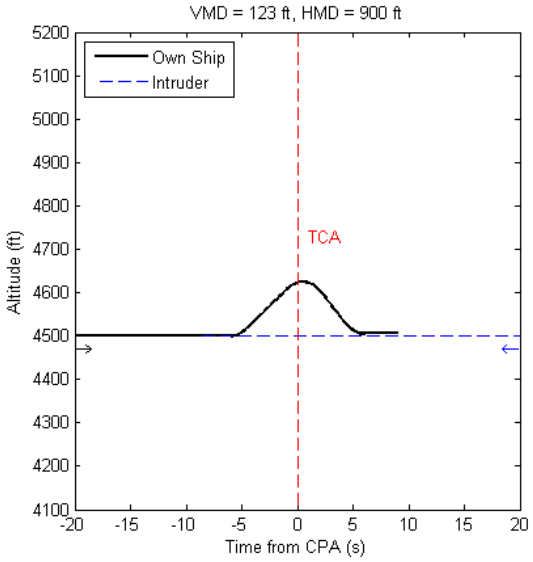
(a) Vertical Speed, $N_p = 10$.



(b) Altitude, $N_p = 10$.



(c) Vertical Speed, $N_p = 3000$.



(d) Altitude, $N_p = 3000$.

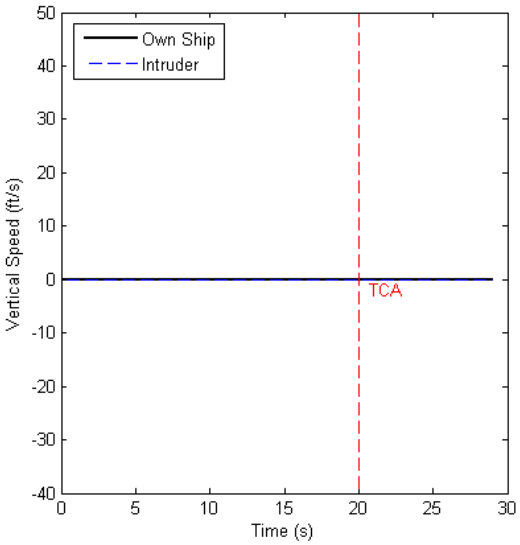
Figure 4-4: Varying N_p, single encounter.

number of samples used (3000 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 $t = 20\,\text{s}$, 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 ($33.3\,\text{ft/s}$) at $t = 19\,\text{s}$, before leveling off just prior to regaining track at $t = 24\,\text{s}$. Because of the low resolution of the action options available to MC-RTBSS in this implementation (1 action every 5 seconds), the algorithm realizes that choosing to descend further will result in even greater deviation below track and chooses to level off instead.

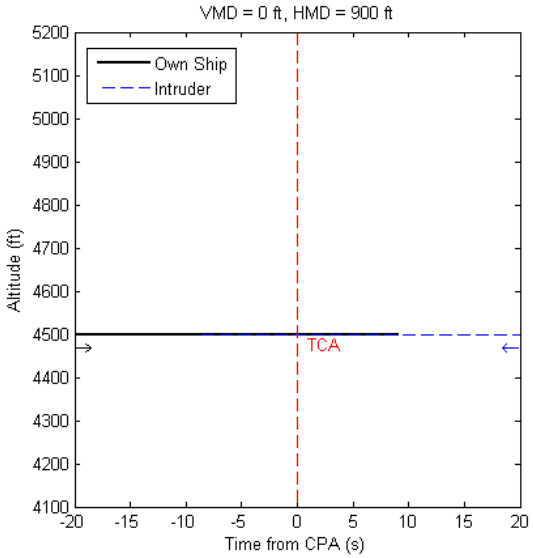
4.2.3 Varying Number of Observations, N_o

The results of an MC-RTBSS equipped aircraft with $N_o = 1$ and $N_o = 10$ are shown in Figure 4-5.

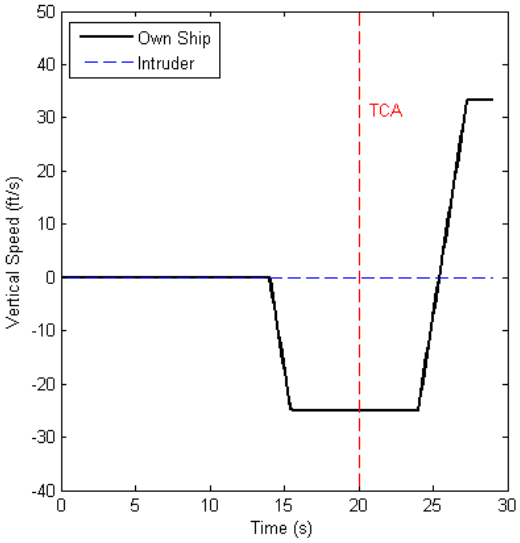
With $N_o = 1$, MC-RTBSS does not command an avoidance maneuver with these parameter settings and VMD is 0 ft and HMD is 900 ft. For $N_o = 10\,\text{s}$, at $t = 14\,\text{s}$ MC-RTBSS commands a descent at $25\,\text{ft/s}$ to avoid a potential NMAC, and at $t = 24\,\text{s}$ it commands a climb at $33.3\,\text{ft/s}$ to regain track. The resulting VMD is $-131\,\text{ft}$ and 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 which more completely span the belief space, allowing MC-RTBSS to account



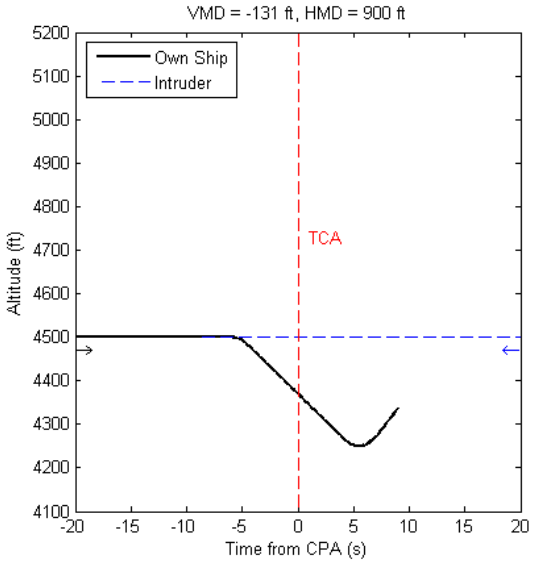
(a) Vertical Speed, $N_o = 1$



(b) Altitude, $N_o = 1$



(c) Vertical Speed, $N_o = 10$



(d) Altitude, $N_o = 10$

Figure 4-5: Varying N_o, single encounter.

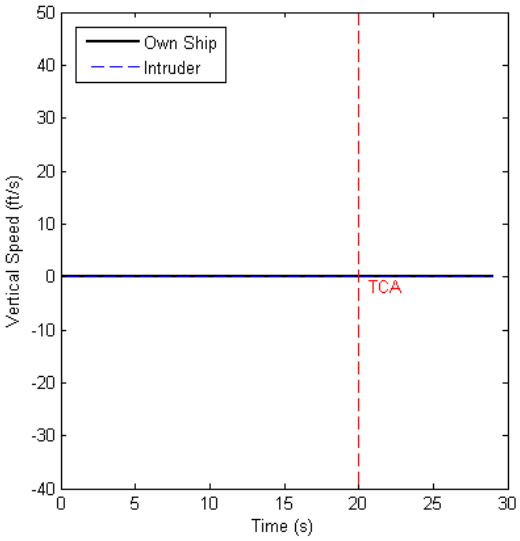
for rare events. With fewer observations, the search tree is narrow and MC-RTBSS may only account for a small subset of the belief space. Due to the stochastic nature of the observation generating process, in the case of few observations, this subset may only include rare events, though it tends to include only the most likely ones.

There is a discrepancy between the avoidance maneuvers commanded in the $N_p = 3000$ 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 idealized situation. If either option will yield the same rewards, then MC-RTBSS will choose whichever comes first in the sorted list of actions, the order of which is determined by the expected utility of a single decision point projection of the current belief state. If the utility of taking either action is arbitrarily close, then the order of the two in the list will be arbitrary as well.

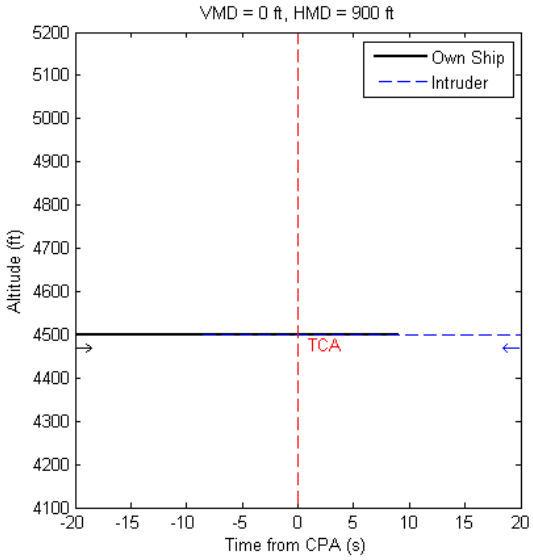
4.2.4 Varying NMAC Penalty, λ

The results of an MC-RTBSS-equipped aircraft with $\lambda = 1000$ and $\lambda = 10^{15}$ are shown in Figure 4-6.

