Planning under Uncertainty for Dynamic Collision Avoidance

by

Selim Temizer

M.S., Massachusetts Institute of Technology, 2001

B.S., Middle East Technical University, 1999

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2011

Author .....

Department of Electrical Engineering and Computer Science

January 26, 2011

Certified by .....

Leslie Pack Kaelbling

Professor of Computer Science and Engineering

Thesis Supervisor

Certified by .....

Tomás Lozano-Pérez

Professor of Computer Science and Engineering

Thesis Supervisor

Accepted by .....

Terry P. Orlando

Chairman, Department Committee on Graduate Students

Planning under Uncertainty for

Dynamic Collision Avoidance

by

Selim Temizer

Submitted to the Department of Electrical Engineering and Computer Science
on January 26, 2011, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

We approach dynamic collision avoidance problem from the perspective of designing collision avoidance systems for unmanned aerial vehicles. Before unmanned aircraft can fly safely in civil airspace, robust airborne collision avoidance systems must be developed. Instead of hand-crafting a collision avoidance algorithm for every combination of sensor and aircraft configurations, we investigate automatic generation of collision avoidance algorithms given models of aircraft dynamics, sensor performance, and intruder behavior. We first formulate the problem within the Partially Observable Markov Decision Process (POMDP) framework, and use generic MDP/POMDP solvers offline to compute vertical-only avoidance strategies that optimize a cost function to balance flight-plan deviation with risk of collision. We then describe a second framework that performs online planning and allows for 3-D escape maneuvers by starting with possibly dangerous initial flight plans and improving them iteratively. Experimental results with four different sensor modalities and a parametric aircraft performance model demonstrate the suitability of both approaches.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor of Computer Science and Engineering

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor of Computer Science and Engineering

Acknowledgments

A unique journey that I am grateful for:
Learning from the best, researching, teaching and more...
Late nights, psets, projects and coding were all fun,
Also, there were challenges that I had to overcome;
He—The Most Gracious—has looked after me, all along, all the time.

Brought to you by:

- Invaluable and continuous intellectual support from my advisors Prof. Leslie Pack Kaelbling and Prof. Tomás Lozano-Pérez. This thesis would not have been possible without their wisdom, supervision, help and patience.
- Extensive and timely technical support from Dr. Mykel J. Kochenderfer and Mr. J. Daniel Griffith. Dr. Kochenderfer also provided great guidance as a member of my thesis committee.
- Patience and help with many missed deadlines from Prof. Terry P. Orlando and Ms. Janet Fischer.
- Family love and support from my mother Güzide Temizer, my father Cevat Temizer and my brother Namık Kemal Temizer.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Systems that warn operators of cars, buses, trucks, trains and ships against possible collisions are being researched, developed, and are becoming available for more and more types and brands of vehicles each and every day [13, 47, 100, 97, 134, 116, 89, 29]. These systems provide safer transportation for the operator, the passengers, and the vehicle itself, usually by estimating traffic risks, detecting whether the eyes of the operator are closed or not, and whether the vehicle is properly following a straight path or swaying from side to side, and warning the operator against drowsiness and incoming traffic [2, 114, 136].

The damage caused by a crash between two or more vehicles increases with the weights and the speeds of the involved vehicles, hence it is more important to have a warning system to assist the operators of heavy and fast vehicles. Of land, sea and air vehicles, aircraft deserve special consideration when it comes to collision avoidance as aircraft are usually very heavy and very fast, and the chance of surviving a mid-air collision is low. Therefore most commercial and passenger-carrying aircraft are equipped with radars continuously scanning and displaying incoming traffic to visually help the pilots who are also usually assisted by ground-based air traffic controllers during the flights, and in addition to these, most commercial aircraft also carry warning systems that operate independent of the ground systems and help the pilots avert dangerous mid-air encounters.

In this document, we present collision avoidance algorithms for autonomously

controlling an unmanned aerial vehicle (UAV) to minimize collision risk during mid-air encounters with other aircraft.

1.1 Collision Avoidance for Unmanned Aircraft

Because of the potential for commercial, military, law-enforcement, scientific, and other purposes, unmanned aircraft have received considerable attention in recent years. However, unmanned aircraft are not currently permitted access to civil airspace in the United States without special permission from the Federal Aviation Administration (FAA). One of the primary concerns with integrating unmanned aircraft is their inability to robustly sense and avoid other aircraft. Although sensor information can be transmitted to a ground pilot who can then maneuver the aircraft to avoid collision, there are concerns about communication latency and reliability. In order to provide the high level of safety required by the FAA, an automated airborne collision avoidance system is likely to be necessary.

The deployment of any collision avoidance system requires a lengthy development process followed by a rigorous certification process. Development of the Traffic Alert and Collision Avoidance System (TCAS) [119], currently mandated onboard all large transport aircraft worldwide, started in the 1950s but was not certified for operational use until relatively recently [1]. The system issues vertical rate resolution advisories to pilots who are then responsible for maneuvering the aircraft. TCAS is not certified for autonomous use, and it is likely that the certification of an autonomous system will require even more extensive testing and analysis.

Further complicating the certification process of collision avoidance systems for unmanned aircraft is the diversity of their aircraft performance characteristics and sensor capabilities. Unmanned aircraft can range from under a pound to many tons with wildly varying flight dynamics. Several sensor modalities have been considered for supporting collision avoidance, including electro-optical/infrared (EO/IR), radar, TCAS, and Automatic Dependent Surveillance-Broadcast (ADS-B) [62, 42, 17, 119, 120, 82, 9]. As Table 1.1 illustrates, these sensor modalities vary in their capabili-

Table 1.1: Qualitative performance characteristics of various sensor modalities. FoV stands for *field-of-view*.

	Measurement Accuracy			Coverage		
Modality	Range	Azimuth	Elevation	FoV	Range	Traffic
TCAS	good	moderate	good	good	good	moderate
Radar	good	good	good	moderate	good	good
EO/IR	poor	good	good	moderate	moderate	good
ADS-B	good	good	good	good	good	moderate/poor

ties. It would be very difficult to develop and certify a different collision avoidance system for every combination of sensor configuration and aircraft platform. Current efforts in the unmanned aircraft industry have focused on proprietary solutions for specific platforms and sensors, but a common system that would accommodate different sensor configurations and flight characteristics would significantly reduce the cost of development and certification.

1.2 Challenges and Approach

In this document, we refer to the UAV that we control as *own aircraft* or *ownship* and to the other aircraft involved in the encounter as *intruder aircraft*. Major challenges of designing an autonomous collision avoidance system for own aircraft can be summarized as follows:

- We have a dynamical system and we need to take time into account in order to plan effective collison avoidance maneuvers.

- Most sensors have inherent measurement noise of different magnitudes depending on the sensor type and specifications. Therefore the detected positions and the estimated velocities of intruder aircraft have observational uncertainties in them. Moreover, usually there is also uncertainty about the *intention* of the intruder aircraft. For example, a hostile intruder might attempt to collide with ownship, a risk-aversive intruder such as one following TCAS resolution advi-

sories might attempt to increase vertical separation between itself and ownship, or an intruder that is oblivious to ownship might follow its regular flight plan. Our algorithms need to account for various possible intentions. For this purpose, we will work with worst case assumptions and adopt parametric random walk models to cover a large spectrum from oblivious intruders[1] to hostile intruders.

- All aircraft, including ownship, have nonholonomic motion constraints. A nonholonomic system in physics and mathematics, is a system whose state depends on the path taken to achieve it [22], therefore planning maneuvers requires not just deciding where to be at a given time, but also how to get there. The implications of nonholonomicity are twofold: On one hand, we can use this information to our advantage by limiting the locations that the intruder aircraft might occupy when we are estimating future states. On the other hand, we need to consider the limited mobility of ownship, too, and make sure that the planned escape maneuvers are feasible within the performance limits.

- Another very important challenge is the large size of the underlying state space of the collision avoidance problem. During the course of designing our algorithms and testing them using simulation software, we worked with up to 13 dimensional vectors to describe the state of a single aircraft. The components of our aircraft state vectors are listed in Table 1.2. The simplest collision avoidance problem involves two aircraft and hence the smallest true state space for an encounter has 26 dimensions. We also worked with realistic control commands shown in Table 1.3 and this necessitated the use of complex and realistic transition models in planning as well.

- As mentioned previously, there are many different types of sensor systems and UAVs which make designing collision avoidance systems difficult no matter whether we are hand-crafting individual algorithms for various different combinations of sensors and aircraft types or designing generic and parametric algo-

[1] This is actually a reasonable assumption for the current state of the global airspace, because, due to high cost and weight, many small UAVs do not carry the necessary transponder hardware that would enable them to inform the intruder aircraft about their presence and/or flight plans.

Table 1.2: Aircraft state vector.

Component	Explanation
v	True airspeed in ft/s
N	Position, north in ft
E	Position, east in ft
h	Position, altitude in ft
ψ	Orientation, yaw in rad
θ	Orientation, pitch in rad
ϕ	Orientation, roll in rad
$\dot{v}$	Airspeed acceleration in ft/s^2
p	Roll rate in rad/s
q	Pitch rate in rad/s
r	Yaw rate in rad/s
$\dot{h}$	Vertical rate in ft/s
$\ddot{h}$	Vertical acceleration in ft/s^2

Table 1.3: Aircraft control command vector. The first component of the control command can be either a vertical rate or a vertical acceleration. The simulation software that we used to test our algorithms is capable of working with both types of control commands.

Component	Explanation
$\dot{h}$ or $\ddot{h}$	Vertical rate in ft/s or vertical acceleration in ft/s^2
$\dot{\psi}$	Turn rate in rad/s
a	Airspeed acceleration in ft/s^2

rithms that can accomodate different sensor modalities and flight characteristics.

Having presented the major challenges, our approach to the problem will consist of the following key components:

Our first objective will be to answer the challenges stated above. Our algorithms will plan dynamical collision avoidance maneuvers. We will try to account for uncertainties in observations of intruder positions, velocities and intentions. The escape maneuvers will be feasible, i.e., ownship will be able to execute the planned maneuvers within its performance limits. In order to handle large problem space dimensionality, we will pick only the most relevant dimensions and come up with new representations

that capture and summarize important aspects of the problem that are sufficient for collision avoidance planning. We will design our algorithms to be parametric such that they will accomodate different sensor modalities and aircraft flight characteristics.

We will be working with realistic aircraft state vectors and control commands. We will also aim for designing algorithms that will work in real time such that they are suitable for deployment on real platforms.

There are avionic transponder systems that allow an aircraft to transmit and receive flight plans, intentions and planned escape maneuvers. If all aircraft in an encounter were equipped with such transponders, it would be possible and probably more effective to plan collision avoidance maneuvers for all aircraft at once. Such maneuvers are called *coordinated escape maneuvers* and the execution of a coordinated maneuver requires strict cooperation from all involved. In this work, we will assume that there is no cooperation between aircraft and we are planning only for ownship. However, some of our algorithms are also adequate for planning coordinated maneuvers and we will briefly make a note of them in respective sections.

Instead of the traditional way that collision avoidance algorithms have been designed, we will use a model-based approach to facilitate the design of algorithms that accomodate different sensors and flight dynamics. The traditional approach and the model-based approach are depicted and described in Figures 1-1 and 1-2, respectively. Briefly, a model-based design system takes as input models of flight dynamics, intruder behavior, and sensor characteristics, and attempts to optimize the avoidance strategy so that a predefined cost function is minimized. The cost function can take into account competing objectives, such as flight plan adherence and avoiding collision.

One way to formulate a problem involving the optimal control of a stochastic system is as a Markov Decision Process (MDP) [56, 110], or more generally as a Partially Observable Markov Decision Process (POMDP) to also account for observation uncertainty [3, 124, 128, 25, 50, 23, 64, 122]. POMDPs have been studied in the operations research and artificial intelligence communities, but only in the past few years have generic POMDP solution methods been developed that can approximately

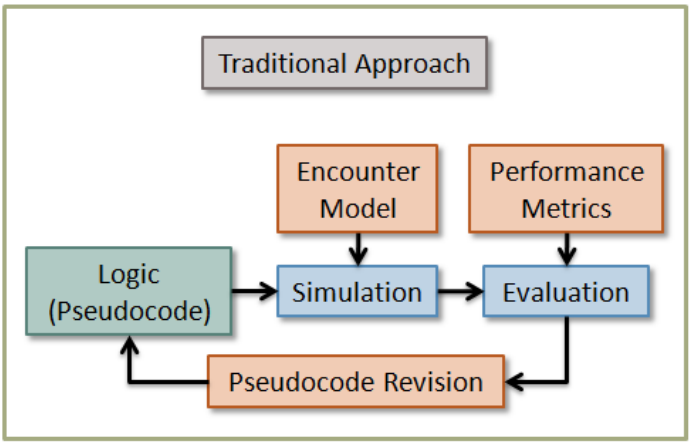


Figure 1-1: Traditional approach to designing collision avoidance algorithms. The input to the design process is the initial collision avoidance logic which is usually in the form of pseudocode. Human effort is spent on designing encounter models, developing performance metrics and revising collision avoidance logic. Simulations and evaluations are usually performed by computers. The design process consists of iterative improvements to the logic until desired performance is achieved. The output of the process is the improved collision avoidance logic.

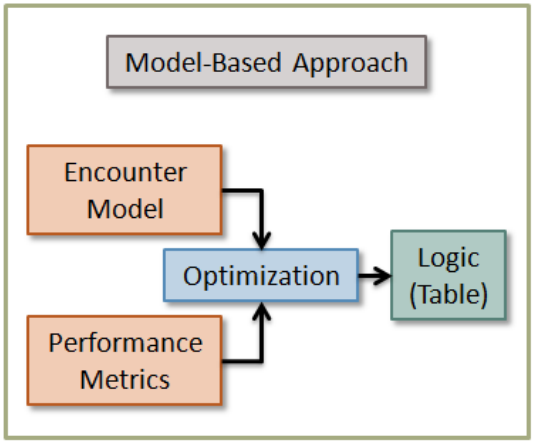


Figure 1-2: Model-based approach to designing collision avoidance algorithms. The input to the design process are encounter models and performance metrics, and human effort is spent on designing the input only. Computers perform the optimizations and it is desirable to do as much offline computation as possible. The design process ends as soon as the optimizations are completed. The output of the process is the optimized logic which might not be in a form that is easily interpreted by humans. For some of our algorithms that will be presented later, the output logic is a cryptic lookup table of high-dimensional vectors that is meant to be executed by special software.

solve[2] problems with moderate to large state spaces in reasonable time (for example the solvers we used [125, 80]). In this work, as our first approach, we will investigate the feasibility of applying state-of-the-art MDP and POMDP solution methods to the collision avoidance problem. Due to the fact that large-sized model spaces usually have a negative impact on the time required for solution and the effectiveness of the policy computed by MDP/POMDP solvers, we will limit our collision avoidance strategies to vertical evasion maneuvers only and compare our results against some baseline collision avoidance systems including TCAS, which also assists the pilots to do vertical-only evasive maneuvers. The experiments we will present in this document show that we can actually model collision avoidance systems using MDPs, and such systems perform very well in terms of both reducing the risk of collision and having very little deviations from the flight plan at the same time, especially with sensors that precisely locate intruder aircraft. We will also present experiments with POMDP models built for sensors with limited observation capabilities that demonstrate how we can still achieve low risk of collision by maneuvering a little more in order to counterbalance the limitations in observability of intruder aircraft.

The MDP and POMDP models we implemented in this study require working with finite number of states, control commands and sensor observations. Therefore, every input, output, and most intermediate results need to be chosen from discretized sets of values. Since the state space for collision avoidance problem is very high-dimensional and even the most powerful MDP/POMDP solvers cannot currently deal with very large sets yet, it is not possible to have nice and fine-grained discretizations of input and output spaces. There are also other negative effects of discretization that reduce the effectiveness of our collision avoidance algorithms as we will point out in the following sections. As a result of these observations about the MDP and POMDP models, our last objective will be set for investigating if we could design a hybrid collision avoidance system that would not require discretization of every data space and that could work on a mixture of continuous and discretized spaces as necessary.

[2] Approximate POMDP solution methods typically return solutions with bounded regret. Regret is the difference in expected cost between the returned solution and the optimal solution.

For this purpose, we implemented a technique that we call the *path-modification* or *spaghetti* method, which basically takes as input the estimated flight plans of all aircraft in an encounter, and outputs an optimized flight plan for ownship that tries to avoid risk of collision, not deviate much from the original flight plan, and minimize maneuvering at the same time. The experiments with algorithms based on the path-modification technique that we will present in this document show the feasibility of using this hybrid method to perform full 3-D evasion maneuvers (planning with full aircraft control commands as shown in Table 1.3, rather than planning for vertical-only maneuvers as we do with MDP/POMDP models).

1.3 Organization of the Thesis

The remainder of this document is organized as follows: In Chapter 2, we present a review of the MDP/POMDP framework that will be the basis for the first set of our algorithms, and a summary of previous work on collision avoidance techniques. Then we describe the parametric aircraft model, the sensor models, and the simulation and evaluation framework that we will work with. In Chapter 3, we build MDP/POMDP based collision avoidance systems with increasing complexities for the cases of perfect, noisy and limited field-of-view sensing, respectively. Chapter 4 describes the path-modification technique and two algorithms based on that technique for planning 3-D escape maneuvers. Finally, Chapter 5 delivers concluding remarks and recommendations for future research.

Chapter 2

Background

In this chapter, we will first present a brief review of MDP/POMDP framework. Then we will look at major approaches that have been applied to aircraft collision avoidance. The review of major approaches will be followed by an overview of the aircraft and sensor models that we implemented for use in our collision avoidance algorithms. Finally, we will introduce the simulation and evaluation framework.

2.1 Review of MDPs and POMDPs

An MDP is a stochastic process where the state of the system changes probabilistically according to the current state and action. MDPs assume that the state is fully observable. POMDPs remove that assumption and replace it with a stochastic model for observations, and hence they have more expressive power. We will briefly review POMDPs in this section.

The solution to a POMDP is a *policy*, or way of behaving, that selects actions in a way that takes into account both the current uncertainty about the underlying state of the system (e.g., exact relative position of the intruder aircraft), as well as future uncertainty about how the system state will evolve (e.g., what kinds of maneuvers the intruder aircraft will make), by aiming to maximize the expected accumulation of some predefined reward or minimize the expected accumulation of some predefined cost [64]. Due to their rich descriptive power, POMDPs have found many uses in

computer science and robotics applications such as robust mobile robot navigation [123], machine vision [6, 31], robust dialogue management [118, 55], autonomous helicopter control [4, 103], and high-level robot control [107], as well as in many other areas like machine maintenance [110], network troubleshooting [133], medical diagnosis [49], and preference elicitation [18]. Cassandra provides a comprehensive survey of applications utilizing POMDPs [24].

Several formulations of POMDPs have been studied in the literature, but this work focuses on the discrete-time formulation with discrete state and action spaces. We briefly present below a POMDP formulation and discuss solution techniques.

2.1.1 Formulation

In this document, we use $\mathcal{S}$ to represent the state space, $\mathcal{A}$ to represent the action space, and Ω to represent the observation space, all assumed discrete. The *state-transition function* $T : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ determines the probability distribution over the next states given the current state and action taken. The probability of transitioning to state s' after taking action a from state s is written $T(s, a, s')$. The *observation function* $O : \mathcal{S} \times \mathcal{A} \to \Pi(\Omega)$ determines the probability distribution over the observations received after taking some action resulting in state s'. The probability of receiving observation o after taking action a and landing in state s' is written $O(s', a, o)$.

In general, the initial state is unknown. The uncertainty in the initial state is represented by a probability distribution $b_0 : \mathcal{S} \to \mathbb{R}$, where the probability of starting in state s is written $b_0(s)$. Since the true state is not directly observable in POMDPs, the states are called *belief-states*, and similar to the initial state, they consist of probability distributions over the state space; $\mathcal{S} \to \mathbb{R}$. The space of all possible belief-states is denoted $\mathcal{B}$. The belief-state b is initialized to b_0 and updated with each observation according to Bayes' rule. If the current belief-state is b and action a is

taken resulting in an observation o, the new belief-state b' is given by

$$\begin{aligned} b'(s) &= \Pr(s' \mid o, a, b) \\ &\propto \Pr(o \mid s', a, b) \Pr(s' \mid a, b) \\ &= \Pr(o \mid s', a) \sum_{s \in \mathcal{S}} T(s, a, s') b(s) \\ &= O(s', a, o) \sum_{s \in \mathcal{S}} T(s, a, s') b(s) \,. \end{aligned}$$

The belief-update process is often referred to as *state estimation*.

Given the current belief-state, the objective is to chose an action that maximizes the *expected discounted return*. The discounted return for a sequence of states s_t and actions a_t is given by

$$\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,,$$

where $\gamma \in [0, 1)$ is a *discount factor* and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward function*. The reward for taking action a from state s is written $R(s, a)$.

The solution to a POMDP is a *policy* $\pi : \mathcal{B} \to \mathcal{A}$ that specifies which action maximizes the expected discounted reward given a belief-state. It is known that optimal policies can be represented as a collection of α-vectors, denoted Γ. Each α-vector is a vector consisting of $|\mathcal{S}|$ components and is associated with a particular action. The expected discounted return when starting with belief b is

$$V(b) = \max_{\alpha \in \Gamma} (\alpha \cdot b) \,,$$

where $\alpha \cdot b$ is the inner product of an α-vector with a vector representation of the belief-state. The function V is known as the *value function*. The policy evaluated at belief-state b is the action associated with the α-vector that maximizes the inner product.

2.1.2 Solution Methods

Finding the collection of α-vectors that represents the optimal policy can be challenging, even for relatively small problems. A variety of exact solution methods can be found in the literature [127, 124, 128, 48, 98, 113], but generally these methods do not scale well to large problems. Approximate solution methods generally scale much better and many of them provide bounds on the *regret* for the policies they find. The regret of a policy π is the difference between the expected discounted return starting at b_0 when following π and the expected discounted return starting at b_0 when following an optimal policy π^*.

Point-based methods for finding approximate solutions to POMDPs (for example, Point-Based Value Iteration, PBVI [106]) have received attention in recent years because of their ability to solve problems that are orders of magnitude larger than was previously possible. Point-based methods involve sampling from the belief space $\mathcal{B}$. The more successful point-based methods focus the sampling on belief-states. In this work we initially used solvers based on Heuristic Search Value Iteration (HSVI2) algorithm [125, 126]. We later switched to a solver that uses Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) algorithm [80, 58, 57] as it performed better on our problems. An implementation of SARSOP is publicly available[1] and we were able to use the software without any modification. SARSOP takes as input a textual representation[2] of a POMDP, including γ, b_0, R, T, and O. When the regret bounds fall below some preset value or the user interrupts the solution process, SARSOP outputs a policy file represented as a collection of α-vectors.