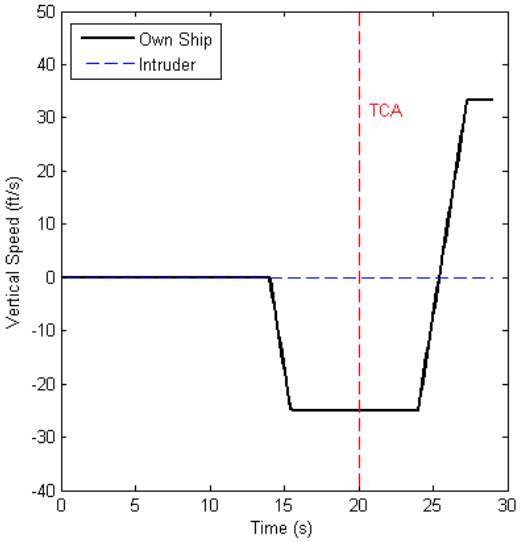
With $\lambda = 1000$, MC-RTBSS does not command an avoidance maneuver and the resulting VMD is 0 ft and HMD is 900 ft. With $\lambda = 10^{15}$, at $t = 14$ s MC-RTBSS commands a descent of 25 ft/s, and at $t = 24$ s a climb at 33.3 ft/s to begin to regain track. The reason for the different command with a larger λ is that in this scenario, the likelihood that the intruder aircraft will maneuver in such a way as to induce 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, then the expected reward associated with samples containing these events



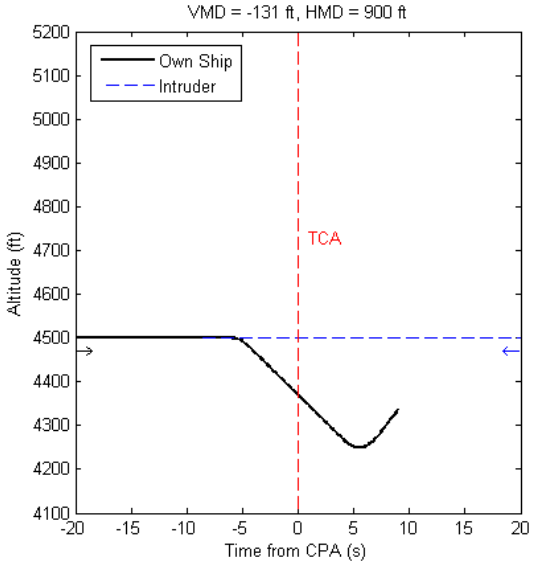
(a) Vertical Speed, $\lambda = 1000$



(b) Altitude, $\lambda = 1000$



(c) Vertical Speed, $\lambda = 10^{15}$



(d) Altitude, $\lambda = 10^{15}$

Figure 4-6: Varying λ, single encounter.

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.

4.2.5 Varying Maximum Search Depth, D

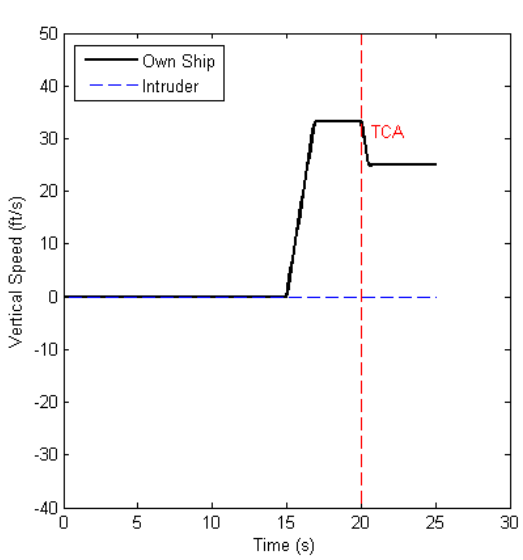
For the maximum search depth experiments, the head-on scenario is used to investigate how varying D affects the algorithm behavior temporally in the presence of an imminent NMAC; the imminence of a collision minimizes the effects of the stochasticity of the algorithm. The results of an MC-RTBSS equipped aircraft with $D = 3$ and $D = 4$ are shown in Figure 4-7.

When $D = 3$, at $t = 15\,\text{s}$ MC-RTBSS commands a $33.3\,\text{ft/s}$ climb, and at $t = 20\,\text{s}$ it commands a reduction in climb rate to $25\,\text{ft/s}$. The VMD is $143\,\text{ft}$ and HMD is $7\,\text{ft}$. When $D = 4$, at $t = 15\,\text{s}$ the algorithm commands a $25\,\text{ft/s}$ descent and the VMD is $-114\,\text{ft}$ and HMD is $5\,\text{ft}$. 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.

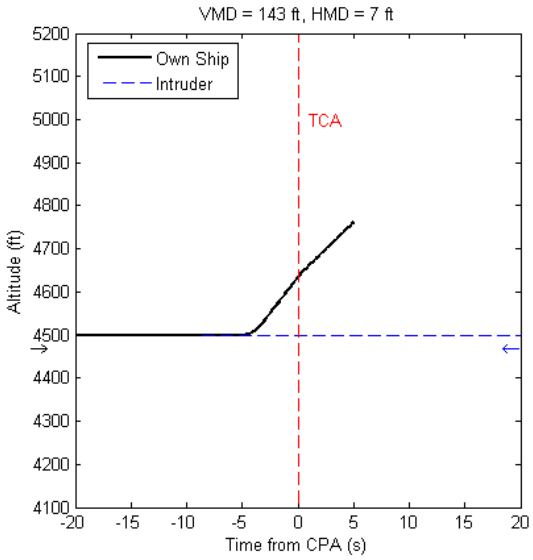
4.2.6 Varying Number of Particles in the Action Sort Function, N_{sort}

The head-on encounter is used for the N_{sort} experiments as well. The results of an MC-RTBSS equipped aircraft with $N_{sort} = 10$ and $N_{sort} = 1000$ are shown in Figure 4-8.

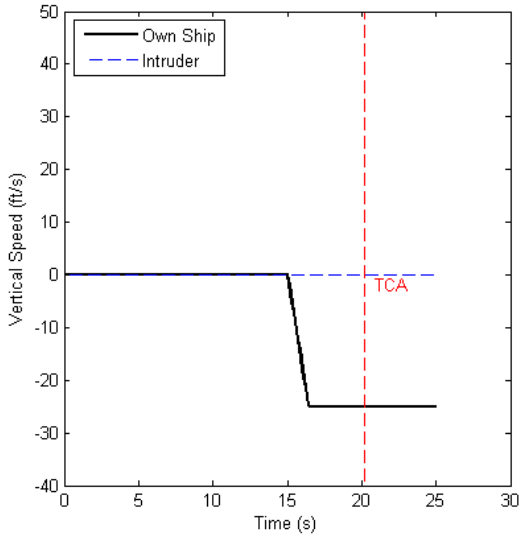
With $N_{sort} = 10$, at $t = 14\,\text{s}$ MC-RTBSS commands a climb at $25\,\text{ft/s}$ and at $t = 24\,\text{s}$, it commands a descent at $-33.3\,\text{ft/s}$ to regain track. The resulting VMD



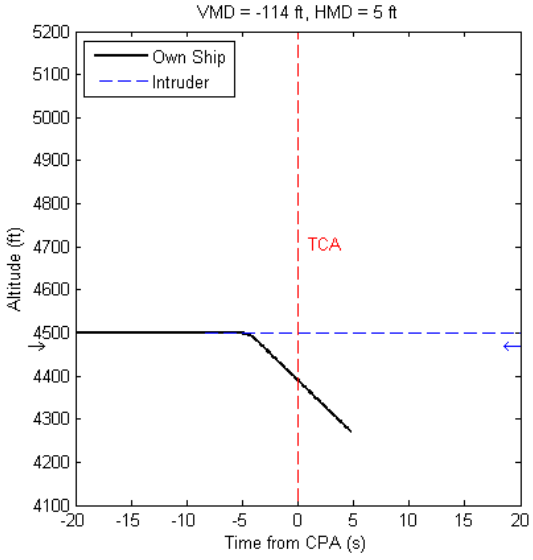
(a) Vertical Speed, $D = 3$



(b) Altitude, $D = 3$

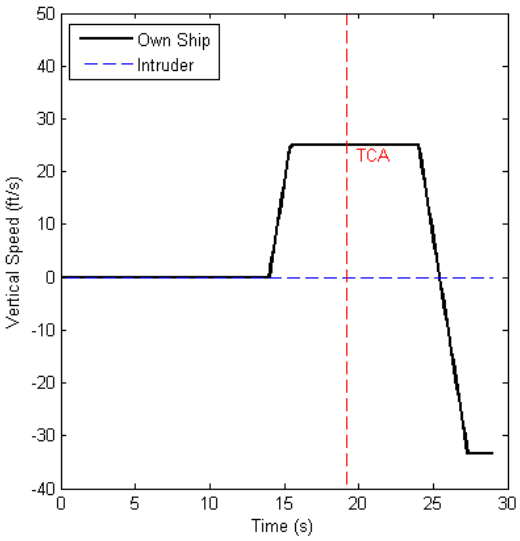


(c) Vertical Speed, $D = 4$

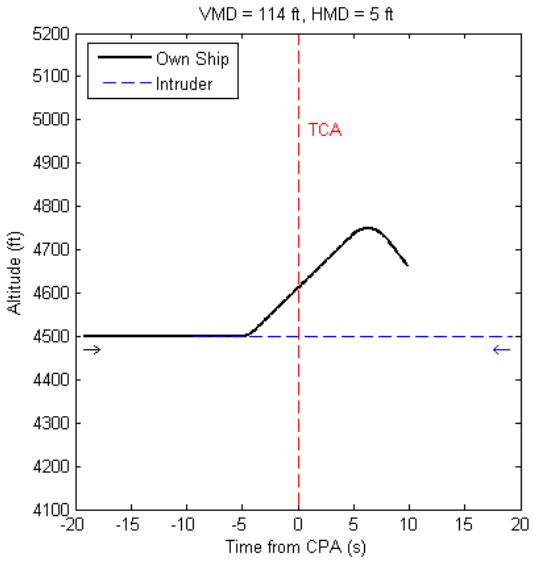


(d) Altitude, $D = 4$

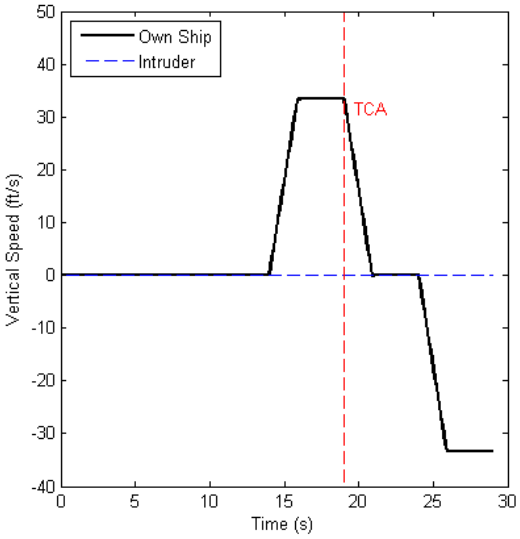
Figure 4-7: Varying D, single encounter.



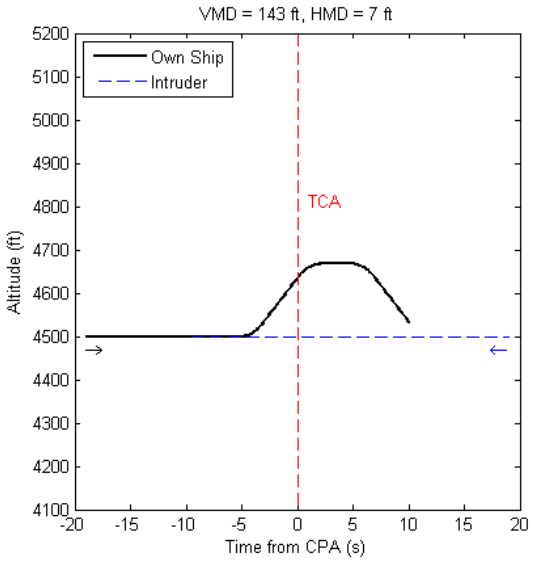
(a) Vertical Speed, $N_{sort} = 10$



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



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



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

Figure 4-8: Varying N_{sort}, single encounter.

is 114 ft and HMD is 5 ft. With $N_{sort} = 1000$, 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. The number of search tree nodes that MC-RTBSS expanded at each decision point in the simulation with each setting is shown in Table 4.3, where the Δ row shows the number of additional nodes pruned with $N_{sort} = 1000$.

Table 4.3: Nodes expanded

N_{sort}	$t = 4$ s	$t = 9$ s	$t = 14$ s	$t = 19$ s	$t = 24$ s
10	216	3198	1332	108	54
1000	237	2961	1098	162	27
Δ	-21	237	234	-54	27

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 $N_{sort} = 10$. In the $N_{sort} = 10$ 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 $N_{sort} = 1000$ case). This could be the result of pruning the optimal action (descending) at $t = 19$ s in the $N_{sort} = 10$ 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.

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 Figure 4.4.

Table 4.4: 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	6175	139	98
Large N_p	**3000**	3	10^{15}	3	10	6175	325	95
Small N_o	100	**1**	10^{15}	3	10	259	5	98
Large N_o	100	**10**	10^{15}	3	10	219661	21568	90
Small λ	100	3	$\mathbf{10^3}$	3	10	6175	134	98
Large λ	100	3	$\mathbf{10^{15}}$	3	10	6175	342	94
Small D	100	3	10^{15}	**3**	10	6175	702	89
Large D	100	3	10^{15}	**4**	10	346201	10404	97
Small N_{sort}	100	3	10^{15}	3	**10**	6175	823	87
Large N_{sort}	100	3	10^{15}	3	**1000**	6175	752	88

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 4.3), increasing the N_{sort} does result in a decrease in the number of nodes expanded, from 832 with $N_{sort} = 10$, to 752 with $N_{sort} = 1000$. The last column of Table 4.4 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.

4.2.7 Discussion

The previous results represent how MC-RTBSS tends to behave with various parameter settings. Each simulation was run several times to ensure that MC-RTBSS consistently commanded an avoidance maneuver or did not. However, the actual maneuvers the algorithm chooses in each simulation vary slightly, particularly in sense (climb vs. descent) in this co-altitude, head-on scenario. This varying behavior highlights the stochastic nature 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. Larger parameter values were not used in these simulations to ensure rapid computation times. 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).

4.3 Large Scale Simulation Results

The 76 encounters used for the performance evaluation were generated from the uncorrelated encounter model and are each 72 seconds in duration. The 76 encounters were first selected from a pool of 1 million randomly generated uncorrelated encounters based on VMD and HMD at TCA, the value of TCA, and simulation execution time. First, the 100 encounters with the earliest TCAs and VMD and HMD each less than 700 ft were selected from the million encounters to ensure that the aircraft would be close enough at TCA to potentially cause collision avoidance systems to alert. The latest TCA of the encounters was 72 seconds, which is used as the simulation length to ensure TCA is reached in every encounter during simulation. Next, several simulations were run with this set of 100 encounters. A large difference in computation time required to execute the simulations was noticed between many of the encounters; some encounters required significantly more time than the rest. Consequently, 24 encounters that required more than an hour of computation time with the nominal case (the parameter settings of the $\lambda = 10^{15}$ run in Section 4.2.4) were removed. Methods for reducing computation time are discussed in Section 5.3. The resulting 76 encounters are used for the parameter sweep experiments. The algorithm computes different bounds on the value function for each different scenario. Because the pruning depends on these bounds, the differences in bounds can result in differences in the amount of pruning and the associated reductions in computation time.

The 76 encounters all have small horizontal and vertical miss distances at TCA in order to observe algorithm behavior when NMACs are likely. The encounters are simulated using no collision avoidance system, with TCAS, and with MC-RTBSS with various parameter settings. In all trials, the intruder aircraft is equipped with a Mode S transponder. The results are presented in terms of objective measures of performance (total number of NMACs, mean miss distance for all encounters, mean maximum deviation, and mean average deviation over the course of each individual encounter) and by comparing these results for each MC-RTBSS parameter setting to the TCAS results. Three examples, sampled from the 76 encounters, are shown in Figure 4-9.

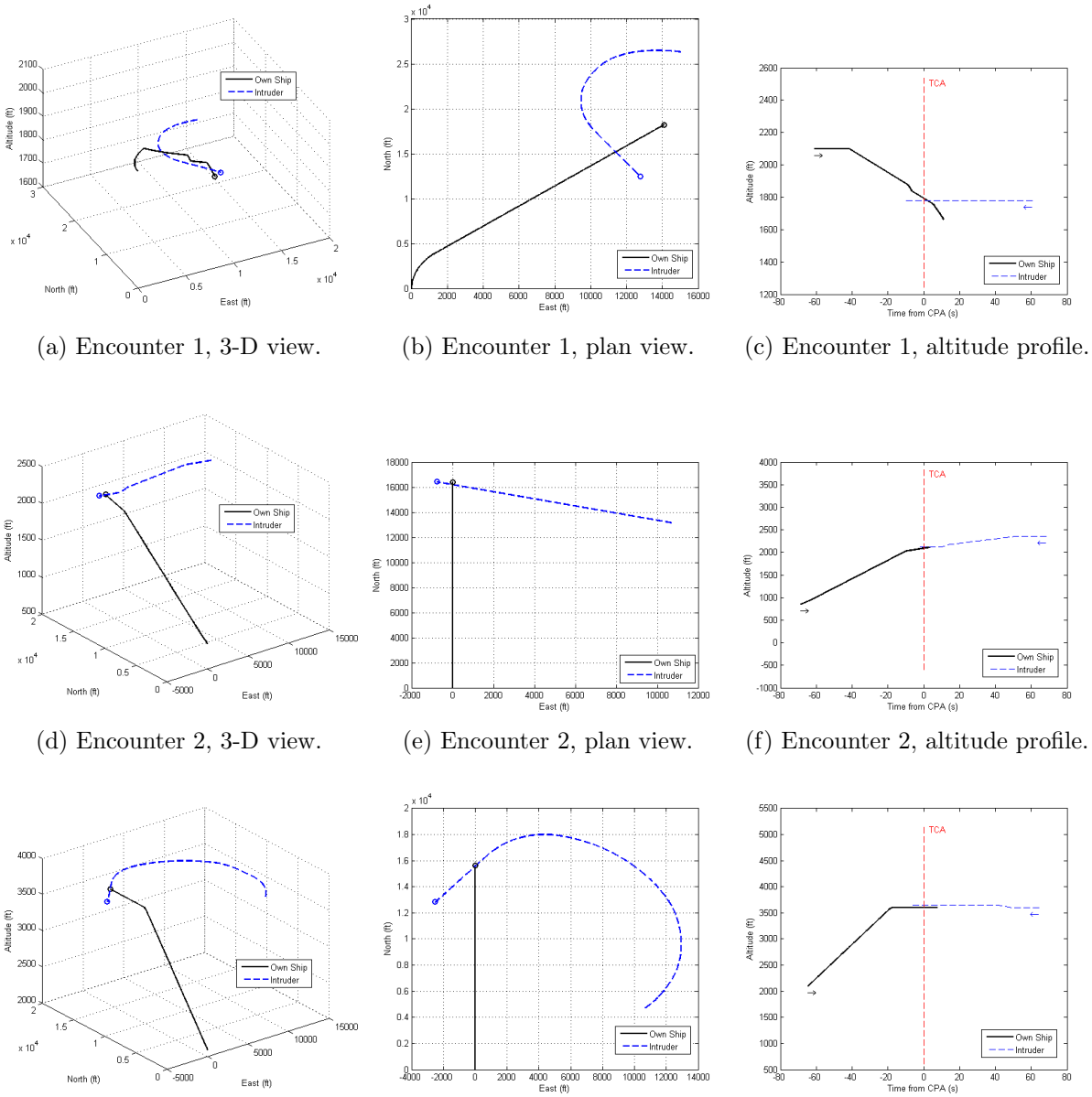
4.3.1 Cost of NMAC, λ

The λ values and other settings used in the parameter sweep are listed in Table 4.5.