Crucially, although it may require considerable computation to find a near-optimal policy, this work is done offline. Once a policy has been computed, it can be executed very efficiently online. In the course of this work we have developed a new algorithmic technique to make the execution process even more efficient, making it entirely suitable for execution online, in real time, on an aircraft.

[1] M^{2}AP Research Group at NUS, *POMDP Planning*, http://motion.comp.nus.edu.sg/projects/pomdp/pomdp.html (August 2010).

[2] The format of the input is the same as the one described by Anthony R. Cassandra, *Input POMDP File Format*, http://www.pomdp.org/pomdp/code/pomdp-file-spec.shtml (August 2010).

Although our MDP/POMDP based collision avoidance algorithms have focused on finding α-vectors offline, there are other approaches to finding and representing policies. Online approaches decide what action to execute by searching only from the current belief-state, instead of trying to find a comprehensive policy that is optimal for all belief-states [117]. From the current belief-state, these methods explore different action sequences up to some horizon and then select the sequence that results in the largest expected discounted return. Computing the expected discounted return for an action sequence involves updating the belief-state based on hypothetical measurements obtained with each state transition. One concern with an online method that involves sampling might be the nondeterminism of the resulting behavior.

2.2 Previous Work

Collision avoidance is a fundamental part of motion planning, and hence there are many different approaches from ad hoc solutions to well-established methods. In this section, we will present a summary of major techniques that have been used for collision avoidance and discuss their advantages and disadvantages.

2.2.1 POMDPs and Dynamic Programming

Due to the large size and the continuous nature of the state, observation and action spaces in most collision avoidance tasks, classical POMDPs operating on discretized sets have been difficult to apply to realistic collision avoidance scenarios. To the best of our knowledge, our MDP/POMDP based algorithms are some of the first examples of application of the original POMDP formulation to a realistic UAV collision avoidance problem, where the models monitor a very large airspace and choose realistic control commands for maintaining a flight plan, collision avoidance, and information gathering. We were able to use the classical framework by choosing compact representations and carefully designing small state, action and observation spaces that contain sufficient information. This is a major difference of our models from the ones described below: Almost all of the following techniques differ in certain ways from

the discrete-time POMDP formulation in order to increase the size of the models that could be handled and/or work with continuous spaces.

One heuristic that is likely to be necessary in order to feasibly employ the discrete-time formulation for even larger problems is to hierarchically decompose the planning task [52, 83, 8]. With this approach, domain-specific knowledge could be leveraged to perform planning in macro and micro scales that are managed by different layers of the hierarchical planner.

If we assume that the world state that is 'most likely' in the current belief-state is in fact true, then we can take the optimal action for the state in the MDP that underlies the POMDP. Similar simplifications include Q-MDP [88] and value-function approximations [51]. One important problem with ignoring uncertainty about current state and assuming full observability is the loss of system's desire to explicitly take actions to reduce uncertainty. Platt et al. [109] employ the key idea of planning directly in belief-space, determinizing the dynamics by using the most-likely observation, and demonstrate a replanning approach to overcome that problem using optimization schemes like linear quadratic regulation [130], direct transcription [38] (solving control problems by treating them as optimization, based on nonlinear optimization methods [12]), and other standard planning/control techniques.

POMDPs with continuous state spaces, leveraging hybrid-linear system dynamics, have been developed and applied to UAV collision avoidance simulations by Brunskill et al. [20, 21]. In their study, a formal analysis with bounds on the quality of resulting solutions has also been presented. Erez and Smart take this approach further, and work with all continuous state, observation and action spaces [39]. They parametrize the belief-state as a mixture of Gaussians and use Differential Dynamic Programming [61] for local optimization. Such local optimization provides no guarantees of global optimality, but it accommodates domains that are much larger than those that could be solved feasibly by state-of-the-art solvers that require the discretization of state, observation and action spaces.

Wolf and Kochenderfer propose an online POMDP approach to collision avoidance [142, 143]. Online planning has the advantage of starting from current state and

searching only the reachable states instead of having to come up with a universal policy for all possible initial states. They utilize continuous state and observation spaces and a finite action space in their formulation, and they introduce sample-based representation of state uncertainty [135] to an existing algorithm called Real-Time Belief Space Search [105].

Kochenderfer et al. use a dynamic programming approach to generate optimized TCAS logic [71, 74]. They also provide guidance in justification of collision avoidance logic that is automatically generated by dynamic programming based solvers, which will be a very important issue as more complex solvers are being developed and used in optimizations. They extend their framework later to include more sophisticated actions, motion estimations in 3 dimensions, probabilistic pilot response, noisy sensor measurements, coordinated resolution maneuvers and multiple intruder scenarios [73].

2.2.2 Potential Field Methods

The artificial potential field approaches have been widely used in robot navigation planning since their introduction [70, 67, 68, 69, 84]. They have also been applied to aircraft collision avoidance [36, 37]. Typically, the problem is set up such that the target location exerts attractive virtual forces and the obstacles exert repulsive virtual forces. The controller then computes and commands to step in the direction of the net resultant force acting on own agent.

Potential field methods are very fast and they allow implementations of real-time planners very easily, but they have fundamental problems [76]. Most important limitations from the point of view of application to aircraft collision avoidance include the following:

- Potential field methods are prone to local minima problems. The attractive and repulsive forces might cancel each other and lead to a zero resultant force. There is a need to have higher-level planners to escape from such traps.

- Nonholonomic motion constraints might prevent the agent from being able to move immediately in the direction of the resultant force. This is an important

limitation, but a technique used by pilots for aircraft formation [40] might be utilized as a heuristic to alleviate the problem: Positioning is decomposed into fore-aft corrections (done by adjusting speed only) and side-side corrections (done by adjusting heading only) which can be applied independently. Balch and Arkin demonstrate the use of this type of corrections to navigate unmanned ground vehicles with nonholonomic constraints [5].

- Potential field methods can work well for slow-moving robots, but it is difficult to fully consider wide range of aircraft dynamics (including probabilistic dynamics of intruder aircraft) when they are applied to aircraft collision avoidance. Large virtual forces are necessary for repelling fast incoming traffic, but with slower intruders, this will cause unnecessary deviation from planned flight trajectories.

- Most importantly, uncertainty in control or observation might be challenging to model with sufficient fidelity for aircraft collision avoidance. It might be possible to account for uncertainties by increasing protected volumes around all aircraft (in the sense of configuration-space based spatial planning [92, 90, 91]), but in the last-minute collision avoidance context, it is not enough due to the short encounter time frame and the catastrophic nature of collision.

Charifa and Bikdash provide a comparison of several variants of artificial potential field approaches with emphasis on the quality of the path geometry, and velocity and acceleration profiles [28].

2.2.3 Sampling-Based Motion Planning

Sampling-based planning algorithms and especially Rapidly-Exploring Random Trees (RRTs) have been widely used because they tend to cover the search space more quickly than a random walk or other types of structured searches [7, 26, 66, 94, 59, 63, 85]. They are usually adequate for building real-time planners (for example, they have been applied to autonomous urban driving [81]).

RRTs generate random samples to explore the configuration space of the agent and they try to find a solution by extending and finally connecting one or more trees

rooted at the origin and at the destination configurations. Similar to our algorithms that we will present later, RRTs work very well with nonholonomic agents as they plan in configuration space.

Some fundamental issues with sampling have received increased attention recently [87], and further improvements have been suggested [86]. As a general condition, sampling-based methods make no guarantee of optimality of the found solution, and consideration of uncertainty in this framework is not yet mature enough to be fully feasible in the airborne collision avoidance problem domain.

2.2.4 Geometric Optimization

Bilimoria introduced a 2-D conflict resolution algorithm in horizontal plane using geometric computations [14]. Conflict predictions are based on straight-line projections using positions and assuming constant velocities. Computed resolutions consist of minimal changes in velocity to avoid a predefined circular protected airspace around intruder aircraft. Dowek et al. generalized this analytical approach to 3-D with cylinders replacing the circular protected zones, and full aircraft control commands rather than lateral-only maneuvers [45].

Geometric solutions to collision avoidance have the unique advantage of being extremely fast and very easily verifiable and validatable, but precautions such as adjusting the protected airspace sizes and breaking the constant velocity assumptions should be taken in order to account for uncertainties in sensing and intruder intent, and unexpected intruder dynamics. Another disadvantage of geometric planning is that, it might not be easy to scale up the approach to avert multiple threat. When there are multiple intruders, there seems to be three ways of approaching the problem, with each one having its associated difficulties:

- Protected airspaces around all intruder aircraft might be merged into a big protected zone (as in building a convex hull) that is to be avoided by geometric computations. There are basically three problems with this approach: First; the resulting protected zone could be very big and cannot be avoided within

the performance limits of ownship. Second; individual protected airspaces are projections of estimations through time, and hence they might shift around and/or shrink/grow in size as estimated positions and velocities of intruders are updated with each new observation. Third; the optimal (safest) trajectory that could be followed by ownship might fall within the convex hull, which will never be considered by the solver.

- Pairwise solutions against each individual intruder could be computed and heuristics could be developed to merge them into a global solution, but fundamental problems with this approach is described by Kuchar and Yang [79].

- A full 3-D global planning that aims to avoid each and every protected airspace is actually the optimal approach, but this turns the planning into a 3-D version of TCAS (which computes just vertical-only maneuvers and is already very complex).

2.2.5 Policy Search Methods

Given a parametric representation of a collision avoidance policy, a local search method known as policy gradient [102] can be used to search the parameter space for an optimal setting that minimizes the expected cost of following that policy. In this method, the state space does not need to be discrete and the policy could be represented very flexibly (for example, it can be a set of parametrized controls to be applied sequentially, or a functional pseudocode such as TCAS, or it can even be in the form of a neural network [53, 54]). Sample applications of policy search include autonomous helicopter flight [104] and aircraft collision avoidance planning [65, 140].

Aside from its benefits, policy search methods suffer from local minima problems as do all local optimization techniques. Also, the design of the parametric representation of a policy requires deep domain-specific knowledge, insight into problem structure and engineering judgment.

2.2.6 Mixed Integer Linear Programming

Spacecraft and aircraft trajectory optimization including collision avoidance can be expressed as a list of linear constraints involving integer and continuous variables, known as a mixed-integer linear program (MILP) [10, 139], which can then be solved using efficient commercial software [108]. Richards and How demonstrate a single aircraft collision avoidance application, and then generalize their approach to allow for visiting a set of waypoints in a given order, and also handling multiple aircraft planning [115]. Luders applies MILP formulation with non-uniform timesteps between target waypoints, and plans a detailed short-term trajectory and a coarse long-term trajectory for own aircraft [93].

As in the geometric optimization approaches, there is usually a protected airspace set up around each aircraft in the MILP formulations. The stochasticity that stems from uncertainties in observations, intruder intent, and unexpected aircraft dynamics could be handled by increasing the sizes of protected airspaces.

MILP formulations using a set of target waypoints that need to be visited in a certain order have a strong structural resemblance to our path-modification based collision avoidance models. An advantage of the MILP formulation over our models is its ability to plan with non-uniform timesteps between waypoints, since our waypoints are currently fixed in time. However, an important difference between the two approaches lies in the problem statement and the solver structure. The MILP approach requires all aspects of the problem (dynamics, ordering of all waypoints in time, and collision avoidance geometry) to be specified as a carefully designed and a usually long list of many linear constraints, and then the solver's task is basically to find a solution that satisfies all of those constraints simultaneously. The path-modification technique requires less information (just the aircraft dynamics and cost formulation) in the formulation stage, and the solver performs iterative optimization of an initial solution (planned flight).

Figure 2-1: Global Hawk.

2.2.7 Other Approaches

Other approaches to aircraft collision avoidance domain such as evolutionary algorithms [11] and nonlinear programming [112] can be found in the literature. Carlos et al. present a survey of a family of high performance controllers that is referred to as Model Predictive Control (MPC) [43], and examine their performances, advantages, and their application to nonlinear systems. Fujimura provides detailed general background information on motion planning in dynamic environments against stationary and dynamic obstacles [41]. Kuchar and Yang present an assessment of 68 air traffic conflict detection and resolution methods in their survey [79]. Kuchar also describes a unified methodology for the evaluation of hazard alerting systems in his thesis [77] that could be used in performance evaluation of miscellaneous and/or new future techniques that do not fit in any of the categories we have reviewed.

2.3 Aircraft and Sensor Models

The aircraft model we developed for our collision avoidance systems is parametric and can be modified to mimic different types of aircraft. In our implementation, parameter values are based on Global Hawk, an unmanned aerial vehicle used by the United States Air Force as a surveillance aircraft, shown in Figure 2-1.

Table 2.1 shows performance limits for Global Hawk. Our collision avoidance

Table 2.1: Global Hawk performance limits.

Maximum velocity	180 kts
Minimum velocity	100 kts
Maximum climb rate	3500 fpm
Maximum descent rate	4000 fpm
Maximum bank angle	35 deg
Maximum bank rate	8 deg/s
Maximum pitch rate	2 deg/s
Maximum turn rate	2.5 deg/s

models use a subset of these values; namely, maximum and minimum velocities, maximum climb/descent rates and turn rate. Our evaluation environment makes full use of them during encounter simulation.

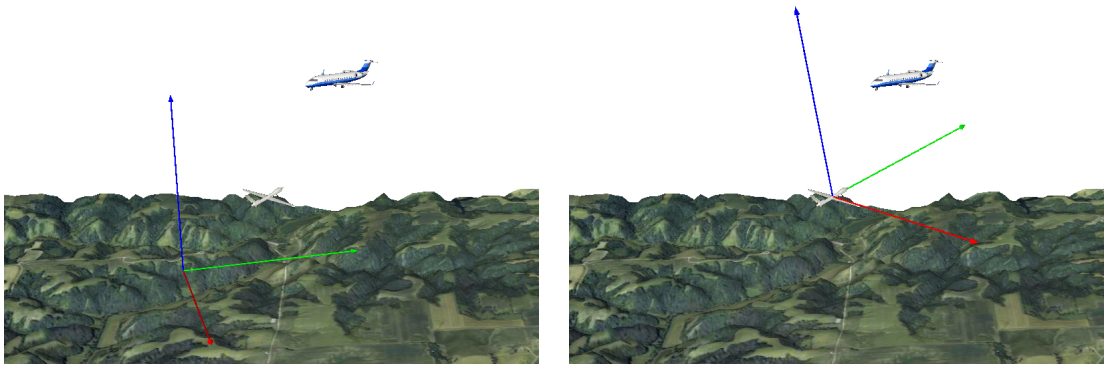
Before describing our sensor models, let us introduce four coordinate systems shown in Figure 2-2 that we will refer to from time to time in the rest of this document:

- **Global Coordinate System (GCS)**: This coordinate system is also known as the *Earth Coordinate System*. The origin is an arbitrary point chosen by the model simulation and evaluation framework. Positive x is *east*, positive y is *north*, and positive z is *altitude*.

- **Local Coordinate System (LCS)**: The origin of LCS is ownship center of mass (i.e., LCS is an *egocentric* coordinate system). Positive x is in the direction of the right wing, positive y is the direction of the nose, and positive z is upwards.

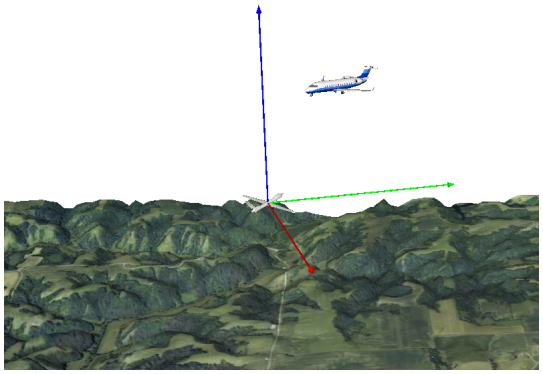
- **Auxiliary Coordinate System (ACS)**: This is also an *egocentric* coordinate system whose x-y-z axes are aligned with the *east*-*north*-*altitude* axes of GCS, respectively.

- **Relative Coordinate System (RCS)**: This is another *egocentric* coordinate system which is obtained by rotating ACS around its z axis until the y-z plane contains (intersects with) intruder aircraft center of mass. RCS is a 2-dimensional coordinate system. The x and y axes of RCS are the y and

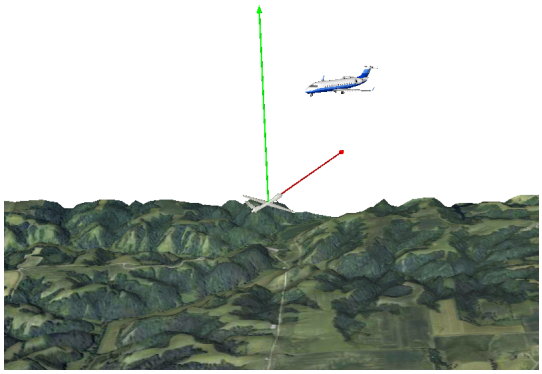


Global Coordinate System

Local Coordinate System



Auxiliary Coordinate System



Relative Coordinate System

Figure 2-2: Coordinate systems.

RCS (viewed from above) Projection Plane

Figure 2-3: Relative position of the intruder aircraft can naturally be represented by a point on *Projection Plane*.

z axes of the rotated ACS, respectively. The RCS is also referred to as the projection plane due to the fact that the vertical and horizontal distances to intruder aircraft can both be naturally projected on RCS to obtain a compact representation of aircraft separation as shown in Figure 2-3.

Input to our collision avoidance systems may come from various sensors with different characteristics and sensing ranges (usually expressed by radii in nautical miles, NM) onboard the UAV. We developed four detailed sensor models that are capable of simulating following types of erroneous measurements and noise:

- **False positive measurements**: We may detect an intruder when, in fact, there is no intruder aircraft in the sensor range (for example, a bird in sensing range might cause false positive measurements).
- **False negative measurements**: We may fail to detect an intruder when one is present in the sensor range.
- **Measurement errors**: We may detect the intruder aircraft in a position or at an angle that is not correct.

The probabilities of false positive and false negative measurements (p_{fp} and p_{fn}) are usually specific to different sensor hardware, and the measurement errors are computed according to realistic error models. In addition to false positive and false

negative measurements, one can think of a third type of false measurement: We may detect a different intruder (for example, a bird or some other random measurement) when there is a real intruder aircraft in sensor range. We excluded this case in our sensor models with the following assumptions:

- Sensors are tested for and free of this type of fault.
- If there are both a plane and a bird in the sensor range (and assuming that this is not a case of a false negative measurement), sensor will detect the plane since it is much bigger than a bird.

The four sensor models studied in this research are as follows:

1. **Perfect sensor**: This is a hypothetical omnidirectional sensor with no noise and no false positive/negative detections ($p_{fp} = p_{fn} = 0$). The sensor reading consists of *east*, *north* and *altitude* coordinates of intruder aircraft in GCS (it can be thought of as providing an abstract resemblance to the functionality of an ADS-B sensor). With this sensor, it is possible to localize intruder aircraft to an exact point in both GCS and LCS.

2. **TCAS sensor**: This is a model of the actual TCAS sensor [119]. It is based on listening to transponder replies from nearby aircraft and is omnidirectional. It provides *bearing* in LCS, *altitude* in GCS, and *range* (the line-of-sight distance between ownship and intruder aircraft, also referred to as *slant range*). The error in *range* measurement is Gaussian with zero mean and 50 ft standard deviation. The error in *bearing* estimate is Gaussian with zero mean and 10 deg standard deviation. The altitude of intruder aircraft is measured with 25 ft quantization. There is also an altimetry error bias that remains constant during an encounter with an intruder aircraft, and is Laplacian with zero mean and 40 ft scale. Probability density function for the Laplace distribution is shown in Figure 2-4. In the TCAS sensor model, $p_{fp} = 0$ (since detection is based on broadcast signals) and $p_{fn} = 0.01$. With a noiseless TCAS sensor, intruder aircraft could be localized to a point in LCS, but considering the given error

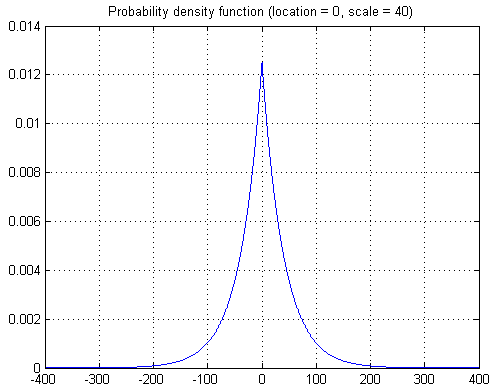


Figure 2-4: Probability density function for the Laplace distribution (location = 0, scale = 40).

model, the region that the intruder could be residing in has approximately the shape of a distorted truncated spherical cone.

3. **Radar sensor**: Our radar sensor model has a limited field-of-view (FoV), ±15 deg elevation and ±110 azimuth. It provides *bearing* and *elevation* readings in LCS, and *range* and *range rate* information. As with TCAS, the error in the *range* measurement is Gaussian with zero mean and 50 ft standard deviation. *Range rate* error is Gaussian with zero mean and 10 ft/s standard deviation. The error in the *bearing* estimate is Gaussian with zero mean and 10 deg standard deviation. *Elevation* error estimate is Gaussian with zero mean and 1 deg standard deviation. For the radar sensor, $p_{fp} = p_{fn} = 0.01$. Intruder aircraft can be localized approximately into a distorted truncated spherical cone in LCS.

4. **Electro-optical/infrared (EO/IR) sensor**: Our EO/IR sensor model is very similar to the radar sensor with less angular measurement noise and without a *range* reading. It has a limited FoV, ±15 deg elevation and ±110 azimuth. Sensor reading consists of *bearing* and *elevation* angles in LCS, and *line-of-sight rate* information. Error in both angular measurements is Gaussian with zero mean and 0.5 deg standard deviation. *Line-of-sight rate* error is Gaussian with zero mean and 0.5 deg/s standard deviation. For the EO/IR sensor, $p_{fp} = p_{fn} = 0.01$.

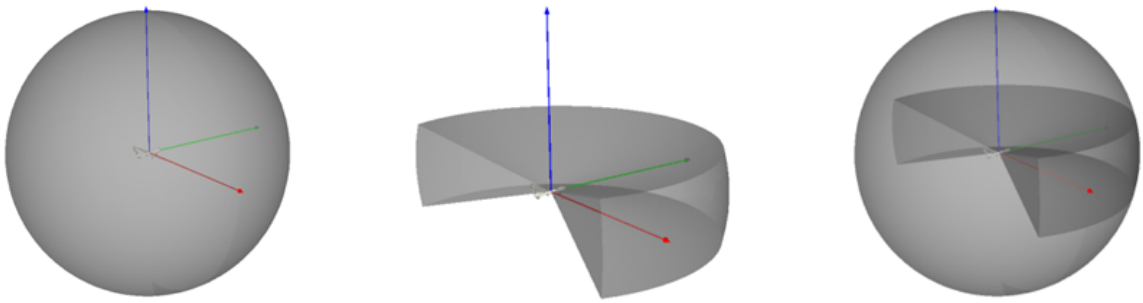


Figure 2-5: Comparison of sensing regions. The figure on the left shows omnidirectional sensing region, the figure at the center shows limited field-of-view sensing region, and the figure on the right shows both sensing regions overlapped for better comparison.