Table 4.5: λ sweep values

λ	N_p	N_o	D	N_{sort}
10^0	100	3	3	10
10^3	100	3	3	10
10^7	100	3	3	10
10^{15}	100	3	3	10

The total number of NMACs for each value are plotted in Figure 4-10. The red line denotes the total number of NMACs when neither aircraft is CAS-equipped. The blue line denotes the number of NMACs that occur when the own ship is TCAS-equipped. The no CAS case results in 36 NMACs. Use of TCAS results in 1 NMAC. MC-RTBSS with $\lambda = 10^0$ results in 35 NMACs; with $\lambda = 10^3$ results in 34 NMACs; with $\lambda = 10^7$ results in 30 NMACs; and with $\lambda = 10^{15}$ results in 29 NMACs. As expected, as λ increases, the number of NMACs decreases. When λ is small enough, MC-RTBSS may not command an avoidance maneuver because even if an NMAC is imminent, the cost of deviating from the nominal trajectory may outweigh the cost of the NMAC. As λ increases, even unlikely potential NMACs have an effect on the



(a) Encounter 1, 3-D view. (b) Encounter 1, plan view. (c) Encounter 1, altitude profile.

(d) Encounter 2, 3-D view. (e) Encounter 2, plan view. (f) Encounter 2, altitude profile.

(g) Encounter 3, 3-D view. (h) Encounter 3, plan view. (i) Encounter 3, altitude profile.

Figure 4-9: Examples of generated encounters.

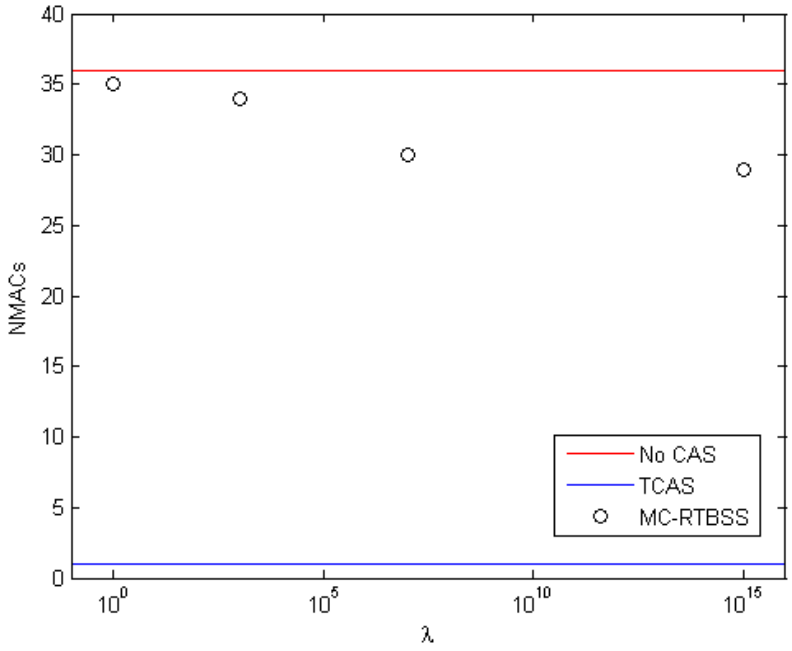


Figure 4-10: Total number of NMACs, varying λ.

expected returns. For large values of λ, the parameter is affecting the tolerance for risk. Larger values of λ will result in avoidance maneuvers for less likely events. When λ exceeds 10^{15}, the number of NMACs remains constant at 29 for even significant increases in λ. This constant number of NMACs with increasing λ implies that MC-RTBSS is unable to identify and account for these NMACs prior to their occurrence with these parameter settings. Increases in N_p and N_o would increase the likelihood that the algorithm would identify these NMACs and plan to avoid them accordingly. While MC-RTBSS is avoiding some NMACs (all data points are below the no CAS results), TCAS performs significantly better for these parameter settings.

The mean miss distances are shown in Figure 4-11 for aircraft with no CAS, with TCAS, and with MC-RTBSS for varying λ values. The miss distance is the minimum straight line distance between the two aircraft at any point during the encounter. For no CAS, the mean miss distance is 404 ft and for TCAS it is 581 ft. For MC-RTBSS with $\lambda = 10^0$, the mean miss distance is 435 ft; with $\lambda = 10^3$ the mean miss distance is 436 ft; with $\lambda = 10^7$ and $\lambda = 10^{15}$ it is 440 ft. In general, TCAS results in an increase of about 140 ft in miss distance compared to MC-RTBSS, which increases the miss

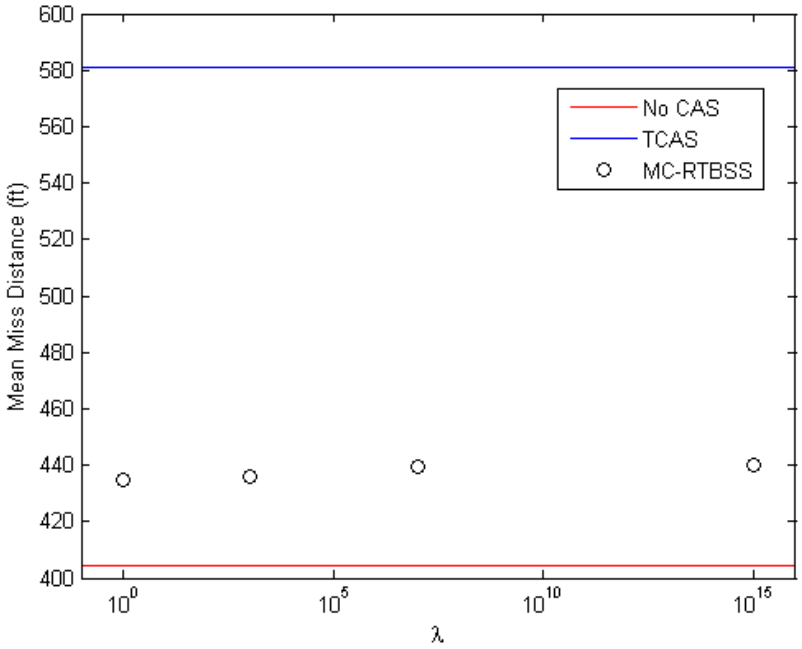


Figure 4-11: Mean miss distance, varying λ.

distance an average of about 40 ft compared to when no CAS is used. These results are consistent with the fact that MC-RTBSS is commanding avoidance maneuvers to avoid NMACs.

The average deviations are shown in Figure 4-12. The average deviation is the deviation of the own ship from the nominal trajectory averaged over the entire encounter. By definition, the no CAS case has no deviation. With TCAS, the average deviation is 108 ft. With MC-RTBSS and $\lambda = 10^0$, the average deviation is 47 ft; with $\lambda = 1000$ it is 48 ft; with $\lambda = 10^7$ it is 49 ft; and with $\lambda = 10^{15}$ it is 50 ft. While TCAS results in fewer NMACs than MC-RTBSS under these settings, it does so at the cost of more deviation overall. Increasing λ also results in MC-RTBSS deviating more, though still significantly less than TCAS.

Figures 4-13 and 4-14 show a comparison of the results from each individual encounter between MC-RTBSS for the various λ values and TCAS for two metrics: miss distance and average deviation. The black line along the diagonal of the plot represents points where both MC-RTBSS and TCAS have the same metric values. Points below the line are encounters in which TCAS results in a greater metric value

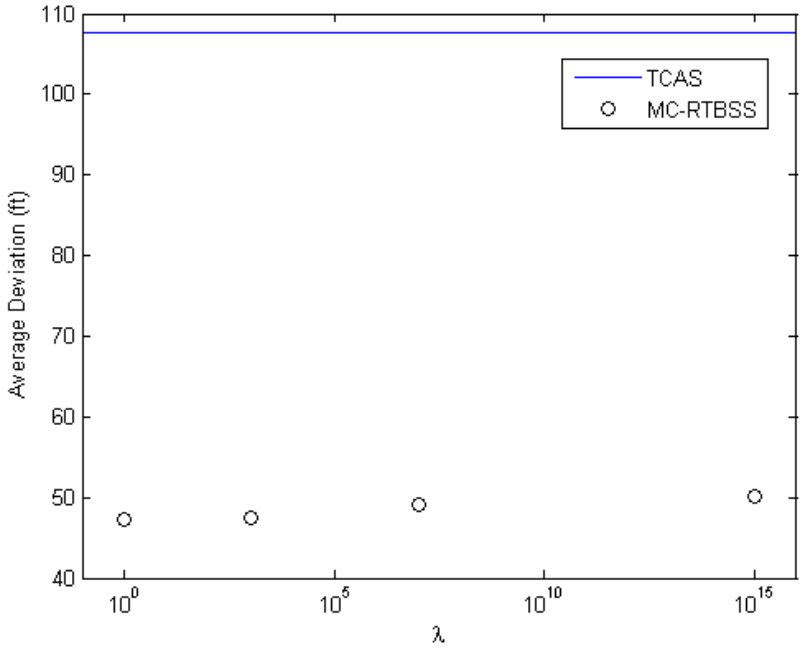


Figure 4-12: Average deviation, varying λ.

and points above the line are encounters in which MC-RTBSS results in a greater metric value. These plots show that in general, TCAS results in a greater deviation and a larger miss distance than MC-RTBSS. MC-RTBSS, on the other hand, results in some of the most significant outliers, which tend to increase the mean values seen in Figures 4-11 and 4-12. The TCAS results tend to be more tightly clustered, displaying less variation in behavior.

4.3.2 Number of Particles, N_p

The N_p values and other settings used in the parameter sweep are listed in Table 4.6.

Table 4.6: N_p sweep values

N_p	N_o	λ	D	N_{sort}
10	3	10^{15}	3	10
100	3	10^{15}	3	10
1000	3	10^{15}	3	10

The total number of NMACs for each value are plotted in Figure 4-15. The red

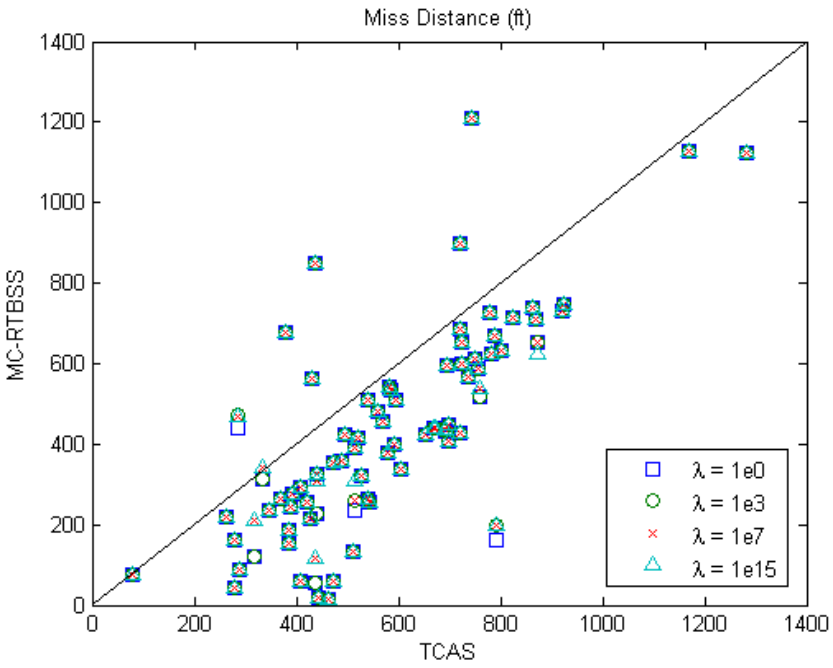


Figure 4-13: Miss distance comparison, varying λ.

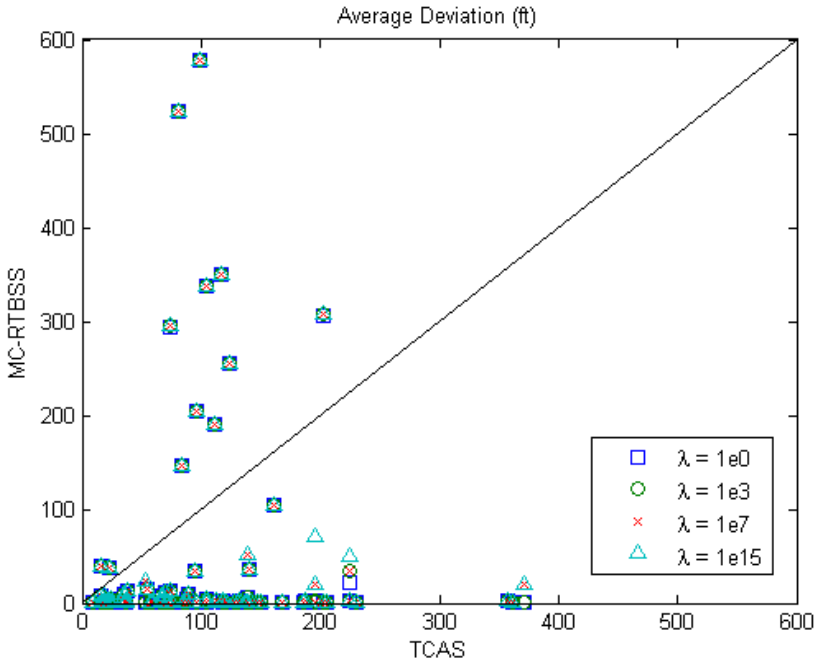


Figure 4-14: Average deviation comparison, varying λ.

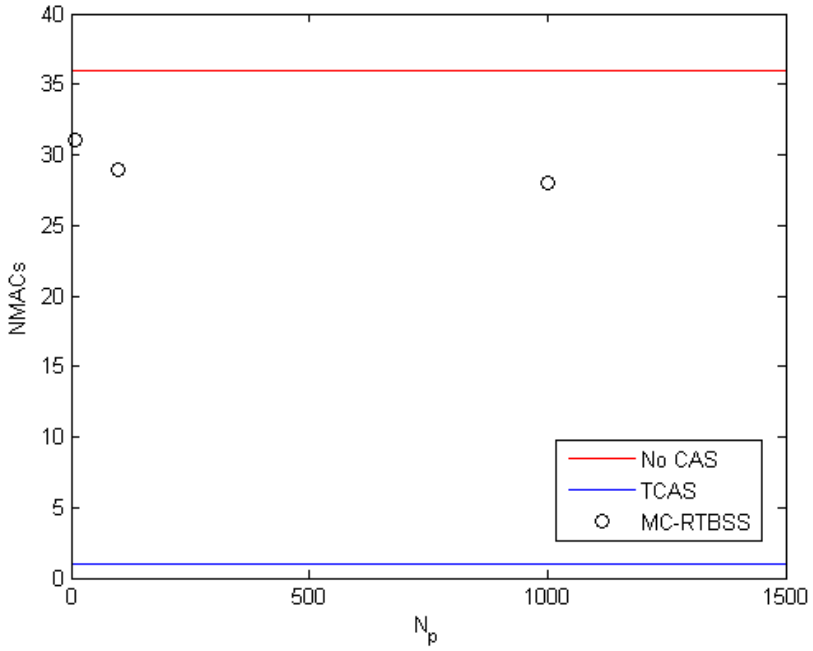


Figure 4-15: Total number of NMACs, varying N_p.

line denotes the total number of NMACs when neither aircraft is CAS-equipped. The blue line denotes the number of NMACs that occur when the own ship is TCAS-equipped. The no CAS case results in 36 NMACs. Use of TCAS results in 1 NMAC. MC-RTBSS with $N_p = 10$ results in 31 NMACs; with $N_p = 100$ results in 29 NMACs; and with $N_p = 1000$ it results in 28 NMACs. As expected, as N_p increases, the number of NMACs decreases. As N_p increases, the algorithm is more likely to sample from rare state subspaces, allowing it to account for rare trajectories, which may contain NMACs. With a small number of particles, MC-RTBSS has a smaller chance of generating these trajectories and is unlikely to avoid such NMACs.

The mean miss distances are shown in Figure 4-15 for aircraft with no CAS, with TCAS, and with MC-RTBSS for varying N_p values. For no CAS, the mean miss distance is 404 ft and for TCAS it is 581 ft. For MC-RTBSS with $N_p = 10$, the mean miss distance is 439 ft; with $N_p = 100$ the mean miss distance is 440 ft; and with $N_p = 1000$ it is 449 ft. In general, TCAS results in an increase of about 140 ft in miss distance compared to MC-RTBSS, which increases the miss distance an average of about 40 ft compared to when no CAS is used. These results are consistent with the

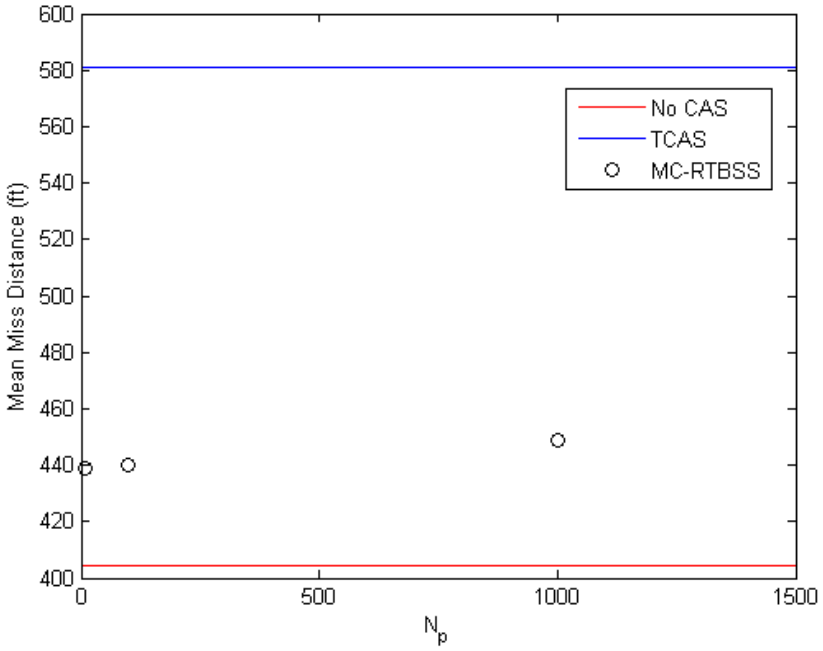


Figure 4-16: Mean miss distance, varying N_p.

fact that MC-RTBSS is commanding avoidance maneuvers to avoid NMACs. As N_p increases, the miss distance is expected to increase because the algorithm is able to act more cautiously, identifying and avoiding unlikely potential NMACs, which are usually the result of unlikely, aggressive intruder maneuvers. MC-RTBSS efforts to avoid the potential NMACs will tend to result in a larger miss distance as compared to an implementation that does not account for the possibility of such aggressive maneuvers (e.g. one that uses a smaller N_p).

The average deviations are shown in Figure 4-17. With TCAS, the average deviation is 108 ft. With MC-RTBSS and $N_p = 10$, the average deviation is 49 ft; with $N_p = 100$ it is 50 ft; and with $N_p = 1000$ it is 52 ft. Increasing N_p results in MC-RTBSS deviating more, though still significantly less than TCAS. This increased deviation is expected, as with increasing N_p, MC-RTBSS will maneuver more frequently and more aggressively in order to account for rarer potential NMACs.