> Intruder aircraft can be localized approximately into a distorted spherical cone in LCS.

Figure 2-5 shows a comparison of the omnidirectional and limited field-of-view sensing regions. Complete list of sensor parameter values are given in Table 2.2.

2.4 Simulation and Evaluation Framework

The performance of our collision avoidance systems were evaluated using a simulation framework called Collision Avoidance System Safety Assessment Tool (CASSATT). The framework was developed for assisting prior TCAS studies [121] and evaluating sense-and-avoid systems for unmanned aircraft [15] at Lincoln Laboratory at Massachusetts Institute of Technology.

We used an encounter model derived from 9 months of national radar data [75] to generate 15,000 scripted encounters between pairs of aircraft and allowed our collision avoidance systems to control one of the aircraft. For comparison, we evaluated the performance of other collision avoidance systems to baseline performance. This section describes our simulation and evaluation process.

Table 2.2: Complete list of sensor parameter values.

Perfect		
Range	5	NM
False positive measurement probability	0.00	
False negative measurement probability	0.00	
TCAS		
Range	5	NM
Altitude quantization	25	ft
Range error standard deviation	50	ft
Bearing error standard deviation	10	deg
Altimetry error scale	40	
False positive measurement probability	0.00	
False negative measurement probability	0.01	
Radar		
Range	5	NM
Minimum azimuth	−110	deg
Maximum azimuth	110	deg
Minimum elevation	−15	deg
Maximum elevation	15	deg
Range error standard deviation	50	ft
Bearing error standard deviation	1	deg
Elevation error standard deviation	1	deg
Range rate error standard deviation	10	ft/s
False positive measurement probability	0.01	
False negative measurement probability	0.01	
EO/IR		
Range	5	NM
Minimum azimuth	−110	deg
Maximum azimuth	110	deg
Minimum elevation	−15	deg
Maximum elevation	15	deg
Bearing error standard deviation	0.5	deg
Elevation error standard deviation	0.5	deg
Line-of-sight rate error standard deviation	0.5	deg/s
False positive measurement probability	0.01	
False negative measurement probability	0.01	

2.4.1 Simulation Framework

CASSATT framework was built in Matlab and Simulink and has been compiled into native code using Real-Time Workshop. The framework was designed to be modular to allow different collision avoidance systems and sensor models to be easily incorporated. As part of this work, we extended CASSATT to allow communication with the collision avoidance system over a TCP/IP socket connection. This extension allows changes to be made to the collision avoidance system without having to recompile the remainder of the CASSATT system. The collision avoidance system runs as a server to which CASSATT connects as a client. Socket communication also allows CASSATT to run on a different machine from the collision avoidance system; for our experimentation however, we always ran the collision avoidance system on the same machine as CASSATT.

Figure 2-6 provides an overview of the simulation framework. An encounter model is used to generate initial conditions and scripted maneuvers for both aircraft involved in the encounter. These initial conditions and scripts are fed into a 6 degree-of-freedom, point-mass dynamic model. The sensor model takes as input the current state from the dynamic model and produces an observation, or sensor measurement. The state estimation process updates the internal state estimate of the collision avoidance system based on the observation. Then the collision avoidance system selects the control command that minimizes some cost depending on the algorithm used. Finally, the dynamic model updates the simulation state, and the process continues until the end of the encounter.

Encounter Model

Initial conditions and scripted maneuvers for both aircraft are generated by an encounter model. The initial condition for each aircraft is basically an aircraft state vector that was introduced before in Table 1.2. A scripted maneuver consists of a set of aircraft control commands described in Table 1.3 and the associated times that each command is to be applied during the encounter. In the simulations, scripted maneu-

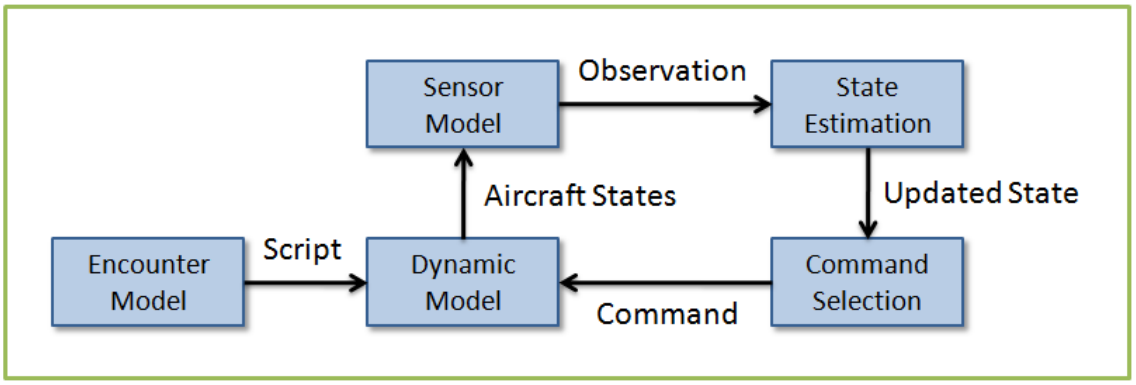


Figure 2-6: Simulation framework.

vers represent the Air Traffic Control (ATC) commands to each aircraft. The intruder aircraft always follows its script for the whole duration of the encounter. Ownship, however, is allowed to choose to follow the ATC commands or apply different control commands selected by the collision avoidance system.

We used a recently developed encounter model derived from 9 months of radar data from over 120 sensors [75]. A dynamic Bayesian network [101] representing the behavior of the aircraft was learned [44] from actual encounters extracted from the dataset. Generating new encounters for use in Monte Carlo analysis involves sampling from this dynamic Bayesian network.

Dynamic Model

Aircraft dynamics are represented using a tunable 6 degree-of-freedom, point-mass dynamic model, which includes aircraft transient response characteristics and performance limits such as maximum pitch rate or bank angle. The aircraft flight trajectories are defined by an encounter model and based on vertical rate, aircraft turn rate, and airspeed acceleration. These control values may change every tenth of a second.

Sensor Model

The sensor simulation module takes as input the raw (non-noisy) coordinates of both aircraft in GCS. First, the position of the intruder aircraft is computed (relative to ownship) in LCS. These intermediate coordinates are then converted into a simulated

sensor reading by adding noise according to the respective sensor's error model as described in Section 2.3. The pseudocode for the simulation of sensor readings is provided as Algorithm 1 in Appendix A. If we are evaluating a POMDP collision avoidance system (rather than a baseline or a path-modification based system) a final step takes the sensor reading, and generates an observation $o \in \Omega$ to be used in state estimation.

State Estimation

The state estimation module estimates the current state based on the measurement from the sensor. For path-modification based collision avoidance models, internal state is updated according to the sensor reading. For an MDP model, the current state is observed directly, and for a POMDP model, the belief-state is updated. Updating the belief-state usually requires iterating over large tables, and computing and normalizing probability values. Therefore, an important practical aspect of the belief-state update process is the overall computation time. This becomes even more crucial in real-time applications such as our collision avoidance systems. In this work, we developed a method that significantly reduces the computation time of the belief-state update process by merging the transition and observation models into a single look-up table that is generated offline and stored using sparse data structures [131, 132] (described in Appendix C). With this method, we have experienced belief-state updates that are 100 to 1000 times faster than a naïve implementation.

Command Selection

The path-modification based algorithms pick a command from the set of available commands by computing various cost terms and trying to minimize the expected cost. For MDP/POMDP based collision avoidance algorithms, command selection is done by evaluating the policy to compute the action, i.e. to choose the control command, to be executed. For MDP models, policy evaluation is carried out by simply referencing the *MDP policy*, which is basically a table that has an action $a \in \mathcal{A}$ assigned to each state $s \in \mathcal{S}$. For POMDP models, the process consists of finding the

α-vector that maximizes the expected long-term reward given the current belief-state, as described in Section 2.1.1. The output is the action associated with that α-vector.

As we mentioned in Section 2.1.2, there are powerful solvers available for POMDP formulations such as SARSOP and HSVI2 solvers, and the solutions generated by these solvers are called *policies*. Computing a policy with very tight regret bounds usually takes a very long time, sometimes on the order of hours or days, especially for large POMDPs such as the ones we use in our algorithms. Therefore, most POMDP solvers usually generate a simple solution first (probably with loose regret bounds), and iteratively improve that solution allowing the user to stop the process when tight-enough bounds are reached. Iterative improvement allows some solvers to work up to a given time limit or until a specified bound is reached. Policies are usually computed offline.

Policy evaluation is a process that takes two inputs and generates a single output. The inputs are a belief-state and a policy. The output is the action that we should take in order to maximize the expected long-term reward. We sometimes refer to the output as the *best action*.

The space of all belief-states for a given POMDP formulation is called the *belief simplex*. An optimal policy maps belief-states (which correspond to points in the belief simplex) to actions that maximize expected long-term reward. Due to the continuous nature of the belief simplex, policies are usually represented as a collection of regions of belief simplex, and the associated actions that should be taken when the given belief-state falls inside those regions. More specifically, a practical implementation of a policy consists of a set of α-vectors with an action associated with each α-vector. An α-vector serves two purposes:

1. The maximum cardinality for an α-vector is the number of states in the POMDP formulation. Usually, most of the α-vectors of a policy contain less than the maximum number of entries. The entries that are present in an α-vector determine the region of the belief simplex this α-vector is applicable to. Usually a policy contains a single α-vector or a small number of α-vectors with maximum cardinality. These vectors are applicable to all of the belief simplex (all possible

belief-states). The rest of the α-vectors in the policy are specialized to different sub-regions. Note that different sub-regions might overlap, and they are not required to cover the belief simplex. Also, a policy may contain more than one α-vectors that are all applicable to the same region.

2. The inner product of an α-vector and the belief-state yields the expected long-term reward in case of taking the action associated with that α-vector.

As a result, the algorithm for policy evaluation takes the following form:

- Determine the applicable α-vectors for the given belief-state.
- Compute expected long-term rewards by taking inner products of all applicable α-vectors with the given belief-state.
- The best action is the one that is associated with the α-vector that yields the highest expected long-term reward.

Since policy evaluation is also executed frequently similar to the state estimation process, we implemented a time-efficient data structure for working with policies, as well. Our design leverages the special data structure for belief-states, and allows us to quickly compute the best action.

2.4.2 Importance Sampling

Monte Carlo safety studies of collision avoidance systems generally involve exposing a collision avoidance system to a collection of encounters selected from some distribution $p(x)$. If $f(x)$ is the probability encounter x leads to a near mid-air collision (NMAC) and $x_1, \ldots, x_N$ are encounters chosen independently from $p(x)$, then the probability of an NMAC may be estimated as follows:

$$P(\text{NMAC}) = \frac{1}{N} \sum f(x_i) \,.$$

To test our algorithms, we used the encounter model developed by MIT Lincoln Laboratory for cooperative aircraft [75]. Most of the encounters generated by this

encounter model do not result in an NMAC. We would need to sample from the $p(x)$ encoded by the model many times before generating a test case that results in an NMAC. To reduce the number of samples required before we generate an "interesting" encounter, we sample from an alternative distribution $q(x)$ that focuses on encounters with low vertical and horizontal miss distances at the time of closest approach. Because we are no longer sampling from $p(x)$ we need to weight the samples in order to produce an accurate estimate of $P(\text{NMAC})$:

$$P(\text{NMAC}) = \frac{1}{N} \sum f(x_i) p(x_i)/q(x_i) \,.$$

This approach is known as *importance sampling*, and it results in a better estimate of $P(\text{NMAC})$ using fewer samples [16, 32].

We generated 15,000 encounters from the encounter model using importance sampling. In the future, we would like to test our system using at least hundreds of thousands of samples to provide better performance estimates. Because the encounters may be simulated in parallel, we used the parallel computing environment at MIT Lincoln Laboratory, known as *LLGrid*. Using 64 compute nodes, it takes approximately 10 minutes to evaluate one of our MDP/POMDP models on 15,000 encounters.

2.4.3 Baseline Collision Avoidance Systems

We compared the performance of our collision avoidance algorithms against the following baseline systems:

- **TCAS Version 7:** The TCAS Version 7 system uses only the TCAS sensor readings as input. The behavior of this system is as specified in the TCAS II standard [119].

- **Basic Collision Avoidance System (Basic CAS)**: It is possible to use all four sensor models with Basic CAS, but the performance decreases severely with the limited field-of-view sensors. The collision avoidance logic is very simple: If an intruder aircraft is detected inside the sensing region, and the projection

of the intruder position on RCS has a positive y value (i.e., the intruder is "above"), then ownship accelerates down with 0.25 g until next observation is received. Similarly, if the intruder is "below" (projection of its position on RCS has a negative y value), then ownship accelerates up with 0.25 g until the next observation.

- **Analytic Collision Avoidance System (Analytic CAS)**: Analytic CAS is based on collecting position data for ownship and intruder aircraft, and estimating their motion (velocities and accelerations) in full 3-dimensional coordinates by simple differentiation. Therefore, it is best suited for use with perfect and TCAS sensors, which are omnidirectional, have none or very little vertical noise (compared to other sensor models) and hence allow the intruder to be localized vertically with high accuracy at each simulation step. The collision avoidance logic works as follows: Based on regularly collected and updated position, velocity, and acceleration estimates, a *clear-of-danger* test is performed using simple quadratic equations of motion at each simulation step. If there is no danger of a collision or a close encounter in the future, ownship continues to follow the scripted maneuver, but if the test fails (i.e., the minimum distance between the extrapolated trajectories of both aircraft is below some threshold), an evasive maneuver is performed, which is simply to increase ownship's altitude by 200 ft as quickly as possible within the performance limits. After the maneuver is completed, the collision avoidance logic resumes monitoring and triggers further evasive maneuvers as necessary. We implemented two versions of the *clear-of-danger* test: The first version, called Analytic CAS 1-D, checks if only the vertical distance between two aircraft will drop below a threshold, and the second version, called Analytic CAS 3-D, checks if the intruder will invade a predefined 3-D volume surrounding ownship (which is usually in the shape of a hockey puck that is 200 ft thick and 1000 ft in diameter).

Like the MDP/POMDP algorithms, the above baseline systems perform evasive maneuvers only in the vertical plane, i.e. they only modify the *vertical rate* component

of the scripted maneuvers (ATC commands) to steer away from danger. We also implemented another very simple basic collision avoidance system that performs full 3-D evasive maneuvers by using the following guidelines:

- accelerate down/up if intruder aircraft is above/below,
- decrease/increase airspeed if ownship is moving towards/away from the intruder aircraft, and
- turn nose away from intruder aircraft.

We used this **Basic 3-D** Collision Avoidance System in comparisons with our path-modification based algorithms.

Chapter 3

MDP/POMDP Based Collision Avoidance Models

In this chapter, we will construct MDP/POMDP based collision avoidance models of increasing complexity for different sensor types. First, we will look at the case where ownship is able to detect the intruder with no noise using a perfect sensor. Then, we will build models that handle noisy observations from a TCAS sensor. And finally, we will design models that work with limited field-of-view sensors.

3.1 Perfect Sensing

The first case we will consider is sensing with no noise, and for that purpose we will assume that ownship is equipped with a perfect sensor. When there is no observation uncertainty, we can model the collision avoidance system as an MDP. Note that we allow uncertainty about the behavior of the intruder aircraft, and MDP formulation lets us capture this uncertainty in the state-transition model. In this section, we will look at the general structure of the state and action spaces and the details of the reward and state-transition models that will form our MDP collision avoidance system.

3.1.1 MDP Collision Avoidance System

As we mentioned before, the true state space model in the collision avoidance problem is continuous and consists of the following aircraft state vector components for both aircraft present in the encounter:

- Position specified in GCS
- Orientation specified as yaw, pitch, and roll angles
- Air speed, air speed acceleration
- Vertical rate, vertical acceleration
- Yaw rate, pitch rate, and roll rate

This is a very high-dimensional continuous space (26 dimensions for both aircraft together). The action space for a UAV is also continuous as it is possible to choose and apply any vertical and/or horizontal accelerations and turn rates within ownship's performance limits.

For our MDP/POMDP collision avoidance systems, we consider a simplified version of the problem in which ownship can only maneuver vertically, but not in azimuth, to evade intruders, similar to TCAS II. We also work with discretized spaces with less number of dimensions that are carefully selected to incorporate important information from the true spaces.

State Space

The size of a discretized state space is exponential in the dimension and in the case of 26 dimensions, we could not have even two discrete values per dimension. So, before we discretize the state space, we must first represent it in a much lower-dimensional subspace that captures the essence of the encounter.

To encode relative positions and velocities of the aircraft, we chose RCS as our main representation. In this coordinate system, the state consists of the following components:

- X : Horizontal distance from ownship to intruder aircraft;
- Y : Vertical distance from ownship to intruder aircraft;
- V_X^{Relative} : Relative velocity in X, representing the horizontal closure rate;
- V_Y^{Intruder} : Vertical velocity of intruder aircraft; and
- V_Y^{Ownship} : Vertical velocity of ownship.

This 5-dimensional state space is discretized by dividing each dimension into a finite number of *bins*. The sizes of the bins may be non-uniform. The overall state-space is then a set of 5-orthotopes (5-dimensional boxes or hyperrectangles) that exhaust a continuous piece of the overall 5-dimensional state space. We augment the state space with two sets of special states: START states and DONE states. These states are used to model situations when the state space is initialized (and the encounter has not started), and when the encounter is over, respectively. Because the vertical velocity of ownship is always known, we always include it in the state space. So, the START and DONE state sets both contain a member for each bin of V_Y^{Ownship} , modeling flight at some vertical velocity before the start of or after the termination of an encounter. Having discretized the state space in this way, a state may be represented simply as an index into the set of boxes spanning the space, or an index to one of the START or DONE states. The structure of the state space is shown in Figure 3-1.

When we use perfect sensor, the state is observed directly as described by Algorithm 2 in Appendix A.

Action Space

We adopted a simple discrete action-space model that consists of commands to ownship to apply positive or negative fixed vertical accelerations for a fixed duration (usually 1 s). For the MDP CAS, our action space consists of 17 uniform samples from the $\pm 8\,\text{ft/s}^2$ ($\pm 0.25\,\text{g}$) acceleration range imposed by the aircraft performance

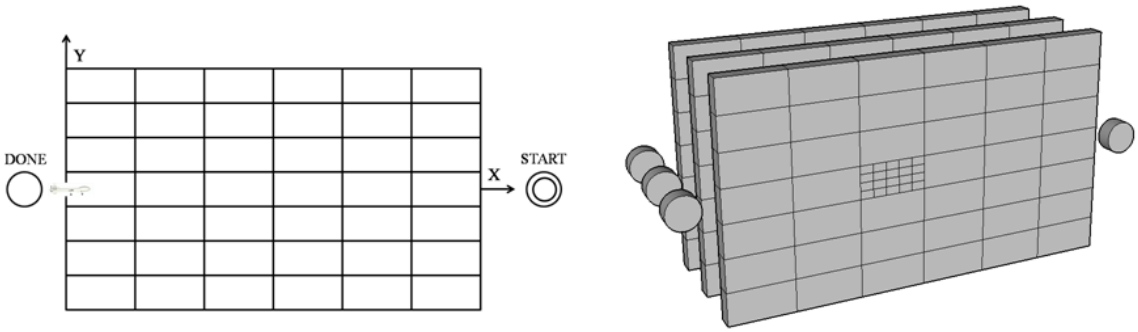


Figure 3-1: Structure of the state space. We begin constructing the state space with discretizing RCS in bins and adding the symbolic states, START and DONE , as shown on the left. We then create a duplicate of each state for each V_Y^{Ownship} bin as shown on the right. We can think of this step as creating layers. Since V_Y^{Ownship} is always known (except for the initialization step where we have not received an observation yet), only one of the layers is active at any given time. Lastly, we once more duplicate all X-Y bins for all combinations of V_X^{Relative} and V_Y^{Intruder} . Note that in the figure on the right, only one X-Y bin is duplicated as an illustration.

limits; $\mathcal{A} = \{-8, -7, \cdots, -1, 0, 1, \cdots, 7, 8\}$. It is possible to sample the range of vertical accelerations more densely, but the solvers would require more time to compute policies.

Reward Model

The reward function in our MDP formulation is in the form of costs (or negative rewards) rather than positive rewards. It is designed with the following three objectives in mind:

- As the primary goal of the collision avoidance algorithm, the intruder aircraft should never occupy the same bin as ownship in the RCS, which implies a collision or a very dangerous encounter. Note that ownship resides at the origin of the RCS, and it is possible that the origin might be on the edge or vertex of one or more bins rather than being inside a single bin due to the chosen vertical and horizontal division strategy. In that case, the collision avoidance algorithm should prevent the intruder from moving into any one of the bins that have any boundaries touching the origin.

- In addition to preventing collision, it is desirable to maintain some protected airspace around ownship where the intruder aircraft should not penetrate. This protected airspace is specified by two parameters: a vertical separation range and a horizontal separation range. In our tests for MDP CAS, we used 100 ft vertical and 500 ft horizontal separation ranges, same as that of the NMAC definition used in prior TCAS safety studies [99, 35, 96, 30]. The second goal of the collision avoidance algorithm should be to prevent other aircraft moving into any bin that has some parts overlapping with the protected airspace.

- As the last goal, if there is no danger of collision or penetration of protected airspace, ownship should level off and try to maintain a zero vertical velocity. It may be argued that ownship should try to return to its commanded flight path. We have taken the position that, during the handling of a close encounter, it is enough to prefer level flight, and that after the encounter is over, standard navigational procedures can be resumed.

In order to satisfy these goals, the reward may be specified as a function of the state of the system. It is specified using three user-defined parameters:

- **Collision cost**: The cost of any state in which the intruder is in the same X and Y bins as ownship, set to −1000;

- **Protected airspace violation cost**: The cost of any state in which the intruder aircraft is within the protected airspace region in X and Y, set to −500; and

- **Vertical velocity penalty**: The cost for being in a state where the V_Y^{Ownship} bin does not contain 0 ft/s; for the MDP CAS, vertical velocity penalties are linearly proportional to the velocity values that correspond to the centers of the V_Y^{Ownship} bins. It is possible to vary the maximum penalty value in order to reach different equilibria in balancing evasive maneuvers and level flight.

The reward model is illustrated in Figure 3-2.

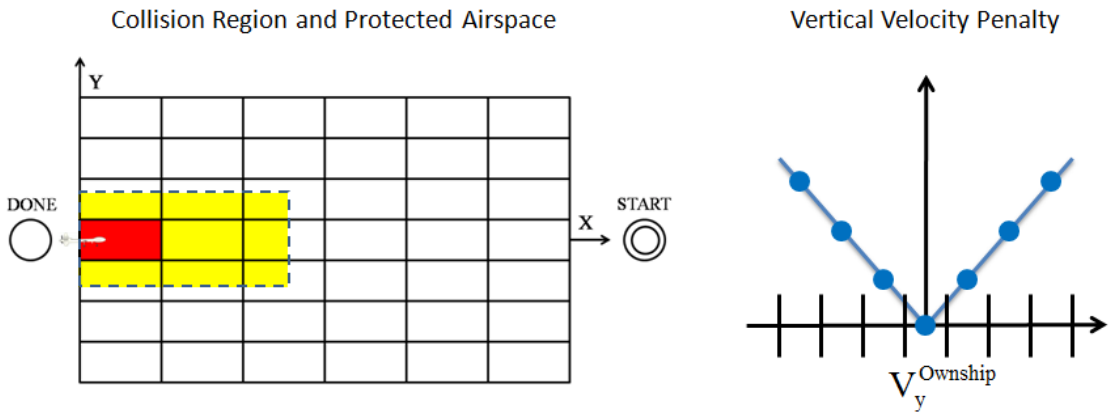


Figure 3-2: Reward model. On the left are the collision region (the bin that contains ownship, shown in red) and the protected airspace (shown as a yellow rectangle with dashed edges). On the right is the vertical velocity penalty function.

All other states are assumed to have a reward of 0. Note that the solution to the MDP will remain the same for any linear scaling of reward values, so only the relative magnitudes have an effect.

In order to emphasize the importance of avoiding crashes at any time (rather than simply trying to postpone them), we used a discount factor of 0.99.

State-Transition Model

The initial state distribution specifies that the system starts in a uniformly chosen START state. At each step, an action is taken and the probability distribution over the state space is updated according to the state-transition model.

Our assumption is that there is no actual stochasticity in the dynamics of the system. However, we model the uncertainty in intruder behavior as a random process; and the fact that the state space is discretized will introduce uncertainty in the transitions, even though they are governed by a deterministic physical process.

Our state-transition model is characterized by the following parameters:

- Controller frequency, ΔT: Duration between successive consultations of the MDP policy for choosing an action. This value is used by the MDP formulation to predict what the state will be in the next iteration.

Table 3.1: Horizontal and vertical acceleration models for intruder aircraft.

Horizontal Model	
$\dot{v}$ (ft/s^2)	Probability
−300.0	0.05
−200.0	0.05
−100.0	0.05
−30.0	0.10
−20.0	0.10
−10.0	0.10
0.0	0.10
10.0	0.10
20.0	0.10
30.0	0.10
100.0	0.05
200.0	0.05
300.0	0.05

Vertical Model	
$\dot{v}$ (ft/s^2)	Probability
−10.0	0.1
−5.0	0.2
0.0	0.4
5.0	0.2
10.0	0.1

- Magnitude of our vertical acceleration, A_Y^{Ownship} .
- Our vertical velocity limits, $V_{Y,\text{ Min}}^{\text{Ownship}}$ and $V_{Y,\text{ Max}}^{\text{Ownship}}$.
- Probability of staying in START state when already in START state.
- Probability of making a transition into any other state when in START state.
- Intruder aircraft's horizontal and vertical acceleration models.

For the horizontal and vertical acceleration models, we used the distributions given in Table 3.1. These distributions roughly model a random walk process where the intruder aircraft is oblivious to ownship or we have no idea about the intention of the intruder aircraft.

Given these parameters, we compute $\Pr(s' \mid s, a)$ as follows:

- First, we consider each possible pair of vertical and horizontal accelerations a_o that might be chosen by the intruder aircraft, and compute their probabilities p_o as the product of the probabilities in the intruder acceleration models.

- For each vertex of the bin s, we determine how that particular point in state space would be transformed given the execution of ownship acceleration a, and the intruder accelerations a_o.

- The result is a new box, B, in 5-dimensional space. For each new state s', we compute the percentage of B that overlaps s'; that overlap percentage is $\Pr(s' \mid s, a, a_o)$. Any probability mass outside the boundaries of the modeled state space is assigned $\Pr(\textsc{done}, V_Y^{\text{Ownship}} \mid s, a, a_o)$.

- Finally,

$$\Pr(s' \mid s, a) = \sum_{a_0} \Pr(s' \mid s, a, a_o) p_o \,.$$

This method of analytically computing the physical evolution of the system eliminates introducing additional discretization in the computation. Therefore, the effectiveness of the state-transition model depends only on the discretization of the state and action spaces and the fidelity of the vertical and horizontal acceleration models for the intruder aircraft. Having the acceleration models match closer to the actual intruder behavior results in better state estimations, since the intruder aircraft would be localized more accurately. The state-transition model is summarized as Algorithm 6 in Appendix A.

3.1.2 Results

Table 3.2 summarizes the results of nominal flight (ownship following the scripted flight path without using any collision avoidance systems) and baseline collision avoidance systems on 15,000 encounters. The table shows the risk ratios, mean vertical velocity magnitudes in ft/s, and mean vertical acceleration magnitudes in ft/s^2 for different algorithms. The risk ratio associated with a particular system is the probability that an encounter leads to an NMAC using the system divided by the probability that an encounter leads to an NMAC without the system. Of course, better performance is indicated by a small risk ratio. It is desirable to have velocity and, if possible, also acceleration values as small as possible without sacrificing the risk ratio (we would

Table 3.2: Risk ratios for nominal flight and baseline collision avoidance systems.

Algorithm	Ratio	Velocity	Acceleration
Nominal	1.000000	4.255460	0.172020
TCAS II (2500 ft/min)	0.061220	5.094360	0.345920
TCAS II (1500 ft/min)	0.062730	4.586190	0.366110
Basic CAS (perfect sensor)	0.000010	33.030760	0.790190
Analytic CAS, 3-D (perfect sensor)	0.054560	4.564330	0.224730
Analytic CAS, 1-D (perfect sensor)	0.016970	5.597470	0.768990

like to remind that the reward model we designed was not structured to penalize high accelerations). Large values of mean velocity magnitude indicate that ownship is maneuvering unnecessarily.

We experimented with gradually increasing the size of the state space (by increasing the number of bins along different dimensions in our discretization) until the time it takes for the solver to compute a policy increases beyond practical limits, and we ended up with an MDP model with 6768 states: 5, 10, 3, 5, and 9 bins for X, Y, V_X^{Relative} , V_Y^{Intruder} , and V_Y^{Ownship} components of $\mathcal{S}$, respectively, and 9 START and 9 DONE states. Solving an MDP using *value iteration* [122] is very efficient especially if the solver is implemented using sparse data structures. Therefore, instead of testing a single instance of an MDP, we were able to vary the vertical velocity penalty (reward) and generate multiple instances of our MDP CAS model to trace out system performance (SP) curves. SP curves are similar in nature to system operating characteristic (SOC) curves, [77, 78, 141] which generally involve plotting unnecessary alert against successful alert. Results for our MDP CAS is given in Table 3.3 and Figures 3-3, 3-4 and 3-5 show SP curves pertaining to our MDP model. In the SP curves, points close to the origin are more desirable as they represent low risk ratios and low velocity/acceleration values (less maneuvering), and our MDP model scores better than the other systems on the Velocity - Risk Ratio curve.

Graphs displaying velocity, acceleration, and probability of NMAC (PNMAC) values from 15,000 encounters using nominal flight strategy plotted against values from same encounters using our MDP collision avoidance logic are shown in Figures 3-

Table 3.3: Risk ratios for MDP collision avoidance system (perfect sensor).

Reward	Ratio	Velocity	Acceleration
−0.10	0.000692	14.174462	2.121009
−0.50	0.000980	7.721526	1.684897
−0.75	0.001428	5.505732	1.745723
−1.00	0.003075	4.970565	1.591075
−1.25	0.022785	4.133050	1.566663
−1.50	0.024709	3.820564	1.286228
−2.00	0.036734	3.125315	0.931763
−5.00	0.063469	2.159921	0.691902
−10.00	0.170806	1.460390	0.539181
−20.00	0.257840	1.059476	0.241147
−30.00	0.431986	0.973162	0.212496

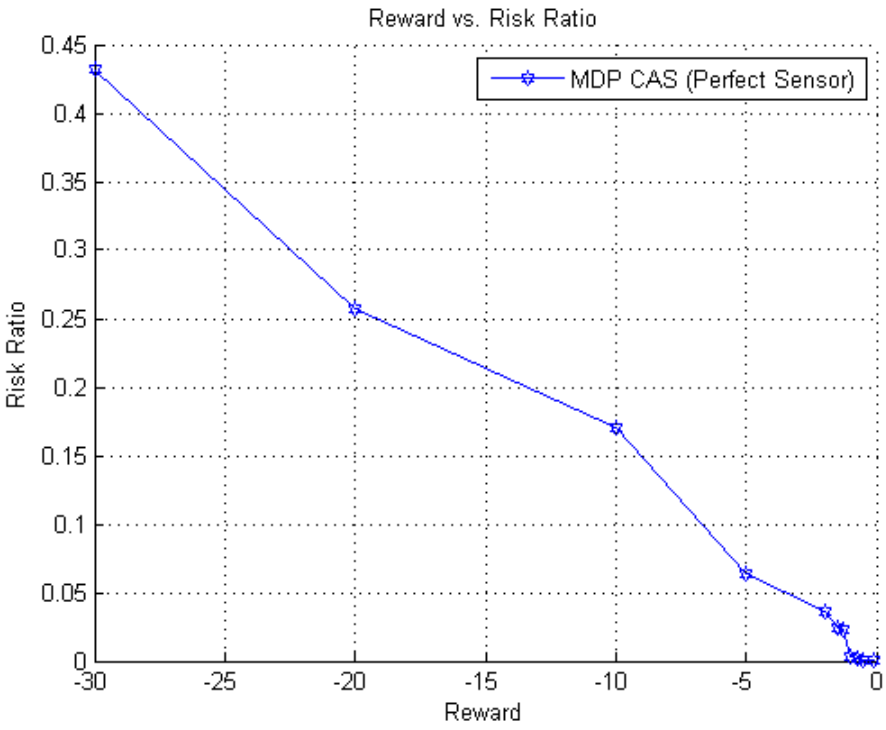


Figure 3-3: Reward vs. Risk ratio.

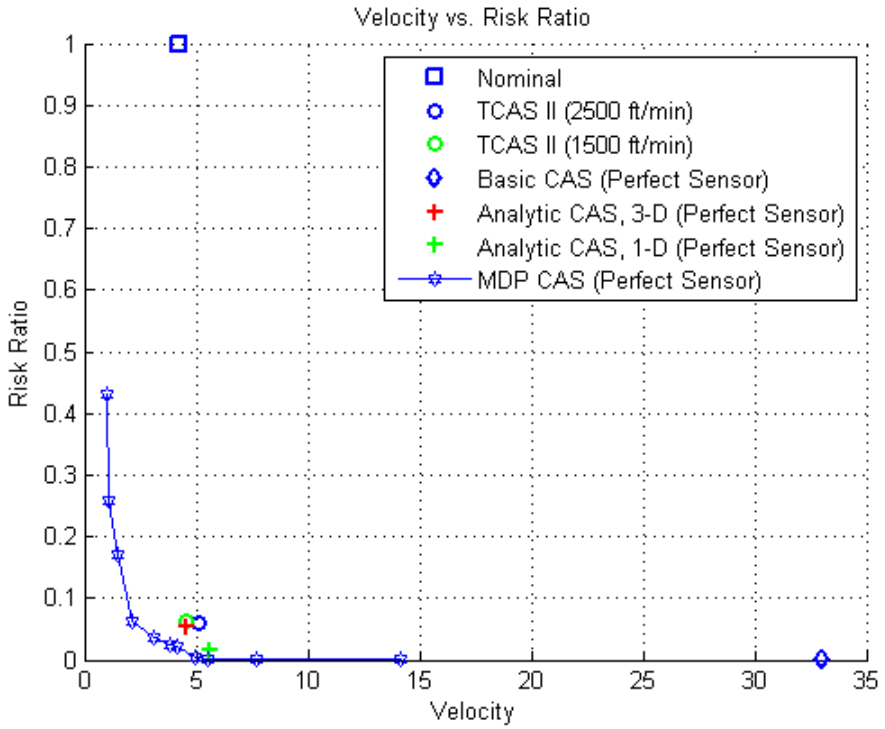


Figure 3-4: Velocity vs. Risk ratio.

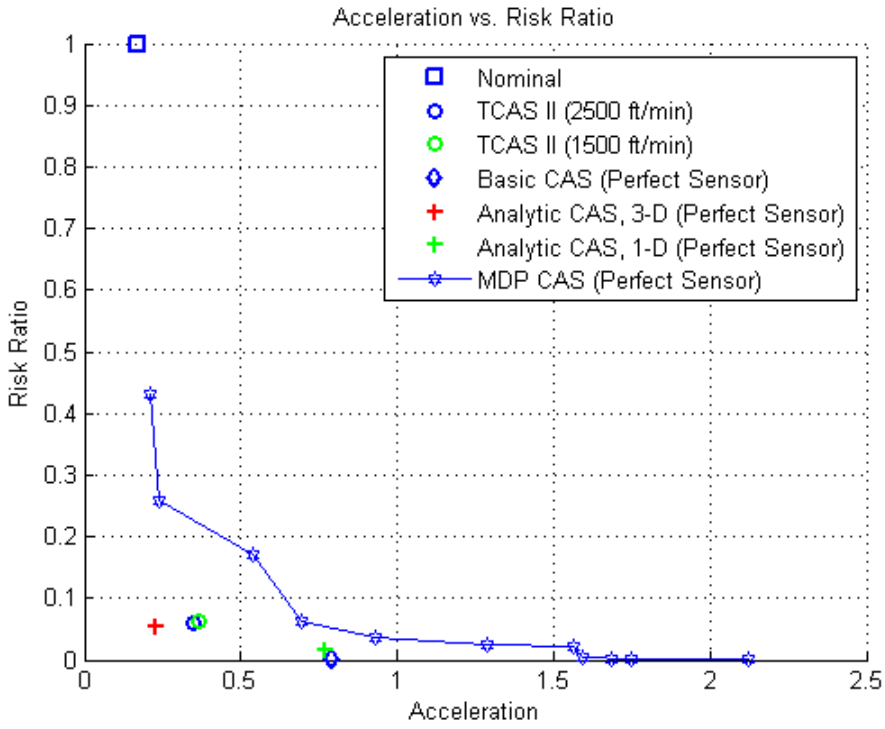


Figure 3-5: Acceleration vs. Risk ratio.

6, 3-7, and 3-8, respectively. In these graphs, scoring below the (red) diagonal are desirable as it indicates that an aircraft equipped with our collision avoidance system performs better (in terms of lower risk ratio or less maneuvering) than an aircraft that just follows the scripted maneuver for that particular encounter scenario. Note that our reward model is constructed to optimize velocities, therefore acceleration plots are not significant for our experiments in general, but presented as a reference. Also, in the PNMAC comparison, there are a few encounters where the MDP PNMAC is higher than Nominal PNMAC (points above the diagonal), which means that the collision avoidance system actually increases the risk of collision. At first, this might seem strange, but it can happen with certain encounter geometries as follows:

- Usually in such encounter scenarios, the intruder aircraft performs a dangerous altitude crossing maneuver with high speeds that, by chance, ends up with a large total miss distance (indicating a small risk of collision) at the closest point of approach when ownship follows the scripted maneuver.

- The collision avoidance system works as usual by minimizing expected costs, and it computes the maneuvers that best avoid an intruder whose intentions are modeled by a random walk process. However, application of the computed maneuvers is not enough to have a total miss distance that is larger than the one obtained by just following the scripted maneuver (therefore, the risk of collision is higher).

- The simulation framework uses total miss distance at the time of closest approach when evaluating PNMAC values, hence, due to the specific encounter geometry a nominal flight might actually score better than a perfectly rational plan.

Since our reward model penalizes high vertical velocities, it is not surprising that we do not get low acceleration values as opposed to the optimization we get with velocities. In fact, the MDP CAS prefers using high acceleration values. The histogram in Figure 3-9 shows the total number of states an action is chosen as the *best*

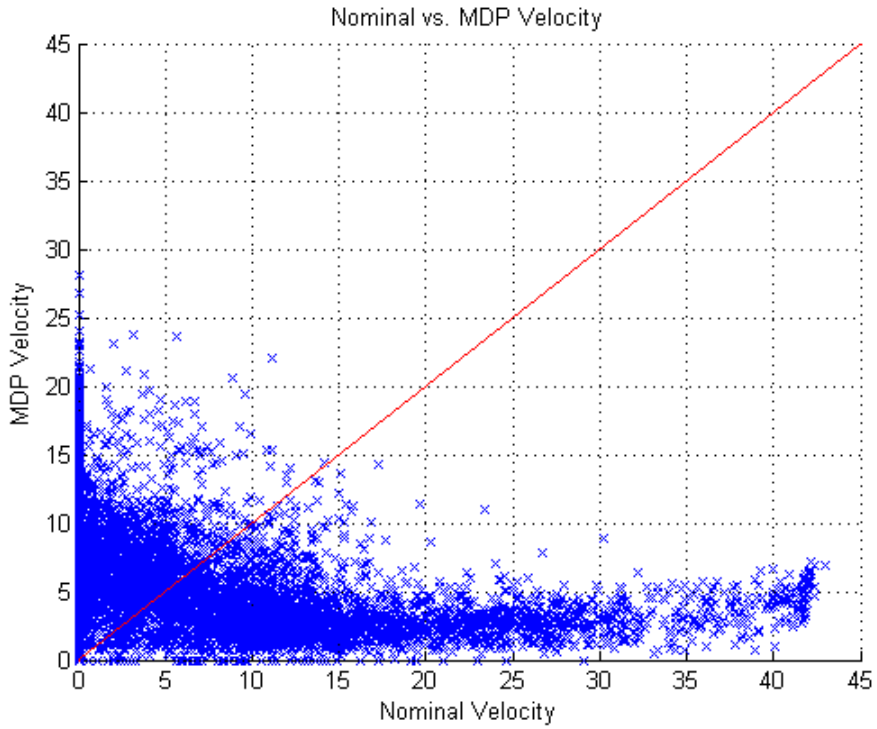


Figure 3-6: Nominal vs. MDP CAS Velocity.

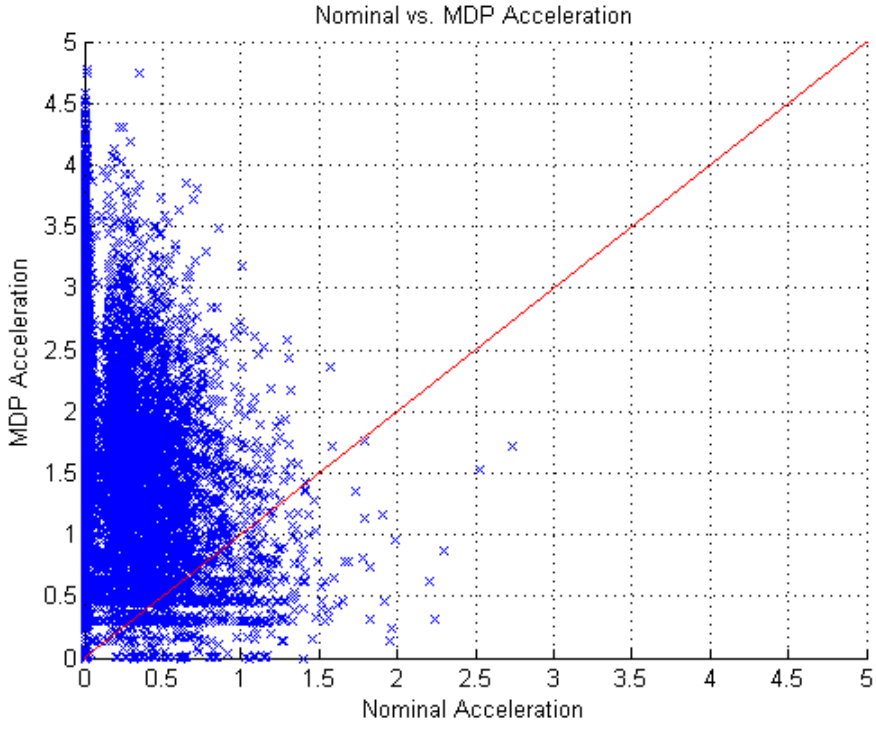


Figure 3-7: Nominal vs. MDP CAS Acceleration.

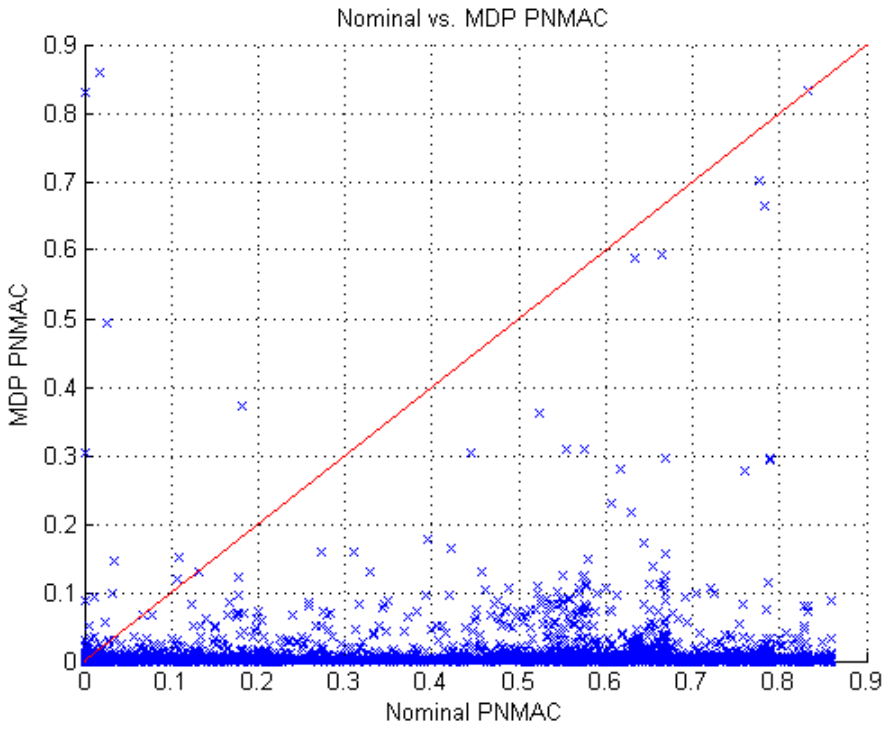


Figure 3-8: Nominal vs. MDP CAS PNMAC.

action by the MDP policy (this specific policy was generated with vertical velocity penalty = −2.00).