Figures 4-18 and 4-19 show a comparison of the results from each individual encounter between MC-RTBSS for the various N_p values and TCAS. These plots show that in general, TCAS results in a greater deviation and a larger miss distance

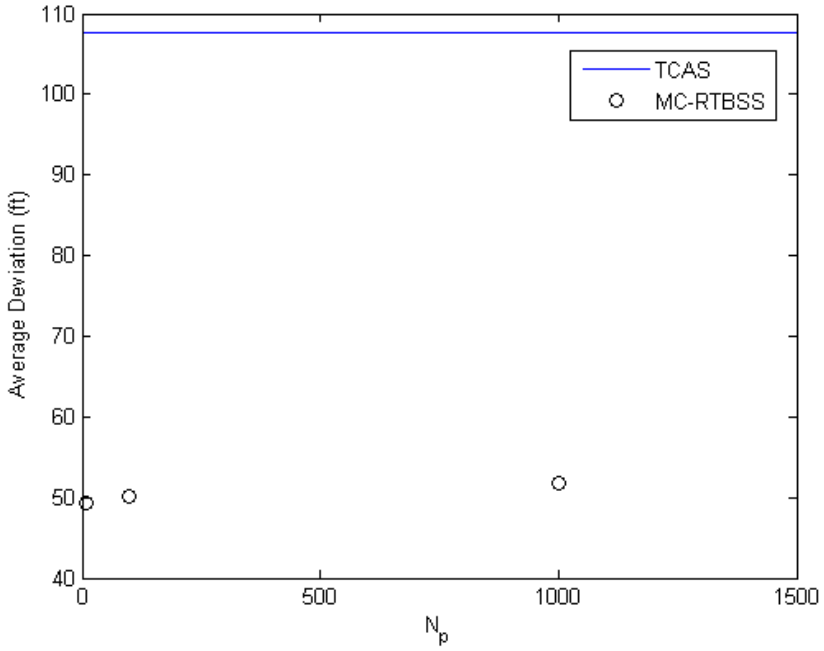


Figure 4-17: Average deviation, varying N_p.

than MC-RTBSS with varying N_p. MC-RTBSS, on the other hand, results in some of the most significant outliers, which tend to increase the mean values seen in Figures 4-16 and 4-17. The TCAS results tend to be more tightly clustered, displaying less variation in behavior.

4.3.3 Number of Observations, N_o

The N_o values and other settings used in the parameter sweep are listed in Table 4.7.

Table 4.7: N_o sweep values

N_o	N_p	λ	D	N_{sort}
3	100	10^{15}	3	10
6	100	10^{15}	3	10
9	100	10^{15}	3	10

The total number of NMACs for each value are plotted in Figure 4-20. The red line denotes the total number of NMACs when neither aircraft is CAS-equipped. The blue line denotes the number of NMACs that occur when the own ship is TCAS-equipped.

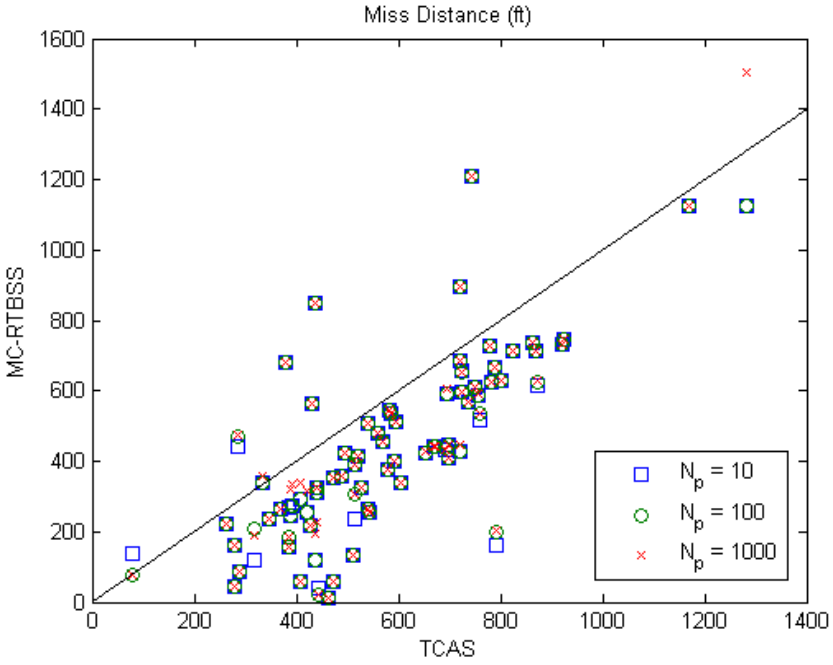


Figure 4-18: Miss distance comparison, varying N_p.

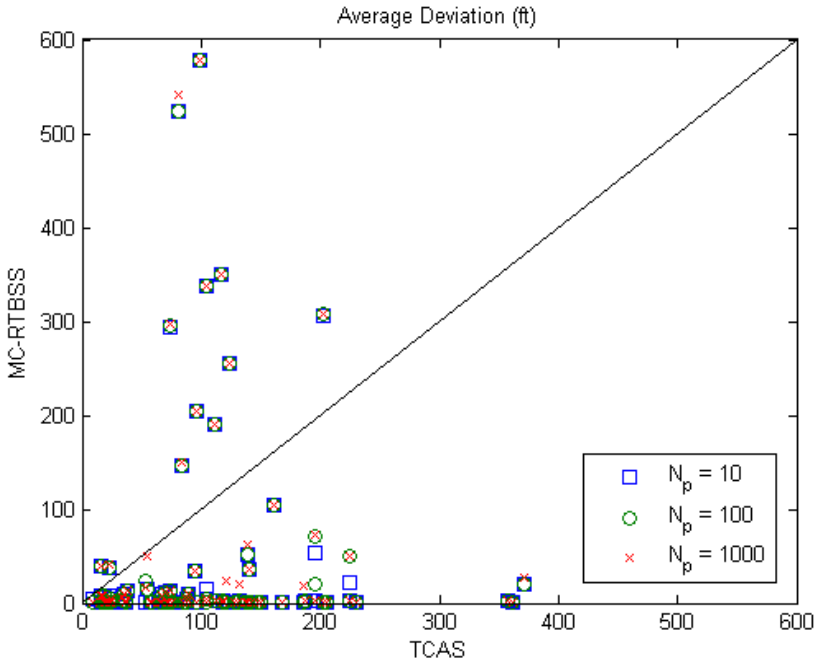


Figure 4-19: Average deviation comparison, varying N_p.

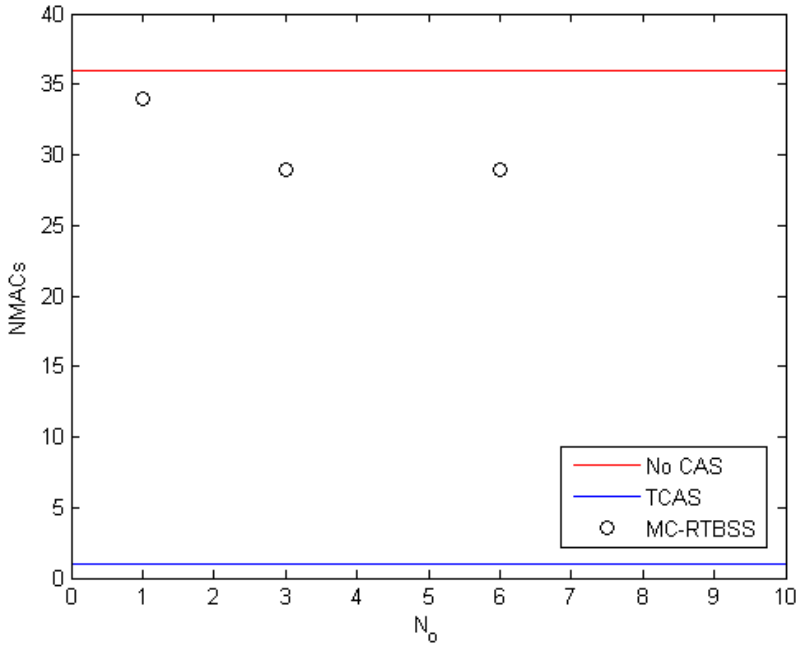


Figure 4-20: Total number of NMACs, varying N_o.

The no CAS case results in 36 NMACs. Use of TCAS results in 1 NMAC. MC-RTBSS with $N_o = 1$ results in 34 NMACs; with $N_p = 3$ results in 29 NMACs; and with $N_p = 6$ it results in 29 NMACs. As expected, as N_o increases, the number of NMACs decreases. As N_o increases, the algorithm is more likely to generate rare observations and consequently sample from rare state subspaces, allowing it to account for rare trajectories, which may contain NMACs. With a small number of observations, MC-RTBSS has a smaller chance of generating these trajectories and is unlikely to avoid such NMACs.

The mean miss distances are shown in Figure 4-20 for aircraft with no CAS, with TCAS, and with MC-RTBSS for varying N_o values. For no CAS, the mean miss distance is 404 ft and for TCAS it is 581 ft. For MC-RTBSS with $N_o = 1$, the mean miss distance is 437 ft; with $N_o = 3$ the mean miss distance is 442 ft; and with $N_o = 6$ it is 442 ft. In general, TCAS results in an increase of about 140 ft in miss distance compared to MC-RTBSS, which increases the miss distance an average of about 40 ft compared to when no CAS is used. These results are consistent with the fact that MC-RTBSS is commanding avoidance maneuvers to avoid NMACs. As N_o increases,

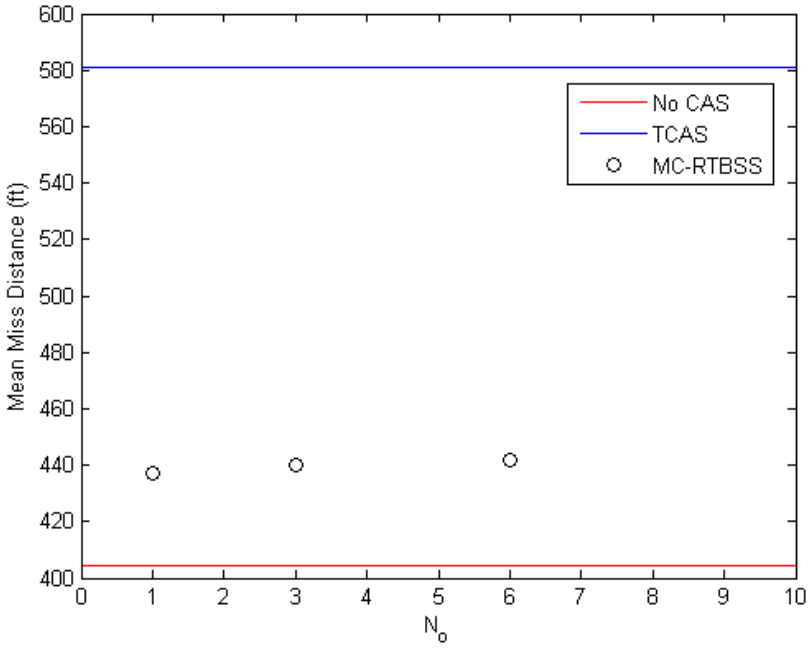


Figure 4-21: Mean miss distance, varying N_o.

the miss distance is expected to increase because the algorithm is able to act more cautiously, identifying and avoiding unlikely potential NMACs, which are usually the result of unlikely, aggressive intruder maneuvers. MC-RTBSS efforts to avoid the potential NMACs will tend to result in a larger miss distance as compared to an implementation that does not account for the possibility of such aggressive maneuvers (e.g. one that uses a smaller N_o).