In conclusion, we can say that MDP CAS works well in the case of perfect sensing, and we can easily outperform baseline collision avoidance systems in terms of much lower risk ratios and velocities (without unnecessary maneuvering).

3.2 Noisy Sensing

Our second case is omnidirectional sensing with noise, and we will use the TCAS sensor model as our input source. If we were to make use of the *bearing* estimate produced by the TCAS sensor in locating the intruder aircraft in any 3-dimensional coordinate system, the error would be considerably big (especially with distant intruders). However, we chose to work with projections of intruder aircraft on RCS and hence we do not need the *bearing* estimate at all. It is possible to accurately locate the intruder on RCS using other TCAS readings. This gives us the following

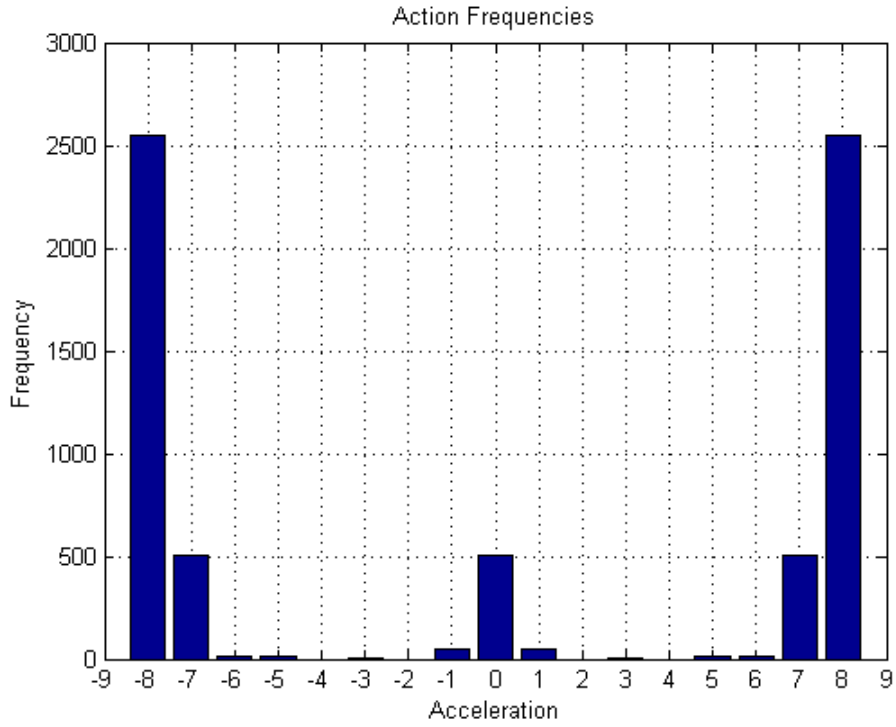


Figure 3-9: Frequencies of best actions in MDP policy (vertical velocity penalty = −2.00).

two options in designing a collision avoidance system that uses the TCAS sensor:

- We treat this as a perfect sensing problem and use the same MDP model developed in Section 3.1. To figure out the state, we either directly use the sensor reading neglecting the fact that it is noisy, or we use an estimator such as an alpha-beta tracker [137] or a Kalman Filtering based technique [46]. Since small observation noise does not affect action selection much in this specific problem, it is also feasible to formulate the problem as a Q-MDP [88] (the Q_{MDP} approximation is calculated by solving the POMDP as though it were fully observable, and then linearizing across Q-values to obtain the value at a belief), but we will demonstrate a solution with an external state estimator in this document.

- We define a discretized observation space Ω, and design an observation model for the TCAS sensor to augment the MDP model of Section 3.1, and turn the problem into POMDP planning.

In this section, we first present results for an MDP collision avoidance model using an alpha-beta tracker to estimate the state, and then we look at a POMDP model.

3.2.1 MDP Collision Avoidance System with State Estimator

The results for baseline collision avoidance systems with the TCAS sensor are shown in Table 3.4. Using a simple alpha-beta tracker for state estimation with $\alpha = \beta = 0.5$, we obtained the results in Table 3.5 with our MDP collision avoidance system for various vertical velocity penalty values. The SP curves are shown in Figures 3-10, 3-11, and 3-12.

Even though alpha-beta tracking is a very simple state estimation method, the results are satisfactory. Using Kalman filters, interacting multiple model methods [95], or nonlinear filters [129] may further improve the quality of state estimation, if desired.

Table 3.4: Risk ratios for baseline collision avoidance systems (TCAS sensor).

Algorithm	Ratio	Velocity	Acceleration
Basic CAS (TCAS sensor)	0.000010	32.909700	1.034700
Analytic CAS, 3-D (TCAS sensor)	0.080100	7.402750	1.096570
Analytic CAS, 1-D (TCAS sensor)	0.020500	19.557490	4.511640

Table 3.5: Risk ratios for MDP collision avoidance system (TCAS sensor).

Reward	Ratio	Velocity	Acceleration
−0.10	0.000916	13.225057	3.088738
−0.50	0.001717	7.431411	2.404085
−0.75	0.002428	5.101627	2.398184
−1.00	0.003337	4.494725	2.151822
−1.25	0.015149	3.857991	1.967395
−1.50	0.023313	3.657201	1.618871
−2.00	0.037456	2.906691	1.221383
−5.00	0.077662	2.033404	0.902261
−10.00	0.212924	1.448597	0.576285
−20.00	0.285638	1.055002	0.284902
−30.00	0.415815	0.993773	0.243202

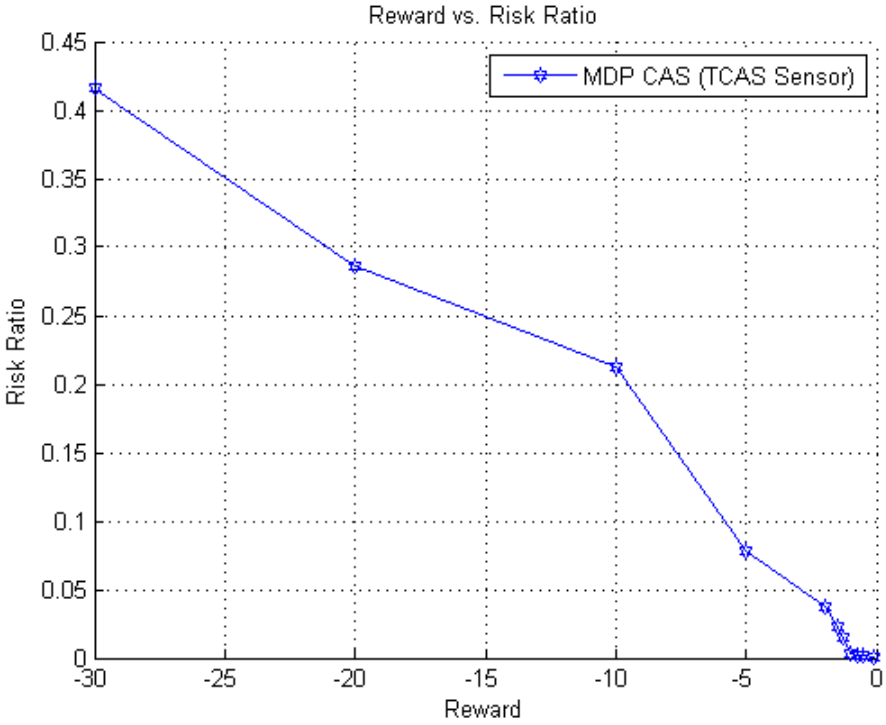


Figure 3-10: Reward vs. Risk ratio.

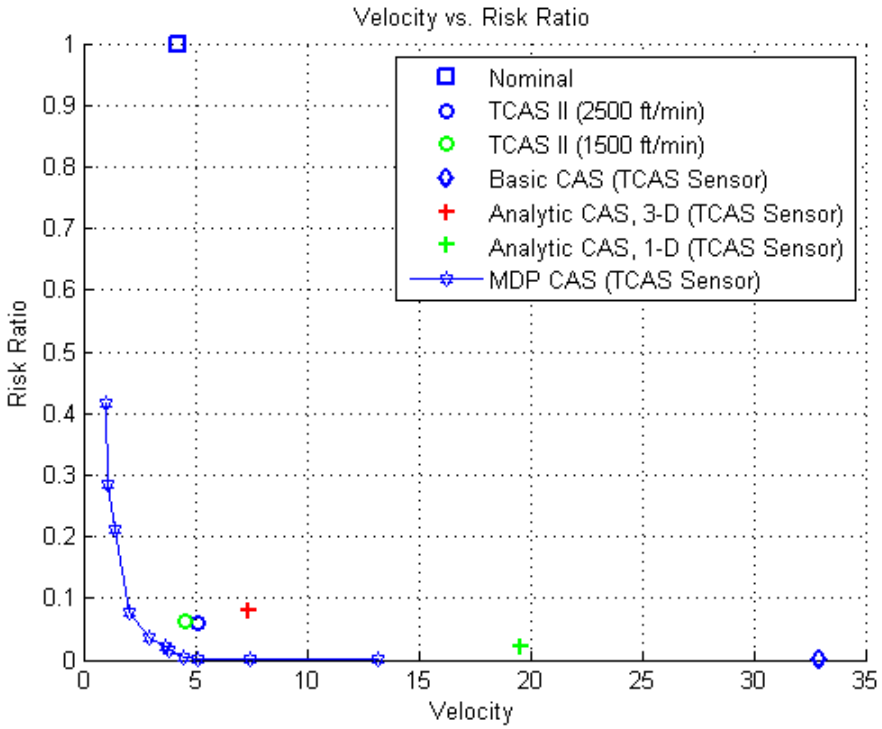


Figure 3-11: Velocity vs. Risk ratio.

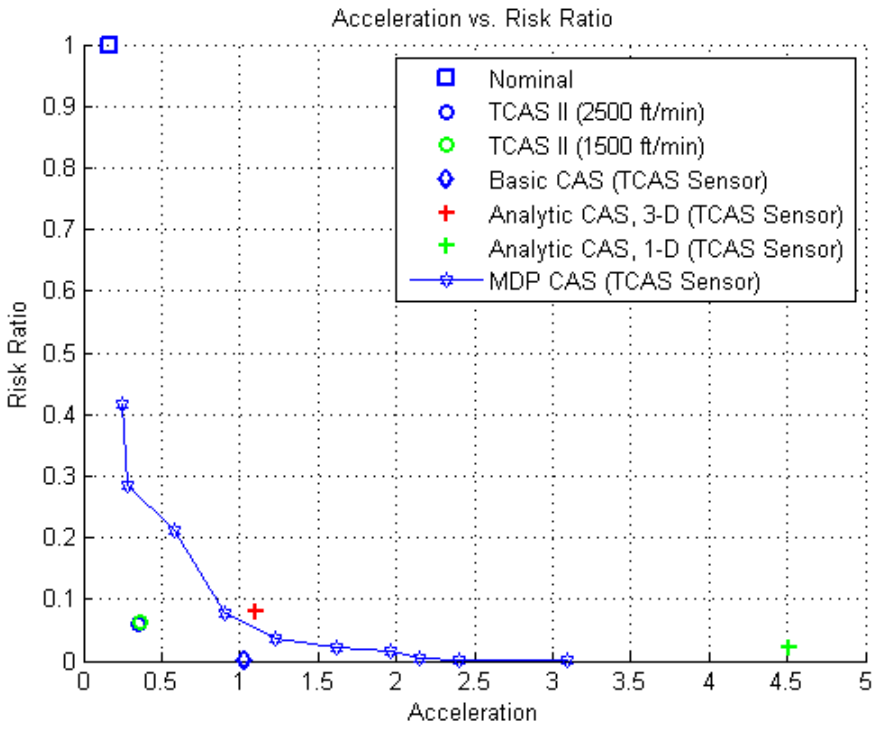


Figure 3-12: Acceleration vs. Risk ratio.

3.2.2 POMDP Collision Avoidance System with TCAS Sensor

The state space $\mathcal{S}$, and the state-transition model we built in Section 3.1 effectively capture important aspects of the encounter geometry and motion dynamics for both aircraft, respectively. Therefore, a POMDP collision avoidance model can be built on top of the MDP model of Section 3.1 by just adding an observation model. In this section, we will define the observation space and the observation model for the TCAS sensor, and we will also look at how we can slightly modify action space and reward model together to reduce the POMDP size and still obtain low risk ratios.

Observation Space

The discrete model of the observation space is constructed in a way similar to the discrete state space. There are two types of observational information: vertical velocity of ownship (V_Y^{Ownship} , which we assume is always completely and correctly

observed), and possible single detection of an intruder aircraft using a sensor system. Observations for the TCAS sensor are discretized into the same bins as the X and Y components of the state space. The model could easily be changed to provide observations at a higher or lower granularity. In addition, there is a special *noObs* observation for the case when no intruder is detected (due to either an empty sensing region or a false negative measurement).

Observation Model

The observation model of a POMDP specifies $\Pr(o \mid s, a)$, that is, the conditional probability of making each possible observation o, given that the actual state is s and the last action was a. All necessary information is encapsulated in s, so we will ignore dependence on a, and specify $\Pr(o \mid s)$ for all discrete o and s.

We assume that, at every step, the observation has two components: o_{ovy}, our measured vertical velocity, and o_d, the observed detection of the intruder, and that these are independent, so

$$\Pr(o_{ovy}, o_d \mid s) = \Pr(o_{ovy} \mid s) \Pr(o_d \mid s) .$$

The measurement of our vertical velocity is always correct, so $\Pr(o_{ovy} \mid s) = 1$ if o_{ovy} is equal to the V_Y^{Ownship} component of s, and 0 otherwise.

The observed detection is more complex due to false positive/negative measurements and measurement errors described in Section 2.3. We assume fixed probabilities for false positives p_{fp} and false negatives p_{fn}, and assume that if there is a false positive detection, it is generated with uniform probability over the space of values of o_d.

When s is a START or DONE state (the encounter has not yet begun or has terminated) or when $Y > maxRange$, that is, when the distance to the other aircraft is greater than the range of the sensor, then $\Pr(o_d = noObs \mid s) = 1 - p_{fp}$. That is, with high probability, the observation is *noObs*. We used a value of 5 nautical miles for *maxRange* for all sensors. For any other observation $\Pr(o_d = d \mid s, fp) = |O_d|^{-1}$; that

is, it is uniform over the space of possible actual detection observations.

Finally, if the intruder is within the modeled volume of the state space, there is some chance of not seeing the intruder: $\Pr(o_d = noObs \mid s) = p_{fn}$. Otherwise, with probability $1 - p_{fn}$, we make a detection d.

A precautionary *margin* is added to all four sides of the X-Y rectangle corresponding to the detection d. Then we consider all of the X-Y bins b_i that overlap the expanded detection bin, and the proportion of the expanded detection bin that overlaps b_i, called p_i. So,

$$\Pr(o_d = d \mid s) = (1 - p_{fp})p_i + p_{fp}|O_d|^{-1} \ ,$$

for any state in which the intruder is in X-Y bin b_i, for all bins b_i, and

$$\Pr(o_d = d \mid s) = p_{fp}|O_d|^{-1}$$

otherwise. For the TCAS sensor, we can define the *margin* in terms of standard TCAS sensor error parameters given in Table 2.2:

$$\begin{aligned} margin \ = \ & \text{Altitude quantization} + \\ & 3 \times \text{Range error standard deviation} + \\ & 3 \times \text{Altimetry error scale} \end{aligned}$$

Including full altitude quantization and 3 standard deviations worth of error in the *margin* gives us an unnecessarily conservative confidence region around the detection d which can, in fact, hinder intruder localization and render the observation model useless. The margin should be large enough so that it covers the region from which a noisy sensor reading may have originated, but it should be small enough to allow the POMDP to properly localize the intruder. Therefore, we used smaller margins in our experiments (half of altitude quantization and 0.5 standard deviations gave us reasonable risk ratios).

The observation model for the TCAS sensor is described by Algorithm 3 in Ap-

pendix A.

Modifications

POMDP solvers work with belief-states instead of exact states and branch on actions and observations, therefore their memory and time demands are typically much higher than MDP solvers, especially if we would like to compute policies with tight regret bounds. A POMDP model with the same state and action spaces as the MDP model of Section 3.1 takes days to just initialize and generate the first heuristic policy in the iterative improvement process. We describe below how the parametric design of our POMDP model gave us leverage to reduce the size without decreasing performance.

As depicted in Figure 3-9, the MDP collision avoidance logic mostly uses very high, very low or zero acceleration options available when picking an action. This is in accordance with our reward model. Based on this observation, we used a new and smaller action space with only three actions, $\mathcal{A} = \{-8, 0, 8\}$, which correspond to accelerating up/down with maximum magnitude or maintaining vertical velocity.

We also used a slightly different discretization for the state space: 7, 10, 4, 4, and 3 bins for X, Y, V_X^{Relative} , V_Y^{Intruder} , and V_Y^{Ownship} components of $\mathcal{S}$, respectively, and 3 START and 3 DONE states, which bring the number of states down to 3366.

Finally, based on some experimental results, we increased the size of the protected airspace around ownship to 200 ft vertical and 1000 ft horizontal separation.

These modifications let the SARSOP solver initialize in about an hour and generate acceptable policies (in terms of low risk ratios) in 3 to 5 hours.

3.2.3 Results

Tracing out SP curves for POMDP models is very time consuming, and essentially the longer the solver runs, the better the generated policies perform. Therefore, we present the single best result we obtained for our POMDP model (in terms of low risk ratio) using a vertical velocity penalty of −0.1 in Table 3.6. The POMDP collision avoidance logic for the TCAS sensor is about 20 times safer than TCAS Version 7

Table 3.6: Risk ratios for POMDP collision avoidance system (TCAS sensor).

Algorithm	Ratio	Velocity	Acceleration
POMDP CAS (TCAS sensor)	0.002770	14.133030	1.759190

currently used on manned aircraft. However, TCAS has a much lower mean vertical velocity magnitude, indicating that it maneuvers less frequently.

Although we use the same sensor model of the TCAS algorithm for our POMDP model and constrain the vertical rate magnitude to be within 2500 ft/min, the comparison is not entirely fair. TCAS was designed for pilot-in-the-loop control and assumes a delay between when the resolution advisory is issued and when the pilot responds. Although the POMDP algorithm has the advantage over the TCAS algorithm because it can maneuver instantaneously, the TCAS algorithm is permitted to make up to 0.35 g maneuvers whereas the POMDP was constrained to 0.25 g maneuvers. We use the standard model of pilot response to TCAS resolution advisories, which is a 0.25 g acceleration after a 5 s delay for the initial advisory and a 0.35 g acceleration after a 2.5 s delay for subsequent advisories [60]. Although a direct comparison between the POMDP model and TCAS algorithm cannot be made, we can be confident, at least, that the POMDP is performing well.

Considering the MDP model results and comparing both risk ratio and flight plan adherence, we conclude that, for the TCAS sensor, an MDP model is the right choice.

3.3 Limited Field-of-View Sensing

As our final case, we look at POMDP collision avoidance using radar and EO/IR sensors. Both of these sensors have noise, and are effective only within a limited sensing region. Most important complications caused by these two sensors are the following:

- Unlike the TCAS sensor which provides an accurate *altitude* reading in GCS, these sensors provide *elevation* estimates that we need to use when projecting

the intruder aircraft location on RCS. Using an angular measurement makes it difficult to localize distant intruders in RCS, so until the intruder is sufficiently close, the altitude estimate will not help the POMDP model much in choosing an evasive action.

- The sensing region is horizontally wide, but vertically, it is a very narrow band in front of ownship. Therefore, nearby aircraft can fly undetected most of the time (even when they are dangerously close). This also causes late detection of some ascending or descending intruders that suddenly enter the detection region, leaving very little space and time for an escape maneuver.

- During an escape maneuver, the sensor orientation (and hence the orientation of the detection region) changes as ownship accelerates (pitches) up or down. Most of the vertical maneuvers cause the intruder to disappear from (move outside of) the sensing region.

In terms of model implementation, there is very little work to do: We base our design on the POMDP model of Section 3.2.2 (using the same state and action spaces described in Section 3.2.2) with some minor adjustments that we describe below, we use the previously introduced special observation, *noObs*, whenever there is no detection (for example, when a state falls outside sensing region), and we just employ the POMDP solver to design effective strategies for dealing with the limitations of sensing. This also means that we are able to achieve one of our goals for this work; POMDP models allow us to easily and quickly design collision avoidance strategies for different sensor configurations.

3.3.1 POMDP Collision Avoidance System with Radar Sensor

For the radar sensor, we use the same observation model as the TCAS sensor with the following two modifications:

- An overly conservative *margin* can be defined in terms of standard error parameters from Table 2.2:

$$\begin{aligned} \textit{margin} \;=\; & 3 \times \text{Range error standard deviation} \;+ \\ & \text{Longest distance to bin edges} \times \\ & \tan(3 \times \text{Elevation error standard deviation}) \end{aligned}$$

- The pitch angle of ownship (and hence the orientation of the sensing region) can be computed using ownship's vertical and horizontal velocity values, but ownship horizontal velocity is currently not part of the state space. In our observation model implementation, we compute some very loose upper and lower bounds for pitch angle using the maximum and minimum velocities of our aircraft model from Table 2.1, and use them to figure out which X-Y boxes fall outside sensing region. We believe that performance could further be improved with a better POMDP model that could accurately predict the field-of-view of the sensor. However, addressing this issue requires an extension to the state space and increases the POMDP size considerably.

The observation model for the radar sensor is described by Algorithm 4 in Appendix A.

3.3.2 POMDP Collision Avoidance System with EO/IR Sensor

The EO/IR sensor reports the elevation angle of the intruder aircraft, therefore the projection of the intruder on RCS can be constrained to lie on a ray (with noise) rather than a point.

Detections from the EO/IR sensor are nominal angles that can be thought of as the centers of angular bins, which are not necessarily uniform. For each state s in which the intruder is located in the modeled X-Y space, we can compute a *nominal* elevation angle $d^*(s)$ to the intruder. We assume that the probability of observing a

Table 3.7: Risk ratios for baseline and POMDP collision avoidance systems (radar and EO/IR sensors).

Algorithm	Ratio	Velocity	Acceleration
Basic CAS (radar sensor)	0.050830	19.232450	2.409710
Basic CAS (EO/IR sensor)	0.047240	19.450350	2.308330
POMDP CAS (radar sensor)	0.063370	23.628310	1.261540
POMDP CAS (EO/IR sensor)	0.035100	28.610760	1.476910

detection angle d when the actual angle is d^* is proportional to a Gaussian density with mean at d^*; so,

$$\Pr(o_d = d \mid s) = (1 - p_{fp})\frac{1}{z}e^{(d-d^*(s))^2} + p_{fp}|O_d|^{-1} \quad ,$$

where

$$z = \sum_s e^{(d-d^*(s))^2}$$

is the normalization constant.

We also use the same pitch angle approximation of the radar sensor described in Section 3.3.1 to assign *noObs* to X-Y boxes that fall outside the sensing region.

The observation model for the EO/IR sensor is described by Algorithm 5 in Appendix A.

3.3.3 Results

Table 3.7 summarizes results for baseline and POMDP collision avoidance systems using radar and EO/IR sensors. The vertical velocity penalty was set to −0.1 for the POMDP models.

As expected, radar and EO/IR sensors have higher risk of collision than TCAS and perfect sensors since their performance is inherently limited by their field-of-view constraints.

There are also two important observations here that we would like to emphasize:

- On one side we have the radar sensor that provides an additional *range* reading

that allows (horizontal) localization of intruder aircraft in RCS, and on the other side we have the EO/IR sensor with a smaller error in *elevation* estimate which allows better vertical localization. Even though a simple comparison is not possible, by looking at the risk ratios we can conclude that accurate vertical localization is more important than accurate horizontal localization for collision avoidance systems that perform evasive maneuvers in the vertical dimension.

- A POMDP solver can in fact generate non-trivial (if not superior) collision avoidance strategies that can compete with hand-crafted ones. The EO/IR sensor, with its limited field-of-view and lack of horizontal localization ability, provides us a good example where the POMDP strategy scores a lower risk ratio than the Basic collision avoidance system using the same sensor. As an example of a non-trivial behavior, we observed that the POMDP strategy for the EO/IR sensor commands ownship to pitch up and down successively especially at the beginning of encounters, which would help to actively search for intruders that might be outside the sensing region and/or to better localize ones that are inside the sensing region. This is a sacrifice in terms of more maneuvering, but it results in low risk ratios that is in accordance with the reward model used. Even though a *policy* generated by a solver might not be easy to verify and validate, it can at least inspire hand-crafted techniques and/or serve as a baseline.

3.4 Discussion

In this section, we briefly discuss the limitations of our MDP/POMDP collision avoidance models and present a short assessment.

3.4.1 Model Limitations

Below are certain ways in which our models were limited and some suggestions about how the performance of the models could be improved.

Discretization

Our state space representation captures most of the features that are necessary in selecting an action to avoid collisions, but there is a loss of information when we go from two 13-dimensional aircraft state vectors to a 5-dimensional state space. One way to improve performance is to augment the state space with more features from the underlying true state space, and another way is to use a finer grained discretization, which involves adding more bins along each dimension, but we should also note that both of those approaches cause huge growth in the size of the state space and the time it takes to compute policies.

Parameter Values

Our models contain many parameters (most of them are externally configurable and some of them are internal to implementation) that have not been tuned to the encounter model. Many of the parameter values were chosen by experimentation. We believe that performance can be significantly improved by better matching the internal model used for decision making to the encounter model used for evaluation.

Missing State Information

There are certain features that may improve performance that are currently not part of our state space. One such feature is ownship *roll angle*. With limited field-of-view sensors, sometimes whether the intruder falls into the active angular range of the sensor or not depends on how much ownship is banking. In the current formulation, there is no way to estimate the current roll angle from a given state, therefore we cannot project the active angular range of sensors onto the projection plane to determine intruder detectability. We currently assume a fixed (0 degree) roll angle, and add some precautionary margins, but this affects the performance in one of two ways:

- If our roll angle is actually 0, we would be assigning positive probabilities to some undetectable bins that are inside the margins.

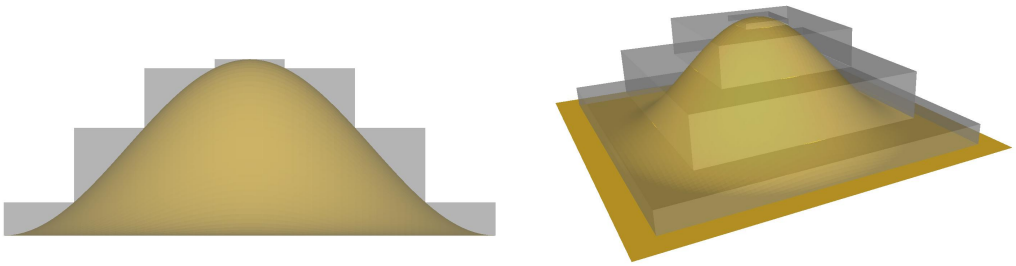


Figure 3-13: Gaussian distribution approximated by four flat distributions stacked on top of each other.

- If our roll angle is larger than the margins and the intruder is detectable as a result of this geometric configuration, we would be assigning zero probability for a case that is actually possible. Having one of those cases during policy execution might lead to a belief-state crash (a belief-state update resulting in an invalid belief-state with 0 probability assigned to all states).

Observation Models

Error models for most of the sensor measurements are Gaussian. In our implementations, we used a method to coarsely discretize a Gaussian distribution as shown in Figure 3-13 and applied it to 2-dimensional observation bins. We believe that a better Gaussian discretization scheme or an analytical solution would further improve results, as better observation models help localize the intruder aircraft in RCS with more precision, and that results in better action selection.

Estimation of Vertical Velocity of Intruder Aircraft

Evasive maneuvers are performed only in the vertical dimension, therefore it is important to estimate the vertical velocity of intruder aircraft as accurately as possible. Unfortunately, this requires a much finer discretization of the heights of the 2-D bins in the projection plane, which in turn increases the size of the state space.

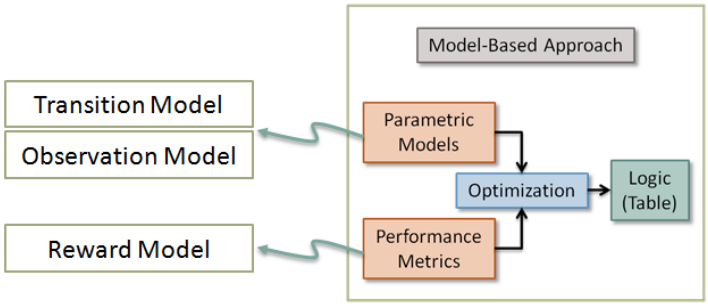


Figure 3-14: Model-based approach, MDP/POMDP based collision avoidance logic.

Estimation of Closure Rate

In order to keep the state space small and still be able to cover a very large projection plane, we used variable sized boxes (both in vertical and horizontal directions). We observed that putting narrow boxes close to the RCS origin and making the boxes wider as we move away from the origin works well for most of our purposes. However, a wide box also means that we will be getting the same observation repeatedly until the projection of the intruder falls into another box. These kinds of observation patterns affect both vertical velocity and relative horizontal velocity (closure rate) estimations, as successively getting the same observation creates the illusion of a stationary intruder, and suddenly getting a different observation results in velocity estimations that are higher than they really are. As in the vertical velocity case, a finer discretization is required to alleviate this problem.

3.4.2 Assessment

According to the results of our experiments we conclude that:

- The MDP/POMDP formulation is flexible enough to accommodate a variety of sensor modalities, intruder behavior, aircraft dynamics, and cost functions as shown in Figure 3-14.
- Complex policies produced by MDP/POMDP solvers can be implemented in real time. Both state estimation and policy execution are quite efficient for the state spaces considered.

- Current state-of-the-art solvers can generate useful collision avoidance behavior using a simplified representation of the aircraft dynamics.

- Improvements in the problem formulation may further improve performance. In particular, we have limited our formulation to representing motion in two (relative) dimensions. Moving to full three-dimensional motion would yield more effective evasive maneuvers, but in a discretized formulation this will take the size of the state space beyond the range of existing solvers. Therefore, better results are likely to be achieved with the investigation of alternative representations [72] and with the improvement of current solvers or the development of new types of solvers that suffer less from exponential explosion.

- During the course of this study we developed three software tools [131]: The first one was built to automate generation of parametric POMDP descriptions. The automated POMDP generation process is described in Appendix B. The second tool converts a POMDP specification into what we call a Processed POMDP (PPOMDP) that allows for very fast belief-state updates. This conversion is described in Appendix C. The third tool was built in order to better understand, analyze and debug POMDP policies generated by the solvers. It is capable of displaying and graphically visualizing encounter data, and it is described in Appendix D. Development of more sophisticated tools is likely to be necessary, especially for the verification of MDP/POMDP policies before they can be deployed on real aircraft.

Chapter 4

Path-Modification Based Collision Avoidance Models

In the previous sections we have shown how to build successful collision avoidance policies offline using the MDP/POMDP framework. We have also demonstrated the effectiveness of the model-based approach where we provided carefully designed encounter models and performance metrics, and we employed the solvers to perform the optimization.

There are basically two important directions in which we would like to extend the MDP/POMDP based collision avoidance models in order to have practically deployable and better performing collision avoidance systems:

- Planning Escape Maneuvers in 3-D: We would like to be able to plan 3-dimensional escape maneuvers by using both of the vertical and horizontal planes and also possibly varying turn rate within ownship's performance limits for more effective avoidance maneuvers. In principle, the POMDP framework does not limit us in adding more dimensions to the state, action and observation spaces. Therefore, it is possible and straightforward to extend the POMDP models to handle 3-D maneuvers, but practically this is not possible due to the limitations of the effectiveness of current solvers on POMDPs with huge spaces.

- Multi-Aircraft Planning: We also would like to handle multiple intruder aircraft

simultaneously in a *global* sense rather than employing *pairwise* resolutions. In the pairwise approach, multiple potential conflicts are examined sequentially in pairs and if one solution induces a new conflict, the original solution may be modified further until a conflict-free solution is reached. On the other hand, a global solution considers more than one aircraft at a time and while more complex, it may be more robust (Kuchar and Yang describe global and pairwise solutions in detail in their review [79]). With a reasoning similar to above, we can say that it is also difficult to automatically generate policies for multi-aircraft encounters on the POMDP framework using various solvers, because it is not easy to both represent the problem in sufficient detail and also have a small size state space that could be managed by the solvers.

With the current solution algorithms and implementations, it seems unlikely that we can solve MDPs and POMDPs with state spaces that are big enough to contain all the necessary information to achieve above goals, in reasonable time. Fortunately, the collision avoidance problem has some nice features that give us leverage to solve some parts of the problem analytically and/or geometrically rather than having to discretize along all dimensions, and still preserve the main principles of Markovian solution techniques. Particularly, aircraft transition models are quadratic in acceleration, and we can solve directly for certain important functionalities such as computing future aircraft states given a set of control commands. We also would like to note that analytical and geometrical solutions are usually verified more easily than the logic automatically derived by some model-based approaches using complicated solvers (such as the POMDP policies in the form of a set of α-vectors and associated action indices), and they are also likely to be used extensively in future collision avoidance systems [27].

In this chapter, we will analyze a dynamic replanning approach that takes advantage of the analytically solvable components of the collision avoidance problem to make it possible to accomodate above goals. Implementationwise, we will continue with the model-based approach, but instead of off-the-shelf solvers, we will be building and exposing the inner workings of our new solvers this time. The algorithms we

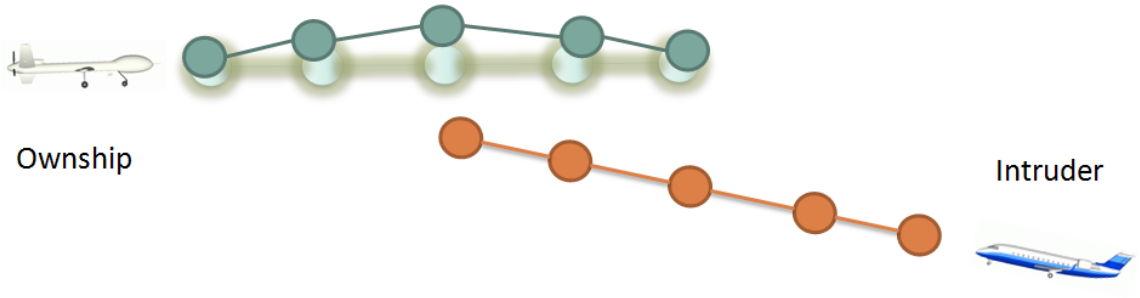


Figure 4-1: Path-modification approach. We will start with initial flight trajectories (pale green) and improve them (green) to reduce expected cost.

will build will still be generic and fully-parametrized to work with different types of aircraft dynamics and sensor modalities. Instead of the MDP/POMDP way of enumerating all states, actions and observations and trying to solve offline for a policy covering all possible situations, we will take an online approach and build planners that start with simple flight trajectories and then improve them in continuous space to avoid collisions as depicted in Figure 4-1.

Below is a quick recap of the key components of the collision avoidance problem domain that our approach will adhere to:

- The underlying state, action and observation spaces are all continuous and are very high dimensional. Therefore, fine discretizations of those spaces would result in sizes that are beyond the limits of practical solutions with existing MDP/POMDP solvers, and coarser discretizations introduce additional uncertainty and other limitations in the solutions. Furthermore, the physics and the geometry governing the evolution of the system can usually be described analytically.

- There is uncertainty in detecting the position and motion of oncoming aircraft due to sensor imperfections. The intentions of intruder aircraft are also uncertain. Therefore, the problem is a case of optimization using probabilistic models of motion and intention under uncertainty.

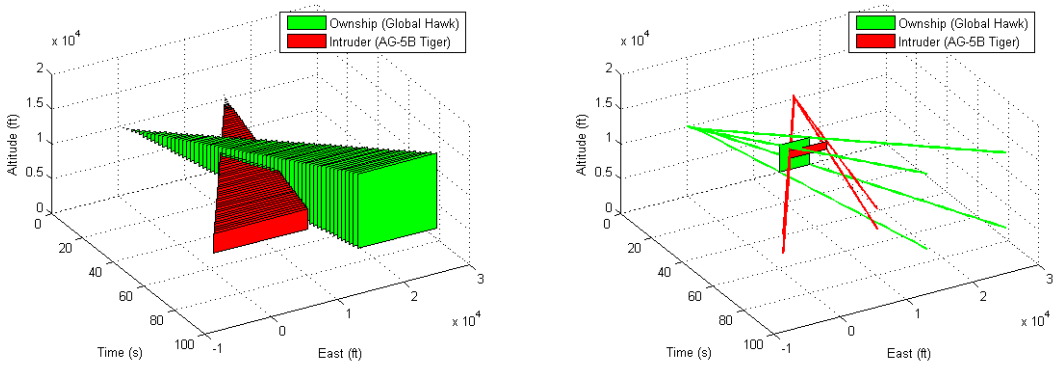
- We can control ownship perfectly, but we need to observe the fact that ownship

motion (or aircraft motion in general) is nonholonomic. Furthermore, since the type of aircraft that we consider in this research usually have strict performance limits including a minimum airspeed that should always be observed, it is not possible to use tricks such as "frequently stopping and turning to orient ownship in the target direction" to imitate holonomic motion. The type of motion planning that we need to have will take into account both dynamic and kinematic constraints, which is known as kinodynamic planning [33].

As will be discussed in the following sections, the nonholonomicity of aircraft motion dictates that the collision avoidance maneuvers that we plan for ownship should only include regions that are attainable by ownship in both space and time. On the other hand, the nonholonomic nature of intruder aircraft can be put to our advantage knowing that the regions that are attainable by intruder in space and in time are also bounded by its performance limits. Before we continue with the next section, we provide below the definition for *Space-Time Attainable Regions*, a term that we will use in the rest of this document.

Space-Time Attainable Regions

All types of feasible motion planning tasks for nonholonomic agents (for example, feedback control synthesis [130] and planning with bounded agent dynamics [34]) require that we only consider reachable configurations of the state space. We will call the set of points in space and time that an aircraft can occupy within its performance limits as Space-Time Attainable Regions, or STAR. Figure 4-2 shows the STAR representation of a 90 seconds long encounter in 2 space dimensions (2-STAR). In the figure, we only consider aircraft motion in *east* and *altitude* axes of GCS. The two aircraft involved in the encounter are a Global Hawk and an American General AG-5B Tiger, and the attainable regions shown as green and red patches at every 1.5 second intervals are obtained by starting from sample initial aircraft states and then varying the horizontal and vertical accelerations of both aircraft within their performance limits.



2-STAR of ownship and intruder

A slice of 2-STAR, $t \approx 35$ s

Figure 4-2: 2-STAR representation of a sample encounter.

The STAR representation gives us all possible locations that could be occupied by an aircraft, but usually aircraft motion is smooth, and we might expect to see the intruder aircraft in a more restricted region inside STAR with higher probability. Figure 4-3 shows an example where the intruder's path is approximated as a widening Gaussian over time.

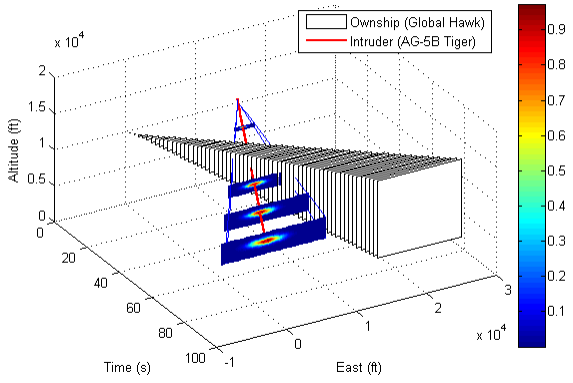


Figure 4-3: Probabilistic 2-STAR representation. A few slices of intruder's 2-STAR is shown with color-coded probabilities of actual location.

Relating to the MDP/POMDP framework, we can think of STAR as a compact representation of part of the state space that is explored step-by-step by the state-transition model of an MDP/POMDP solver during policy computation. In the following sections, we will take advantage of the compactness of this representation in order to come up with non-discretized collision avoidance solutions.

The aircraft collision avoidance problem also allows turning maneuvers in addition to the vertical and horizontal motion, so we will need to work with 3 space dimensions (3-STAR) in general, in order to compute the most effective escape maneuvers.

The STAR representation helps us generate compact geometric depictions of all possible maneuvers at once. Hence, coarsly, the goal of our collision avodance algorithms that will be designed in the following sections can be summarized as computing trajectories for ownship that reside within 3-STAR of ownship and that completely avoid 3-STARs of intruders, if possible, or that stay outside high probability regions, if avoiding completely is not possible. We will also balance avoidance maneuvers with the second goal of achieving low vertical velocities, i.e. level flight. Therefore, it is not enough to just quickly search the *edges* of ownship 3-STAR to come up with an optimal trajectory.

4.1 Path Modification

We begin describing our new approach by giving a simple and informal example. Please note that in the rest of this document we use the terms "trajectory" and "path" interchangeably.

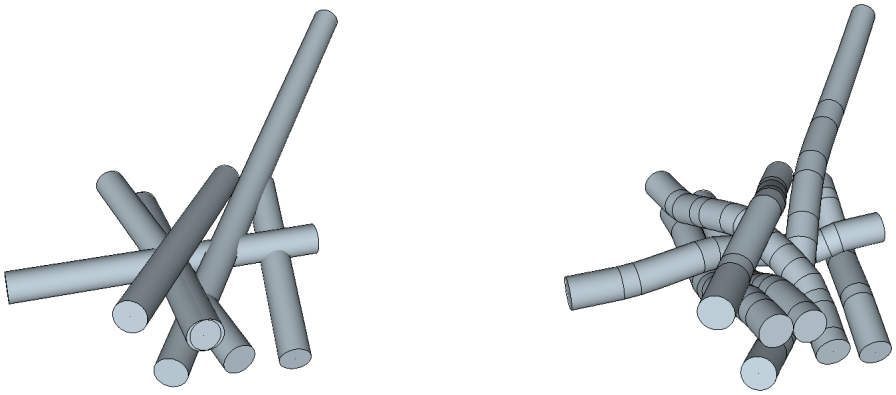
Let us assume that we would like to have a protected airspace in the shape of a sphere with radius r_{pa} around all aircraft involved in an encounter. In that case, we can say that our collision avoidance system is successful if the distance between any two points on the trajectories of any two aircraft is at least r_{pa} for the whole duration of the encounter. Therefore, instead of trying to optimize velocity and/or acceleration commands like the MDP/POMDP planners, let us work with aircraft positions and represent the intended or estimated paths of all aircraft as a sequence of *waypoints* first. Then we put enough separation between these paths using a method inspired by observing what happens when we cook spaghetti:

- In the spaghetti analogy, our paths are imaginary lines passing through the centers of spaghetti. The protected airspace around paths are represented by the

thickness of spaghetti. At the beginning, each spaghetti is very thin, therefore there is not enough separation between paths.

- As cooking progresses, spaghetti get thicker and push away their neighbors until they all reach their maximum radius, r_s. When cooking terminates, each path will have enough separation around it provided that $r_s \geq r_{pa}/2$.

In a implementation of the described idea, we iterate over all pairs of paths and all waypoints along each path, and incrementally add more separation between them until all paths are separated from each other by at least r_{pa}. In Figure 4-4 we present an example encounter scenario with 7 paths. Around each path is a protected airspace of radius $r_{pa}/2$. After the iterations are over, paths become modified such that there are no intersections between protected airspaces.



Before modification After modification

Figure 4-4: Demonstration of path modification. The trajectory of each aircraft is represented as a sequence of waypoints (not visible). The tubes extruded along the waypoints show the desired protected airspace around trajectories. In the figure on the left, most tubes intersect with other tubes, indicating protected airspace violations (in fact, two tubes at the center are almost coaxial). The figure on the right shows the modified trajectories; no two tubes have any intersecting regions anymore.

This simple method of looping over pairs of trajectories and iteratively separating them from each other is very flexible and it can in fact be very easily extended to allow for the following capabilities:

- In the given example, we assume that there is no positional *uncertainty* about the flight plan of any of the aircraft and we know exactly where to place way-

points. Even though this scenario is possible with the use of next generation sensors that broadcast intent information as a set of waypoints, it is not realistic for the purpose of the research described in this document. We need to be able to handle uncertainty in both the sensory observations as well as the intent of the intruders. With the path-modification method, it is possible to start by putting a large-enough protected airspace around the first waypoint of a trajectory to account for the observation uncertainty, and then placing enlarging protected airspaces around subsequent waypoints along the estimated trajectory, or in other words building a form of 3-STAR for the intruder, to account for intent/transition uncertainty (which makes the trajectory look more like a cone rather than a tube). An example is shown in Figure 4-5.