The average deviations are shown in Figure 4-22. With TCAS, the average deviation is 108 ft. With MC-RTBSS and $N_o = 1$, the average deviation is 49 ft; with $N_o = 3$ it is 50 ft; and with $N_o = 6$ it is 51 ft. Increasing N_o results in MC-RTBSS deviating more, though still significantly less than TCAS. This increased deviation is expected, as with increasing N_o, MC-RTBSS will maneuver more frequently and more aggressively in order to account for rarer potential NMACs.

Figures 4-23 and 4-24 show a comparison of the results from each individual encounter between MC-RTBSS for the various N_o values and TCAS. These plots show that in general, TCAS results in a greater deviation and a larger miss distance than MC-RTBSS with varying N_o. MC-RTBSS, on the other hand, results in some of

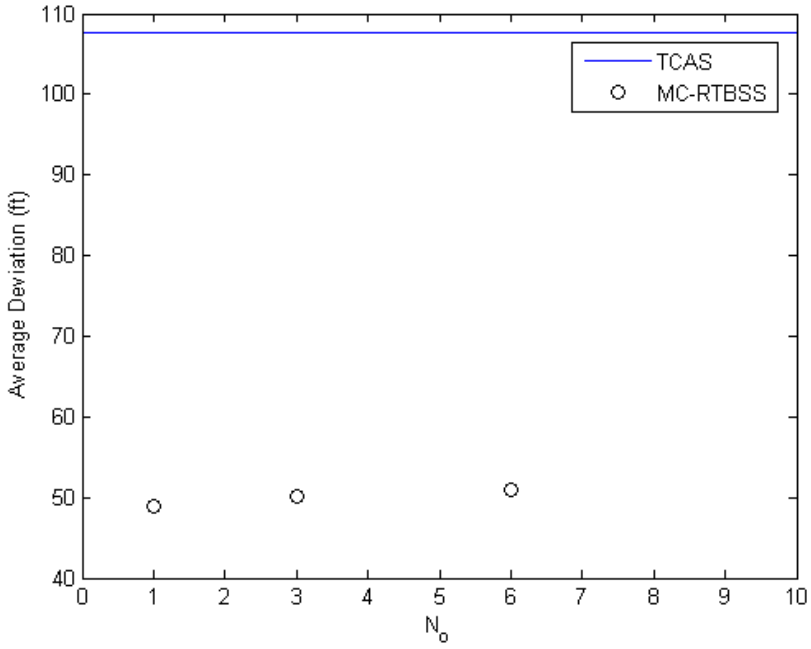


Figure 4-22: Average deviation, varying N_o.

the most significant outliers, which tend to increase the mean values seen in Figures 4-21 and 4-22. The TCAS results tend to be more tightly clustered, displaying less variation in quantitative behavior.

4.3.4 Discussion

The general trend of the parameter sweep results is that increasing parameter values results in better MC-RTBSS performance. TCAS avoids more NMACs than MC-RTBSS but deviates more. The breadth of the parameter values used in the parameter sweeps was limited by the large computation time associated with simulations involving large N_p and N_o values. In its current Simulink implementation, the computation time required to achieve performance matching TCAS would make use of MC-RTBSS in real-time infeasible.

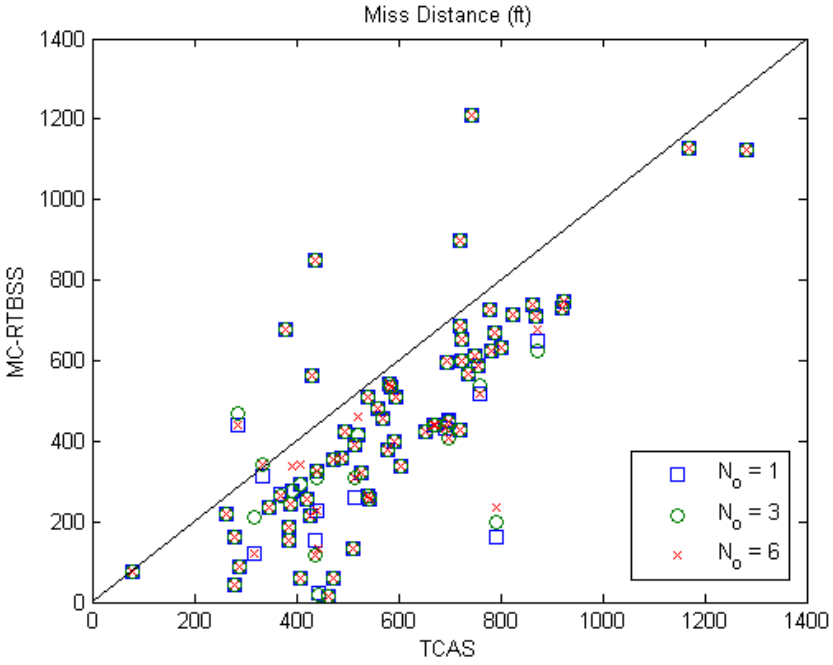


Figure 4-23: Miss distance comparison, varying N_o.

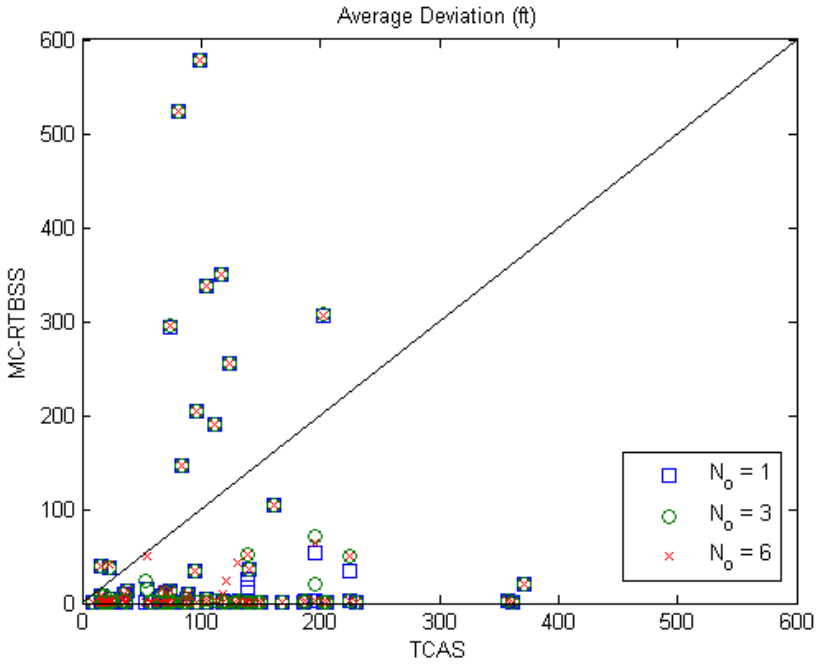


Figure 4-24: Average deviation comparison, varying N_o.

Chapter 5

Conclusion

This chapter summarizes the results of this thesis, outlines the contributions made, and suggests areas of further research. Many of the suggested further research areas are focused on reducing computation time to permit real-time use of MC-RTBSS and to improve performance with the use of increased parameter values.

5.1 Summary

Chapter 4 presents the results of two sets of simulations: single encounter simulations and batch runs of 76 randomly generated encounters. Section 4.2 presents the results of the single encounter simulations, which demonstrate how each algorithm parameter affects the behavior of MC-RTBSS. As was shown in Chapter 4, the number of particles, N_p, affects the ability to identify potential NMACs. Increasing N_p 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. The cost of NMAC, λ, affects the relative cost of an NMAC versus deviating from the nominal path. Increasing λ tends to make the algorithm more conservative, commanding maneuvers to avoid less likely potential safety events and willing to deviate more to avoid them. The maximum search depth, D, controls the planning horizon of the algorithm.

Increasing D allows the algorithm to look farther ahead and consider events farther into the future. MC-RTBSS will not necessarily behave differently with a different value for D; the algorithm should identify the proper action to take at the proper time and act accordingly. However, the algorithm can induce an NMAC that occurs at least a planning horizon after the inducing action is taken. In such a situation, the algorithm would not be able to consider the full effects of its action choice because the planning horizon is too short. The number of particles used in the sorting function, N_{sort}, affect the accuracy of the heuristic function used in the branch-and-bound method. Increasing N_{sort} increases the accuracy of the upper bound on a particular action choice, which may help to prune more subtrees and to avoid pruning the optimal subtree.