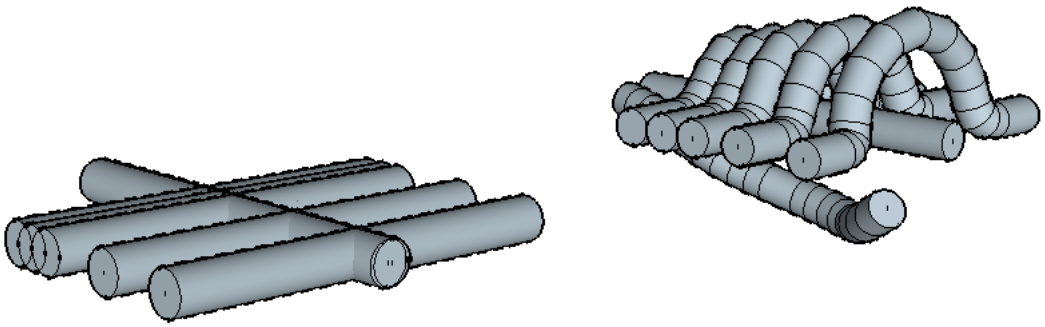


Figure 4-5: Path modification with uncertainty. On the left, we have 7 intersecting tubes (2 of them are almost coaxial) representing non-optimized flight trajectories. On the right, one trajectory is assigned increasing uncertainty along its waypoints and path-modification method is applied, resulting in increased separation between trajectories. All trajectories, including the one with uncertainty, are drawn as tubes graphically, but the conic form of the uncertainty is visible as additional space around the trajectory with the uncertainty in the middle.

- It is also possible to assign numerical *priorities* to each trajectory and update the path separation algorithm such that high priority trajectories are modified less (waypoints that belong to high priority trajectories get pushed away less than waypoints from lower priority trajectories at each iteration) during the process. This helps aircraft with important missions and/or with restricted performance limits deviate less from their planned trajectory when they encounter

other aircraft serving missions of lesser importance and/or having higher maneuvering capabilities.

- We can also take the priority approach further and completely *freeze* one or more trajectories making them non-modifiable during the separation process. A frozen path might represent aircraft on extremely important missions or aircraft that are oblivious to other aircraft. A very important point to note here is that a non-frozen trajectory might geometrically be trapped between frozen trajectories initially, therefore allowing some paths to be frozen might give rise to local minima problems during the application of the path-modification technique (the original problem in which all paths are modifiable does not suffer from local minima; it might take a long time to separate all paths, but similar to the spaghetti analogy, there will eventually be enough separation between all paths).

- Similar to assigning priorities to each trajectory, we can impose different *smoothness* constraints for each trajectory, too. A slight modification to the basic separation iteration enables us to end up with less/more curvy trajectories: Whenever a waypoint is moved in space, the same displacement vector with gradually decreasing magnitude is also applied to a few waypoints that precede and succeed it. The number of additional waypoints and their displacement amounts depend on how smooth we want the resulting trajectory to be. As an example use for this capability, we might want to plan smoother and hence more comfortable trajectories for passenger-carrying aircraft, whereas it might be okay to have jaggier trajectories for UAVs.

- Just like the way the container limits the displacement of spaghetti during cooking, it is possible to *constrain* the planning within a bounded region. For example, we can modify the separation iteration such that no waypoints move below/above given boundaries, ensuring flight within certain altitudes. In fact, any kind and number of virtual "walls" can be set up individually for each trajectory.

- Also, additional steps can be taken to post-process the planned trajectories to smooth them more or to wrap them tighter around each other [111].

4.1.1 Formulation

In this section, we present formal definitions of the necessary data structures and functionalities that we will use in constructing our path-modification based collision avoidance algorithms.

Data Structures

Our algorithms will use the following data structures:

- **Observation**: An observation ϕ is basically a raw sensor reading. The set of all possible observations for a given sensor is represented by Φ.

- **State Estimate**: A state estimate e contains the estimated values (with uncertainty) for a subset of the aircraft state vector for an intruder aircraft. The number of components in a state estimate depends on the collision avoidance algorithm, but the most important components are position and velocity estimates. The set of all possible state estimates is represented by $\mathcal{E}$.

- **Waypoint**: A waypoint w is used as the building block of both ownship and intruder trajectories, and very coarsly, it represents the location, speed and other important information about an aircraft at a specific point in time. Similar to a state estimate, waypoints for intruder aircraft will usually contain estimated values for a subset of the aircraft state vector. Waypoints for ownship however will include other components as well, such as the aircraft control command vector to be applied at that point in time. From an implementation point of view, waypoints might contain other information such as the position of a waypoint in an ordered set and/or links to the previous and the next waypoints. The set of all waypoints is represented by $\mathcal{W}$.

- **Trajectory**: A trajectory t is an ordered sequence of waypoints. The set of all trajectories is represented by $\mathcal{T}$, and $\mathcal{T} = \wp(\mathcal{W})$, where $\wp$ denotes the power set operator. The number of waypoints in a trajectory t is written $|t|$. The scripted (nominal) flight path for an aircraft can also be represented as a trajectory and is written t^s. We will use t^* to denote the optimal trajectory.

Cost Measurement

In order to optimize ownship trajectories against estimated trajectories of intruder aircraft, we need to be able to quantitatively measure the cost associated with a trajectory. For that purpose we will make use of the following guidelines when assigning costs to various aspects of the collision avoidance task in a quantitative manner:

- **Maneuvering Cost**: Within the MDP/POMDP framework, we penalized high vertical velocities when computing the maneuvering cost. In the rest of this research, we will stick with the same approach, and aim for level flight in the absence of nearby intruders. However, it might also be desired to penalize vertical acceleration rather than vertical velocity, for example in the case of UAVs, where high velocities create no discomfort as there are no humans onboard, but high accelerations might be less desirable as they require higher performances from the engines. The general form of measuring maneuvering cost can be described by a function $M : \mathcal{W} \times \mathcal{W} \to \mathbb{R}$ that takes two successive waypoints from the same trajectory and outputs a quantitative measure of the required maneuvering to get from the first waypoint to the second one. Using this function, we define the function $\text{Cost}_M : \mathcal{T} \to \mathbb{R}$ that measures the average maneuvering for a given trajectory, as follows:

$$\text{Cost}_M(t) = \frac{1}{|t| - 1} \sum_{k<|t|} M(w_k, w_{k+1}) \ , \quad w \in t \, .$$

- **Deviation Cost**: Similar to the MDP/POMDP models, we will in general assume that the deviation that needs to be penalized is the positional displacement from the nominal flight plan, which we will assume a level flight, but we

provide the general form as a function $D : \mathcal{W} \times \mathcal{W} \to \mathbb{R}$ that takes two waypoints (one from the nominal trajectory and the other from the actual trajectory, both representing state at the same point in time) and measures the deviation between the two in some quantitative terms. Using this function, we define $\text{Cost}_D : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$, which measures the deviation of a trajectory t from the scripted flight path t^s, as follows (assuming that waypoints are densely placed so that we can ignore the paths between waypoints):

$$\text{Cost}_D(t, t^s) = \frac{1}{|t|} \sum_{k \leq |t|} D(w_k, w_k^s) \ , \quad w \in t \text{ and } w^s \in t^s \, .$$

- **Collision Cost**: Let $\mathcal{U} = \wp(\mathcal{T})$. The function $\text{PNMAC} : \mathcal{W} \times \mathcal{U} \to \mathbb{R}$ takes a waypoint w that belongs to ownship trajectory (that is being tested as a possible escape maneuver) and a set of intruder aircraft trajectories, $u \in \mathcal{U}$, and computes the probability of collision for w. Using this function, we define $\text{Cost}_C : \mathcal{T} \times \mathcal{U} \to \mathbb{R}$, which assigns a cost to a trajectory t (in terms of either or both of average and maximum collision probabilities of waypoints along that trajectory) given a set of intruder aircraft trajectories u as follows (a_1 and a_2 are nonnegative constants):

$$\text{Cost}_C(t, u) = \frac{a_1}{|t|} \sum_{k \leq |t|} \text{PNMAC}(w_k, u) + a_2 \max_{k \leq |t|} \text{PNMAC}(w_k, u) \ , \quad w \in t \, .$$

  Note that, similar to the deviation cost, we assume that the waypoints are densely placed and there are no crossings of paths between waypoints of ownship and intruder aircraft.

Main Functions

Implementation of path-modification based collision avoidance systems will use the main functionalities that are described below:

- **State Estimation for Intruder Aircraft**: A function $F : \wp(\Phi) \to \mathcal{E}$ estimates the state of the intruder aircraft based on a set of sensor readings. The function

F can be designed to be as simple as accepting the current sensor reading as the most likely state and computing a region of positional uncertainty around it based on sensor specifications, or it might be more complicated to employ an alpha-beta tracker or a Kalman-Filter based approach to make use of past observations in estimating the intruder state, as well.

- **Trajectory Estimation for Intruder Aircraft**: Once we have a state estimate e for an intruder aircraft, a function $G : \mathcal{E} \to \mathcal{T}$ generates an estimated trajectory t for that intruder. The function G might take other implementation-dependent parameters such as the desired number of waypoints and the desired length of the generated trajectory in time.

- **Trajectory Computation for Ownship**: This step is the core functionality in planning the escape maneuvers using path-modification method. After we generate a set of estimated trajectories $u \in \mathcal{U}$ for the intruder aircraft using function G, a function $H : \mathcal{T} \times \mathcal{U} \to \mathcal{T}$ takes u and the ownship scripted flight path t^s, and computes a trajectory t for ownship by minimizing a cost function $\text{Cost} : \mathcal{T} \times \mathcal{T} \times \mathcal{U} \to \mathbb{R}$ which is defined as:

$$\text{Cost}(t, t^s, u) = c_1 \text{Cost}_M(t) + c_2 \text{Cost}_D(t, t^s) + c_3 \text{Cost}_C(t, u) \ ,$$

where c_1, c_2 and c_3 are nonnegative constants. The optimal trajectory t^* is defined as the trajectory with the minimum cost.

In our implementations, the function H will first create one or more initial trajectories including t^s as possible candidates, and then modify them to reduce costs (which is done by "bending, twisting, stretching, and/or shrinking" them to ensure safety from possible hazardous encounters with intruder aircraft and balancing with small vertical velocity values and as little deviation from a level flight as possible, while observing ownship performance limits) until a "good enough" trajectory is obtained. We can define "good enough" quantitatively

using some nonnegative constant c_{ge} as follows:

$$\text{Cost}(t, t^s, u) - \text{Cost}(t^*, t^s, u) \leq c_{ge}\,.$$

Note that in some special cases it might be possible to search some intuitive trajectories such as the extreme ones that can be achieved at the limits of ownship performance, or to use other heuristics in order to determine the optimal path t^*, but in most other encounter scenarios it might not be possible to come up with an optimal path, at all (especially when there are multiple intruder aircraft). In such cases, we can just compare the cost against a constant threshold for the "good enough" test.

Modification Techniques

A trajectory t consists of $n = |t|$ waypoints, w_1 to w_n. Note that we have neither specified nor restricted the full contents of a waypoint as it may differ from application to application. We just note that a waypoint contains enough components to describe the state of an aircraft at a specific point in time. In the simple path-modification example with 7 aircraft presented before, each waypoint was implemented to contain just the *east*, *north*, and *altitude* coordinates of an aircraft. We even omitted *time* component for the sake of simplicity. Therefore, each trajectory in the given example can be thought of as a $3n$-vector in *position* space. In that example, we implemented an ad hoc iterative process that optimizes each trajectory against all others. The process was for demonstration purposes only and thus it did not have enough complexity to verify that the resulting trajectories lie within 3-STARs of each aircraft.

Before describing *how* to modify trajectories, we first present below a brief discussion of *what* we will be modifying:

- We will work with time-stamped waypoints, i.e., all of our waypoints will contain a *time* component. The modifications will target separating only waypoints with the same time stamps. This allows for an intruder and an ownship waypoints to exist at the same position at different times. Working without time and creating

air tubes that are safe to fly anytime might be possible as demonstrated by the simple example, but its computational demands are high. Also note that when we consider noisy sensors and quickly growing uncertainties in the estimated trajectories of intruder aircraft, it might not be possible to come up with safe trajectories without taking time into account in our computations.

- We are interested in trajectories that are safely separated in *position-time*, but our algorithms do not actually have to work in *position* space. What this statement implies is, during the separation process, instead of modifying position components of the waypoints, we can modify, and hence effectively compute, control commands to be applied at each waypoint such that the application of those control commands will result in safely separated trajectories. Working in *control* space instead of *position* space has a tremendous advantage: During the optimization process, if we limit our selection of control commands to reside within the boundaries enforced by aircraft performance limits, we also automatically ensure that the resulting trajectories will be bounded by 3-STAR of the aircraft. Working in position space however does not have this additional and very important benefit. Whenever we modify a waypoint in position space, we need some extra steps to propagate the nonholonomic constraints backwards and forwards to make sure that this recently modified waypoint can still be reached from the preceding one, and the aircraft can also reach the succeeding waypoint from current waypoint. Therefore, we will work in control space when designing our algorithms. The aircraft control command vector contains 3 members, so each waypoint will be treated as a 3-vector, and each trajectory will be treated as a $3n$-vector in control space.

- The simple example demonstrated a *coordinated* collision avoidance scenario where trajectories of all aircraft involved in the encounter are optimized and each aircraft is expected to cooperate in executing the planned maneuvers. Although coordinated collision avoidance is also an active research area, the case that we are actually interested in in this research is where the intruder

aircraft are oblivious to ownship and we optimize just the ownship trajectory against *frozen* intruder trajectories with uncertainties, through time.

Given a set of frozen intruder trajectories, we can compute a collision cost for each point in position-time space. The other two components of the overall cost structure, namely the maneuvering and the deviation costs, are also easily defined in position-time space. As a result, our ownship waypoints will contain time, aircraft control command vector, and position data, and we will be optimizing a $3n$-vector defined in control space against costs computed in position-time space.

Modifying the control component in a waypoint affects the position components in all subsequent waypoints (illustrated in Figure 4-6), which makes it difficult to come up with a compact and easily differentiable formula for computing cost. For this reason, we will turn to numerical solutions rather than analytical ones.

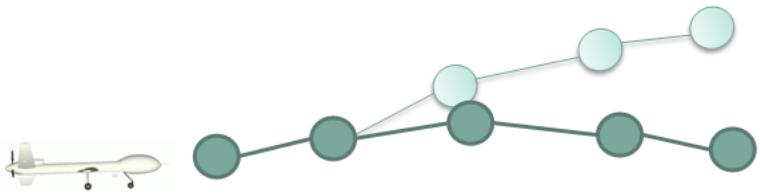


Figure 4-6: When we modify a control component at a waypoint, the position components of the subsequent waypoints need to be updated.

Algorithmically, *Gradient Descent* (GD) is a first-order optimization algorithm that can be used for the modification process, and it can be applied numerically, too. To find a local minimum of a function using GD, we basically take steps proportional to the negative of the gradient of the function at the current point. However, computing the gradient for a $3n$-vector can be very time consuming for large n. In our experiments, we set $n = 30$, yielding a 90-vector to be optimized at each iteration. Therefore, we decided to borrow ideas from another optimization technique that is frequently used in computing inverse kinematics for articulated motion; *Cyclic Coordinate Descent* (CCD). CCD is a member of a class of iterative relaxation algorithms (known as Jacobi or Gauss-Seidel methods) [19] and it was originally developed as

an improved method for solving inverse kinematics problems in robotics [138]. In CCD we also optimize by descending proportional to the negative of the gradient, but instead of considering the whole vector, we work in a single dimension at once and simply optimize 1 coordinate at a time. CCD is usually not as effective as GD in quickly moving towards local minima since we are not taking the steepest possible descent at each iteration. On the other hand, CCD is computationally cheap, easier to implement and it is often very effective. Another important reason for us to prefer a CCD-like iteration is the following: As we mentioned above, in our experiments we worked with trajectories that have $n = 30$ waypoints, i.e. each trajectory is a 90-vector. It is usually the case that when we modify the controls in the first few waypoints, we observe drastic drops in cost. For example, a *turn* command applied at the first few waypoints to steer away from the intruder trajectory/trajectories can very effectively move the rest of the waypoints in the trajectory at very safe points in position-time space. In such cases, we might want to stop optimization if the cost for this trajectory is below some threshold value. CCD allows us to stop optimization loop without ever modifying some/most of the waypoints.

In light of the above discussion, here is the step-by-step description of how we set up function H to optimize a given ownship trajectory t against the cost function derived before (t^s is the nominal trajectory for ownship, u is a set of estimated intruder trajectories to avoid, all trajectories have n waypoints with matching time stamps, and $1 \leq k < n$) :

- Iterate over t, waypoint by waypoint, and perform the following steps for each $w_k \in t$.

- Numerically compute the gradient of cost along vertical acceleration ($\ddot{h}$) component of *control* data in the current waypoint $w_k \in t$. (To do that, first increase $\ddot{h}$ by a small test amount, $\Delta_{\ddot{h}}^{\text{Test}}$, and recompute *position* data for all subsequent waypoints w_i , $k < i \leq n$, to obtain the trajectory t'. Then compute the difference; $\text{Cost}(t', t^s, u) - \text{Cost}(t, t^s, u)$. Also do the same by decreasing $\ddot{h}$ by $\Delta_{\ddot{h}}^{\text{Test}}$ and see how the cost is affected). Based on the gradient, increase/decrease

$\ddot{h}$ by a small increment amount, $\Delta_{\ddot{h}}^{\text{Increment}}$, or leave it unaltered if the slight perturbating of vertical acceleration did not have any effect on the cost.

- Similarly, compute the gradient of cost along turn rate ($\dot{\psi}$) component and modify it using the test/increment amounts $\Delta_{\ddot{\psi}}^{\text{Test}}$ and $\Delta_{\ddot{\psi}}^{\text{Increment}}$.

- And lastly, compute the gradient of cost along airspeed acceleration (a) component and modify it using the test/increment amounts Δ_{a}^{Test} and $\Delta_{a}^{\text{Increment}}$.

- Modification of waypoint w_k is complete for this iteration. Compute the cost a final time (using recently computed control values for w_k and regenerating the trajectory) : If it is below a specified threshold; stop optimization. Otherwise set w_{k+1} as the current waypoint and continue with the iteration.

In our implementations, the described single-pass trajectory optimization runs inside an outer loop since it is almost never enough to run it just once. The outer loop terminates as soon as the cost drops below the given threshold, or if that does not happen, it stops after a certain number of iterations (set to ≈ 200 in our experiments).

4.1.2 Considerations

There are a few important points that we would like to emphasize about computing collision avoidance maneuvers using path-modification based algorithms:

- We represent trajectories using waypoints, and essentially we separate waypoints from each other when planning. However, the aircraft actually have to traverse the link between all pairs of successive waypoints when following the planned trajectory. Therefore the distance between successive waypoints in a single trajectory and the required separation between waypoints that belong to different trajectories should be carefully selected to also geometrically cover/protect the links between waypoints as illustrated in Figure 4-7. In our experiments, we set always-overlapping and large-enough uncertainty regions around successive waypoints of intruder trajectories that safely cover the links

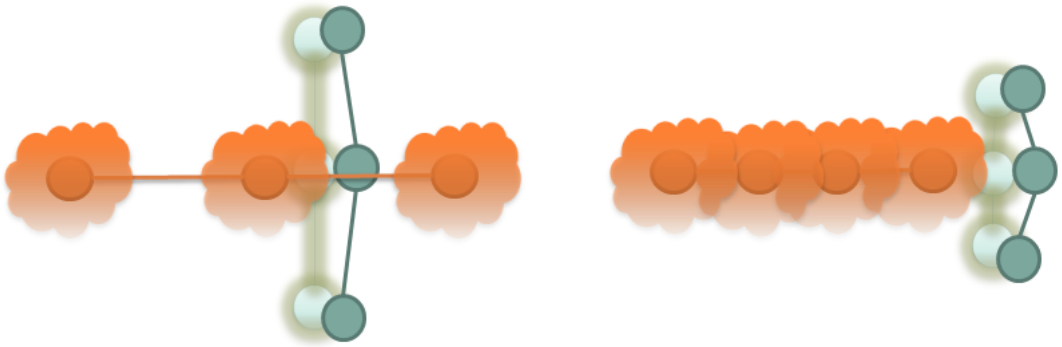


Figure 4-7: In the figure on the left, the selection of the waypoint locations and/or the sizes of the protected zones allow undesirable path crossings. Careful selection of values that consider encounter geometry is necessary to avoid this potential problem, as in the figure on the right.

between waypoints. With that setting, an optimized ownship trajectory that avoids uncertainty regions around all intruder waypoints means that the links are also avoided.

- It is possible that, in the presence of multiple intruders, the initial ownship trajectory that will be fed to the optimization function might geometrically be trapped between estimated intruder trajectories as shown in Figure 4-8. In such cases, the optimization algorithm might not be able to come up with a trajectory that has a low cost. To improve the effectiveness of optimization against such local minima, we might extend the algorithm to construct a few sufficiently different initial ownship trajectories as candidates, have them optimized separately (which can be done in parallel), and then choose the one with the minimum cost. In 3-D, we need at least 3 intruders with carefully planned flight trajectories to realize a local minima scenario. In our tests using our simulation and evaluation platform, CASSATT, we ran single-intruder scenarios. Therefore, it was enough to use a single initial trajectory for the optimization in our experiments.

- Depending on the number of intruders and the number of waypoints used to represent each trajectory, path-modification based algorithms might require a long time to execute even though there are areas for improvement using a

Figure 4-8: When there are enough intruder aircraft that could potentially create local minima problems, we might try optimizing a set of sufficiently different initial candidate trajectories instead of a single candidate. In the figure, the ownship trajectory in the middle (filled green waypoints) is trapped between intruder trajectories. The other two candidate trajectories (outlined green waypoints) will yield lower expected costs after optimization.

parallel processing setup. However, our proposed formulation, including the cost computation, is linear in the number of intruders. For example, with the path-modification technique, adding a second intruder in the collision avoidance planning means that we will just need an additional set of state and trajectory estimations, and we will need to just double the amount of work that is required to compute collision cost. On the other hand, accomodating a second intruder with the MDP/POMDP framework using the same discretizations we had before does not seem practical to be realized with current solvers.

- Ensuring *path feasibility*, i.e. making sure that the aircraft can actually fly the planned trajectory within its performance limits, might be challenging if the path modification is done in position space. Additional steps are needed to propagate nonholonomic motion constraints across the trajectory at every iteration of the modification process. In our implementations, we chose to work in control space rather than position space. This requires an additional step to translate the effects of modifications in control space to the position space due to the fact that we compute cost in position space, but it helps us bypass the complexity of constraint propagation and satisfaction steps.

4.2 Single-Trajectory Collision Avoidance System

Our first path-modification based collision avoidance algorithm is a straightforward application of the main functionalities described above. In this section, we first go over the structure and implementational details of our Single-Trajectory collision avoidance system, and then we provide a comparison of its results with baseline systems.