Section 4.3 presents the results of the batch runs, which investigate how varying parameter settings affect the overall performance of the algorithm. Increasing λ, N_p, and N_o all cause the algorithm to avoid more NMACs, but as expected, the algorithm deviates from the nominal path more. In general, TCAS avoids more NMACs than MC-RTBSS, but TCAS deviates more from the nominal path. The most significant limit on performance is likely the large computation time associated with large parameter settings; with a large enough N_p and N_o, MC-RTBSS should be able to identify and avoid rare NMACs.

5.2 Contributions

This work presents a novel online POMDP approximation algorithm. Chapter 3 explains the synthesis of the new algorithm, MC-RTBSS, by combining elements from Paquet's RTBSS and Thrun's belief state projection. The result is an online, sample-based POMDP approximation algorithm for continuous state and observation spaces. MC-RTBSS is able to integrate dynamic Bayesian network-based airspace encounter models into the transition model, effectively leveraging all available information for the decision-making process.

- **Integration of encounter model into collision avoidance system logic.**

The sample-based belief state representation used in MC-RTBSS lends itself well to the utilization of a dynamic Bayesian network, which must be sampled to produce posterior distributions which account for the potentially complex interdependencies between state variables. As Section 3.3.2 explains, the MC-RTBSS transition function, which propagates individual particles forward, samples from the encounter model to determine the intruder action at each time step. While most collision avoidance systems use a naive intruder action model, with enough particles, MC-RTBSS predicts future intruder behavior that is statistically consistent with behavior observed over the U.S. airspace, permitting a more accurate approximation of the optimal policy.

- **Synthesis of sample-based online POMDP solver.** Thrun uses belief state projection in an offline reinforcement learning algorithm. Paquet's RTBSS is an online algorithm that requires a finite number of discrete states and observations. MC-RTBSS successfully integrates Thrun's belief state projection into Paquet's RTBSS algorithm, resulting in a sample-based online POMDP solver, with few restrictions on the state or observations spaces.

- **Investigation of computation time reduction.** Section 4.2.6 offers insight into the effect of N_{sort} on pruning and computation time. This work investigates the use of a sample-based heuristic method to branch-and-bound pruning, demonstrating a trade off between additional computation required for sorting and reduction in total computation time.

5.3 Further Work

Computation time was the most limiting factor in this work. Reduction in computation time would open the door to more thorough analysis and investigation of MC-RTBSS performance, in addition to a real-time implementation.

- **Alternate coding methods.** Chapter 4 demonstrates how various parameters affect algorithm behavior in the collision avoidance problem. In particular,

increasing N_p and N_o enhances algorithm performance. However, this enhanced performance is associated with a significantly longer computation time. Further work would investigate implementing MC-RTBSS in different programming languages 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.

- **Parallelization.** Portions of MC-RTBSS may be easily parallelized: particle projection in particular. A significantly larger number of particles could be used if multiple processors are used during execution of the ParticleProject subroutine.

- **Reward function.** Further work is needed to investigate methods of compensating for when the belief state space cannot be spanned sufficiently in adequate time. The reward function relies on the identification of rare events (NMACs) using Monte-Carlo simulation (through belief state projection) to compute bounds on the value function. The low probability of an NMAC occurring necessitates large N_p and N_o values to adequately account for the possibility, which results in large computation times. One possible solution to reduce the required N_p and N_o is to increase the size of the definition of an NMAC within the reward function. A larger NMAC cylinder could compensate for the sparsity of particles by providing a safety buffer. A larger cylinder would essentially increase the probability of an NMAC in the reward function, causing MC-RTBSS to be more cautious. The NMAC penalty, λ might need to be changed as well, to offset the artificially increased probability of NMAC. Further work would investigate the effect of varying NMAC sizes and λ values on collision frequency and deviation through Monte-Carlo simulation.

- **Application of Q_{MDP} method.** By assuming full observability of the intruder state, the aircraft collision avoidance problem may be treated as an MDP. The POMDP search tree would consist only of future states instead of belief states

weighted by generated observations. Estimates of the optimal action from the current belief state may then be computed efficiently using a variety of methods (Littman et al., 1995). Because the observation model is not used for planning, this method ignores the value of future information. This phenomenon could significantly affect performance when a sensor configuration permits information gathering behavior, such as passive ranging for EO/IR sensors (Shakernia et al., 2005). Future work could investigate how using Q_{MDP} approaches affects computation time and performance.

- **Importance sampling.** Another strategy for reducing computation time while enhancing performance is to reduce the number of particles and observations required to 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 (Srinivasan, 2002). 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.

- **Action choices.** Changing the actions available to MC-RTBSS would dramatically affect performance. In particular, future work should examine the benefit of more or varied action choices, such as unscripted turns and accelerations, on MC-RTBSS performance in simulation.

- **Monte-Carlo evaluation of algorithm performance.** Significant reduction in computation time would also make MC-RTBSS more amenable to Monte-Carlo simulation, which would likely be required of any certification process for implementation aboard an unmanned aircraft. Much work is required to tune the algorithm parameters for nominal performance.

- **Transition function dynamics.** Section 3.3.2 states that the transition func-

tion does not model vertical acceleration ($\ddot{h}$), bank angle (ϕ), angular rates (p,q,r) or accelerations ($\dot{p}, \dot{q}, \dot{r}$) and consequently is unable to model aircraft performance limitations on these variables. Further work is needed to model these variables in the transition model as well as to incorporate aircraft performance limits. Closing the gap between the transition model and the real world would enhance performance and would ensure predictable algorithm behavior in all situations the algorithm might encounter.

- **Performance metrics.** The use of an explicit reward function permits the fine tuning of performance objectives. However, the stochastic nature of MC-RTBSS presents challenges to guarantees on this performance. Future research is needed to investigate the convergence of the value function with varying parameter settings. In addition, future work may involve developing a metric describing how often the algorithm prunes optimal subtrees.

Bibliography

Aveneau, C. and Bonnemaison, B. (2001). ACAS programme ACASA project work package 3 — final report on ACAS/RVSM interaction. Study report, Eurocontrol.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI.

Edwards, M., Kochenderfer, M., Kuchar, J., and Espindle, L. (2009). Encounter models for unconventional aircraft version 1.0. Project Report ATC-348, Lincoln Laboratory.

Frazzoli, E., Dahleh, M. A., and Feron, E. (2004). Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Aerospace Computing, Information, and Communication*, 25:116–129.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134.

Kim, H. J. and Shim, D. H. (2003). A flight control system for aerial robots: Algorithms and experiments. *Control Engineering Practice*, 11(12):1389–1400.

Kochenderfer, M., Espindle, L., Kuchar, J., and Griffith, J. (2008a). Correlated encounter model for cooperative aircraft in the national airspace system version 1.0. Project Report ATC-344, Lincoln Laboratory.

Kochenderfer, M., Kuchar, J., Espindle, L., and Griffith, J. (2008b). Uncorrelated encounter model of the national airspace system version 1.0. Project Report ATC-345, Lincoln Laboratory.

Kochenderfer, M. J. (2009). Personal communication.

Kochenderfer, M. J., Espindle, L. P., Kuchar, J. K., and Griffith, J. D. (2008c). A comprehensive aircraft encounter model of the national airspace system. *Lincoln Laboratory Journal*, 17(2):41–53.

Kuchar, J. K. and Drumm, A. C. (2007). The traffic alert and collision avoidance system. *Lincoln Laboratory Journal*, 16(2):277–296.

Kuchar, J. K. and Yang, L. C. (2000). A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems*, 1(4):179–189.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings in Robotics: Science and Systems*.

Littman, M., Cassandra, A., and Kaebling, L. (1995). Learning policies for partially observable environments: Scaling up. In Prieditis, A. and Russell, S., editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370.

Neapolitan, R. (2004). *Learning Bayesian Networks*. Pearson Prentice Hall.

Paquet, S., Tobin, L., and Chaib-draa, B. (2005a). Online POMDP algorithm for complex multiagent environments. In *AAMAS*, pages 970–977.

Paquet, S., Tobin, L., and Chaib-draa, B. (2005b). Real-time decision making for large POMDPs. *Artificial Intelligence*, (LNAI 3501):450–455.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025–1032.

Ross, S., Pineau, J., Paquet, S., and Chaib-draa, B. (2008). Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704.

RTCA (1997). Minimum operational performance standards for traffic alert and collision avoidance system II (TCAS II) airborne equipment. Technical report, RTCA/DO-185A.

Schouwenaars, T., Mettler, B., Feron, E., and How, J. (2004). Hybrid model for trajectory planning of agile autonomous aerial vehicles. *Journal of Aerospace Computing, Information, and Communication, Special Issue on Intelligent Systems*, 1.

Shakernia, O., Chen, W.-Z., and Raska, V. M. (2005). Passive ranging for UAV sense and avoid applications. In *Proceedings of Infotech@Aerospace*, number AIAA 2005-7179. AIAA.

Smith, T. and Simmons, R. G. (2004). Heuristic search value iteration for POMDPs. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Srinivasan, R. (2002). *Importance Sampling: Applications in Communications and Detection*. Springer-Verlag.

Sundqvist, B.-G. (2005). Auto-ACAS — Robust nuisance-free collision avoidance. In *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference*, pages 3961–3963. IEEE.

The MITRE Corporation (1983). System safety study of minimum TCAS II. Technical Report MTR-83W241.

Thrun, S. (2000). Monte Carlo POMDPs. In Solla, S., Leen, T., and Müller, K.-R., editors, *Advances in Neural Information Processing Systems 12*, pages 1064–1070. MIT Press.

Vengerov, D. (2008). A gradient-based reinforcement learning approach to dynamic pricing in partially-observable environments. *Future Generation Computer Systems*, 24(7):687–693.

Williams, J. D. and Young, S. (2007). Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2):393–422.

Winder, L. F. (2004). *Hazard Avoidance Alerting with Markov Decision Processes*. Ph.D. thesis, MIT.

Yang, L. C. (2000). *Aircraft Conflict Analysis and Real-Time Conflict Probing using Probabilistic Trajectory Modeling*. Ph.D. thesis, MIT.