4.2.1 Structure and Implementation

The simplest planner that avoids a single intruder aircraft by optimizing the given ownship trajectory t^s works as follows:

- Let $o = \{\ldots, \phi_{-2}, \phi_{-1}, \phi_0\}$ be the set of current and past observations available to us. We first generate the estimated trajectory $t^i = G(F(o))$ for the intruder aircraft. At this step, if we are working with very accurate observations from a low-noise sensor, we would be able to estimate the future positions of intruder aircraft with high accuracies, too. This means that the estimated trajectory that we generate for the intruder, t^i, can be constructed with small positional uncertainties around each waypoint (note that the uncertainties and/or the protected airspaces should still be large enough to cover links between successive waypoints). Ideally, we would be avoiding a *tube* that surrounds the air pathway to be actually flown by the intruder aircraft. Normally, we should increase uncertainty as time progresses, and therefore the trajectories to be avoided look more like *cones*. In short, a closely estimated trajectory is easier to avoid as we do not need to consider the full 3-STAR region during collision avoidance maneuvers, but if the sensor and the observations are noisy, then estimations should include large-enough uncertainty regions even though the estimated trajectories will approach the full 3-STAR depending on how noisy the observations are. Figure 4-9 illustrates the sources of uncertainty that should be considered when setting up protected airspaces around waypoints of intruder aircraft trajectories.

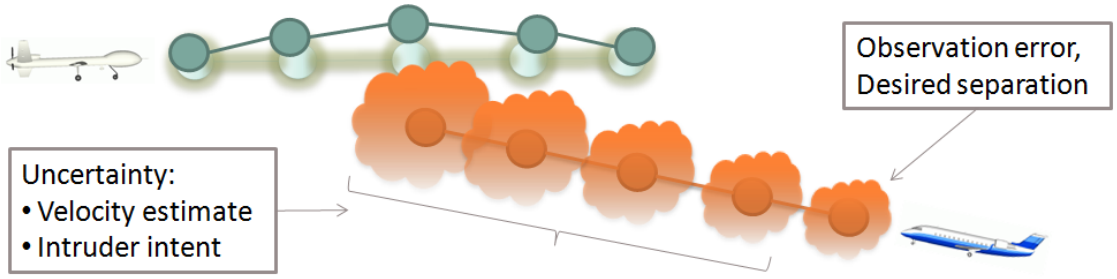


Figure 4-9: The protected airspace around the first intruder waypoint should take into account the observation error and the desired minimum separation between aircraft. The protected airspaces around subsequent waypoints should observe the same conditions as the first waypoint, and also account for uncertainties in the estimated intruder velocity and intruder intent.

- We then let $u^i = \{t^i\}$, and compute the trajectory to be followed by ownship $t^o = H(t^s, u^i)$ that minimizes $\text{Cost}(t^o, t^s, u^i)$. This concludes the planning for the current time step.

In the case of an open-loop encounter scenario where there will be no further observations, we might just execute the computed plan and fly t^o until the end. In our experiments on CASSATT, we receive a sensor reading every second, so we do dynamic replanning: we execute the plan until the next observation is ready, and then we replan using the above steps. A direct implication of performing dynamic replanning is that our algorithm has to run reasonably fast to keep up with the rate at which observations are received (1 Hz in our experimental setting).

Important parameters pertaining to our implementation and their values are presented in Table 4.1. Note that the maximum turn rate for Global Hawk was reported as 2.5 deg/s in Table 2.1 before, but we used 3 deg/s in our experiments to match it to the internal CASSATT parameters.

Other implementation details are as follows:

- Computing the maneuvering cost, i.e. function M, is implemented as multiplying the absolute vertical velocity at the current waypoint by a constant (set to 1.0).

Table 4.1: List of parameters and values used in the implementation of the Single-Trajectory collision avoidance system.

Ownship Performance Limits		
Maximum vertical acceleration, $\ddot{h}$	8	ft/s^2
Maximum turn rate, $\dot{\psi}$	3	deg/s
Maximum airspeed acceleration, a	20	ft/s^2
Trajectories		
Number of waypoints, n	30	
Amount of time between successive waypoints, ΔT	1	s
Collision Geometry		
Minimum desired separation, vertical, Sep_v	100	ft
Minimum desired separation, horizontal, Sep_h	500	ft
Initial uncertainty in intruder position, vertical, Unc_v^0	85	ft
Initial uncertainty in intruder position, horizontal, Unc_h^0	200	ft
Rate of growth of uncertainty, vertical, Unc_v^Δ	50	ft/s
Rate of growth of uncertainty, horizontal, Unc_h^Δ	500	ft/s
Cost		
Vertical velocity cost	$1.0 \times \lvert V_Y^{\text{Ownship}} \rvert$	
Deviation cost	$0.01 \times$ deviation	
Protected airspace violation, base cost	2000	
Protected airspace violation, maximum cost	9000	
Modification Parameters		
$\Delta_{\ddot{h}}^{\text{Test}}$ / $\Delta_{\ddot{h}}^{\text{Increment}}$	0.01/0.1	ft/s^2
$\Delta_{\ddot{\psi}}^{\text{Test}}$ / $\Delta_{\ddot{\psi}}^{\text{Increment}}$	0.01/0.1	deg/s
Δ_a^{Test} / $\Delta_a^{\text{Increment}}$	0.1/1.0	ft/s^2

- Deviation cost computed by function D is also similarly implemented as multiplying the distance between nominal and actual locations by a constant (set to 0.01).

- In order to compute the cost of collision, we first set up a desired protected airspace around each waypoint that belongs to the intruder aircraft trajectory (if ownship never invades a large-enough protected airspace that also accounts for position/intention uncertainties, there will be no collisions). The protected airspace is in the shape of a hockey puck with the following geometry:

$$\text{Height} = 2 \times (\text{Sep}_v + \text{Unc}_v^0 + \text{Unc}_v^\Delta \times \text{waypoint time stamp})$$

$$\text{Radius} = \text{Sep}_h + \text{Unc}_h^0 + \text{Unc}_h^\Delta \times \text{waypoint time stamp}$$

  If ownship waypoint violates this protected airspace, we incur a base cost plus an additional cost that is proportional to the intrusion amount. The Cost_C function is implemented as a sum of incurred costs along ownship trajectory.

The cost formulation is illustrated in Figure 4-10.

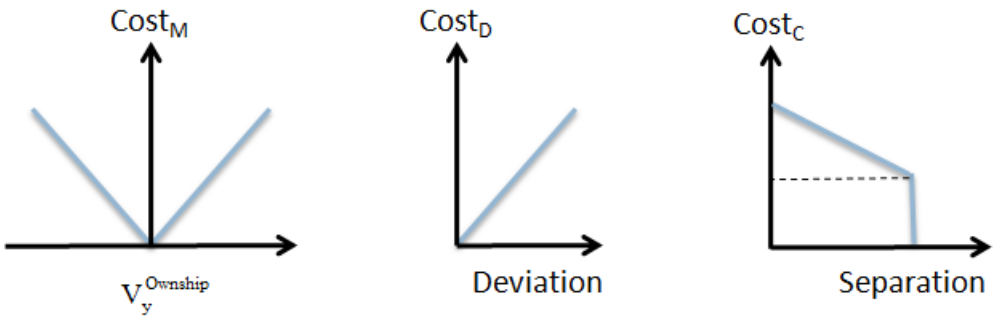


Figure 4-10: Cost formulation for a single waypoint.

4.2.2 Results

The Single-Trajectory collision avoidance system that we have described in this section and the Single Branch-Point collision avoidance system that we will present in

the next section are computationally very expensive. This is because of the fact that we are computing everything online rather than offline computation of logic in the form of quickly executable look-up tables, and also since optimization in continuous space has a lot of parameters that can be tweaked (such as the number of waypoints, the number of iterations and the increment amounts to be applied at each iteration) : it is generally the case that the more the algoriths run with fine-grained parameters, the better the results will be. Therefore, from the timing point of view, we will be only interested in whether useful collision avoidance maneuvers can be generated in the time between two successive observations, which is 1 Hz in our evaluation platform, CASSATT.

In our experiments, we observed that with the parameter settings given in Table 4.1, it is possible to optimize a single ownship trajectory against a single intruder trajectory in less than 0.5 seconds, allowing for real-time performance. However, the Single Branch-Point collision avoidance system of next section optimizes multiple candidate ownship trajectories against multiple probabilistic intruder trajectories, and with the parameter settings we used there (9 ownship trajectories against 6 intruder trajectories) it takes more than 50 times longer to execute. (We would like to note that we used a single computer when testing our path-modification based algorithms, but it is possible to extend our implementation to distribute pairwise optimizations on parallel hardware and still have the Single Branch-Point CAS work under 1 second).

Due to the time complexity of Single Branch-Point CAS, we ran all of our path-modification based algorithms on a small encounter set consisting of 100 encounters. The selection of the 100 encounters was done in a way to make sure that representatives of "difficult" cases were richly included such as:

- Encounters where the intruder approaches ownship from behind, making it difficult for algorithms using limited FoV sensors.

- Encounters where both the vertical and horizontal speeds of intruder are very high. In addition to making the intruder more difficult to avoid, high speeds also imply that the intruder might jump in and out of the sensing range of

Table 4.2: Results for nominal flight and baseline collision avoidance systems.

Algorithm	Sensor	Ratio	Velocity	Acceleration
Nominal	-	1.000000	0.017375	0.001353
Basic CAS (1-D)	Perfect	0.000074	0.295180	0.007720
Basic CAS (1-D)	TCAS	0.000074	0.294333	0.009425
Basic CAS (1-D)	Radar	0.060892	0.144299	0.027926
Basic CAS (3-D)	Perfect	0	0.296551	0.007933
Basic CAS (3-D)	TCAS	3.0096e-08	0.294082	0.011408
Basic CAS (3-D)	Radar	0.033258	0.135120	0.037552
MDP	Perfect	0.008477	0.111626	0.022956

limited FoV sensors unexpectedly.

- Encounters that involve turning with high speeds. The future trajectory of the intruder cannot be estimated very closely by simple differentiation when there is extensive turning.

- Encounters that combine the above, such as ones with scripted flight plans for both intruder and ownship that include short-radius turns with high speeds at closing altitudes before ending up face to face.

Table 4.2 summarizes the results of nominal flight, baseline collision avoidance systems, and a representative system from the MDP/POMDP framework (to help us compare path-modification based methods to MDP/POMDP algorithms) on the 100-encounter test set. In the table we provide the risk ratio, mean vertical velocity in ft/s, and mean vertical acceleration in ft/s^2 for various algorithm/sensor pairs.

Note that path-modification techniques require observations that can be used to localize the intruder to a point in GCS, therefore, we cannot use EO/IR sensor directly, and we will not include comparisons with EO/IR sensor in our discussion below. Also, when relating our results to the MDP/POMDP framework, we will compare against the MDP model that uses a perfect sensor as this was the pairing that lead to the best results.

Table 4.3: Results with perfect sensor.

Algorithm	Sensor	Ratio	Velocity	Acceleration
Single-Trajectory CAS	Perfect	0	0.013605	0.024657

Our first set of tests were conducted using the perfect sensor. Below are two aspects of the computation that we would like to mention:

- When estimating the intruder trajectory, we compute the speed of intruder by simple differentiation using its current and previous locations received from the sensor in the form of observations. We then use this speed value to decide where the future waypoints will be located at in position space.

- Since this is a hypothetical sensor, we took advantage of its noiseless nature, and employed constant protected airspaces around each waypoint. Each protected space was in the form of a sphere with 2000 ft radius. Briefly, we worked with *tube*-like estimated trajectories rather than *cone*-like ones.

The results are shown in Table 4.3 and Figure 4-11. As seen in the table, we easily achieved 0 risk ratio with a mean vertical velocity that is in fact lower than the nominal flight itself. This is due to the fact that the actual nominal flight plans were not all level in some of the encounter scenarios, but our algorithms were aiming for level flight when there is no danger of collision. We would like to note that it is very easy to work with arbitrary t^s when employing path-modification by decreasing maneuvering cost and increasing deviation cost. We worked with the assumption that the part of nominal flight that is in the future is not accessible to our algorithms, and we structured our algorithms to aim for level flight as we did in the MDP/POMDP framework.

The second set of results we would like to present are with the TCAS and radar sensors. Again, we would like to mention two important aspects of the computation that differ from the perfect sensor case below:

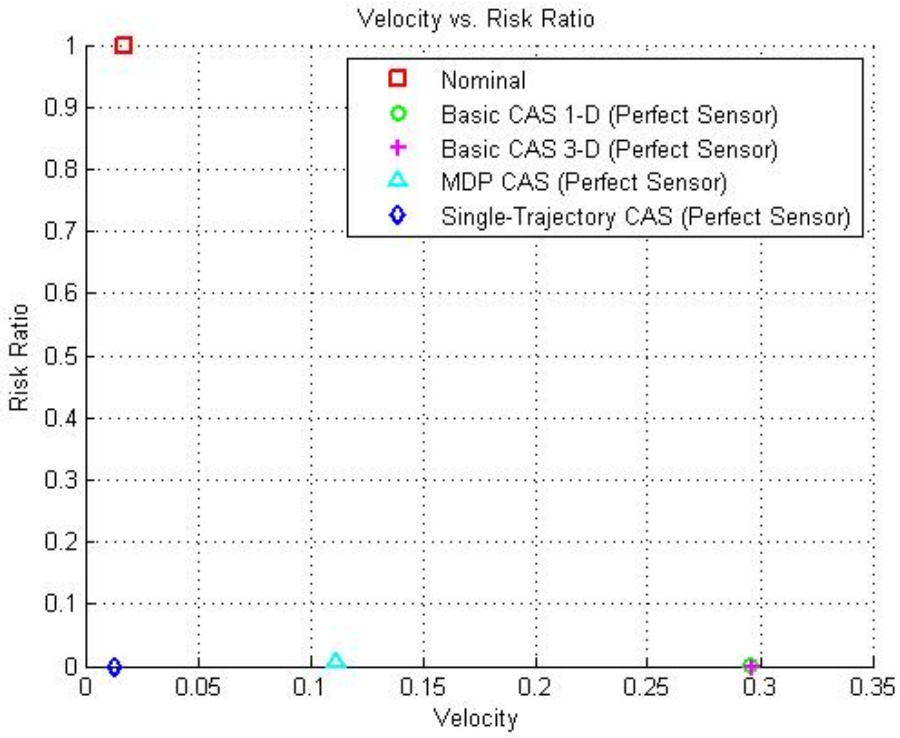


Figure 4-11: Velocity vs. Risk ratio.

- TCAS sensor is good in localizing the intruder vertically, but the noise in bearing estimate is very large. Similarly, the bearing and elevation estimates of radar sensor have large noise that makes it very difficult to localize distant intruders, both vertically and horizontally. Especially with distant intruders, the angular measurement errors severely diminish the effectiveness of using position estimates in computing an estimated velocity for the intruder. Therefore, when estimating the intruder trajectory, we place the first waypoint in the observed location, and do not make any assumptions about the direction of intruder motion. We place all subsequent waypoints at the same location and increase uncertainty in "all" directions. Therefore, the estimated intruder trajectory looks like an enlarging hockey puck rather than a cone as shown in Figure 4-12.

- The hypothetical perfect sensor is a very specialized case that allowed us to use constant protected airspaces, but with TCAS and radar sensors, we computed successively enlarging protected airspaces as described before using the

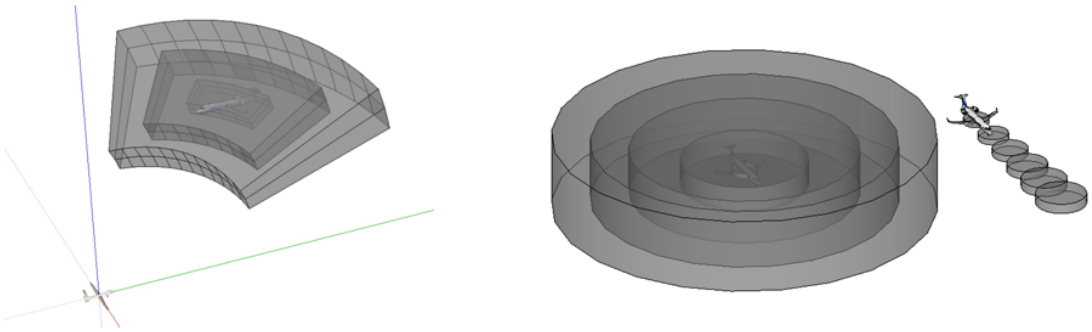


Figure 4-12: The figure on the left depicts the observation noise due to range, bearing and elevation errors. Three standard deviations worth of errors are drawn as transparent regions that the observations might come from, where darker regions indicate higher probabilities. The figure on the right is a comparison of the protected airspaces that need to be set up when assuming motion in all directions, and motion in a certain direction.

Table 4.4: Results with TCAS and radar sensors.

Algorithm	Sensor	Ratio	Velocity	Acceleration
Single-Trajectory CAS	TCAS	0.005515	0.105404	0.044911
Single-Trajectory CAS	Radar	0.000439	0.106621	0.050756

parameter values listed in Table 4.1.

The results with TCAS and radar sensors are shown in Table 4.4 and Figures 4-13 and 4-14. We would like to compare these results with the MDP algorithm: The Single-Trajectory CAS with TCAS and radar sensors was able to perform better by scoring both a smaller risk ratio and a smaller mean vertical velocity than an MDP with perfect sensor. This is in fact an anticipated outcome since we are allowing ownship to move in 3-D rather than restricting the escape maneuvers to vertical plane.

And finally, we present a third set of results using TCAS and radar sensors in Table 4.5 and Figures 4-15 and 4-16. In order to catch up to the lower risk ratios provided by Basic collision avoidance systems, we ran the same algorithm by increasing the rate of growth of uncertainty: We set Unc_v^{Δ} = 100 ft and Unc_h^{Δ} = 900 ft. A comparison of results in this table with the Basic systems show that we can also

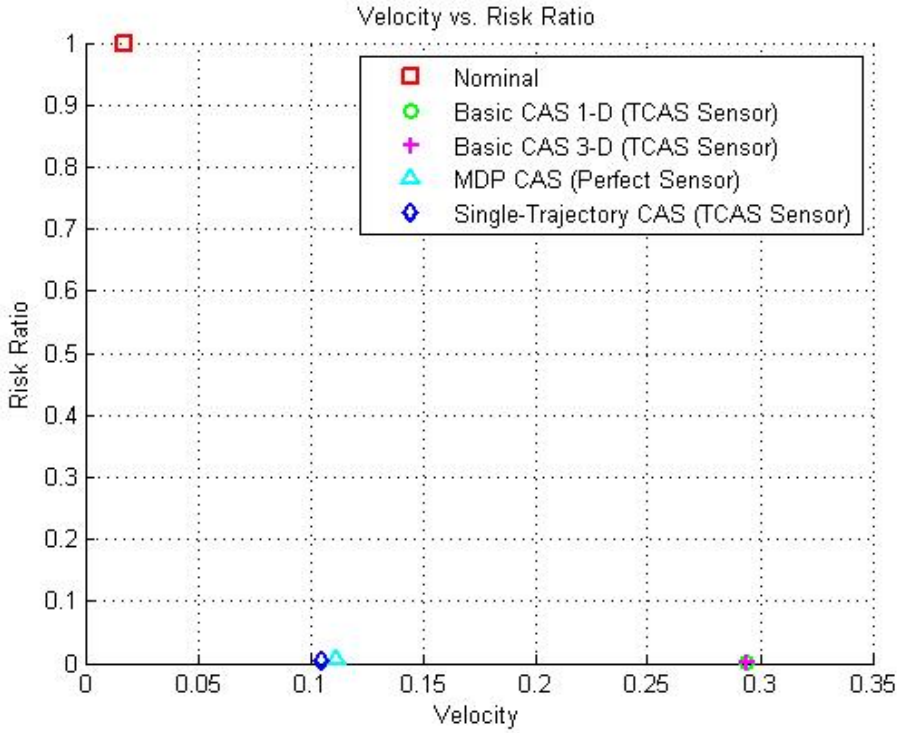


Figure 4-13: Velocity vs. Risk ratio (TCAS sensor).

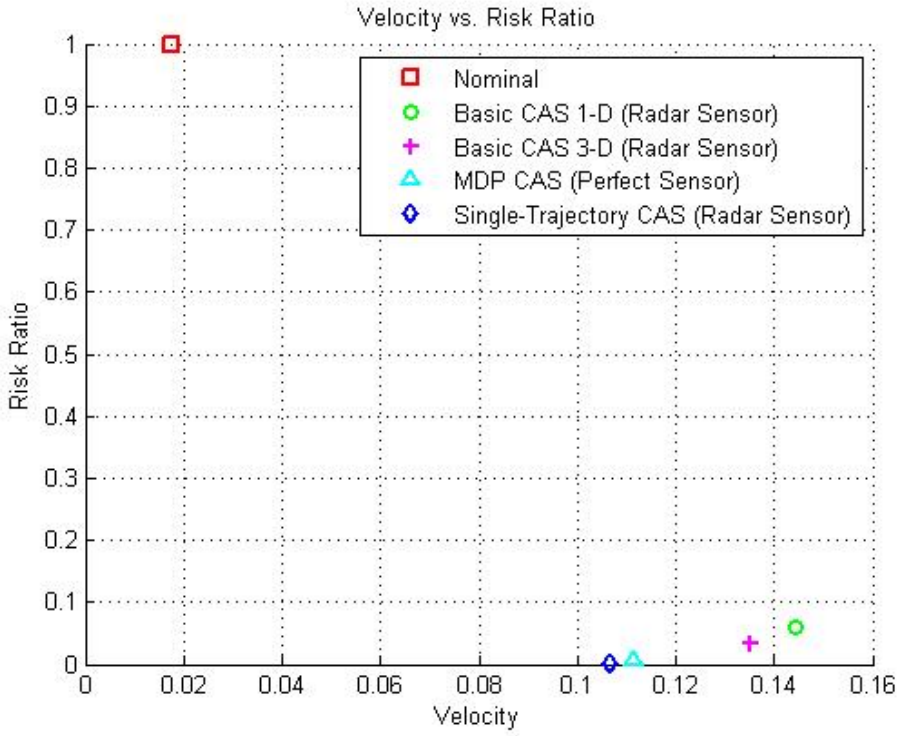


Figure 4-14: Velocity vs. Risk ratio (radar sensor).

Table 4.5: Results with TCAS and radar sensors using a larger uncertainty growth rate.

Algorithm	Sensor	Ratio	Velocity	Acceleration
Single-Trajectory CAS	TCAS	0.000021	0.166395	0.052772
Single-Trajectory CAS	Radar	0	0.132648	0.058688

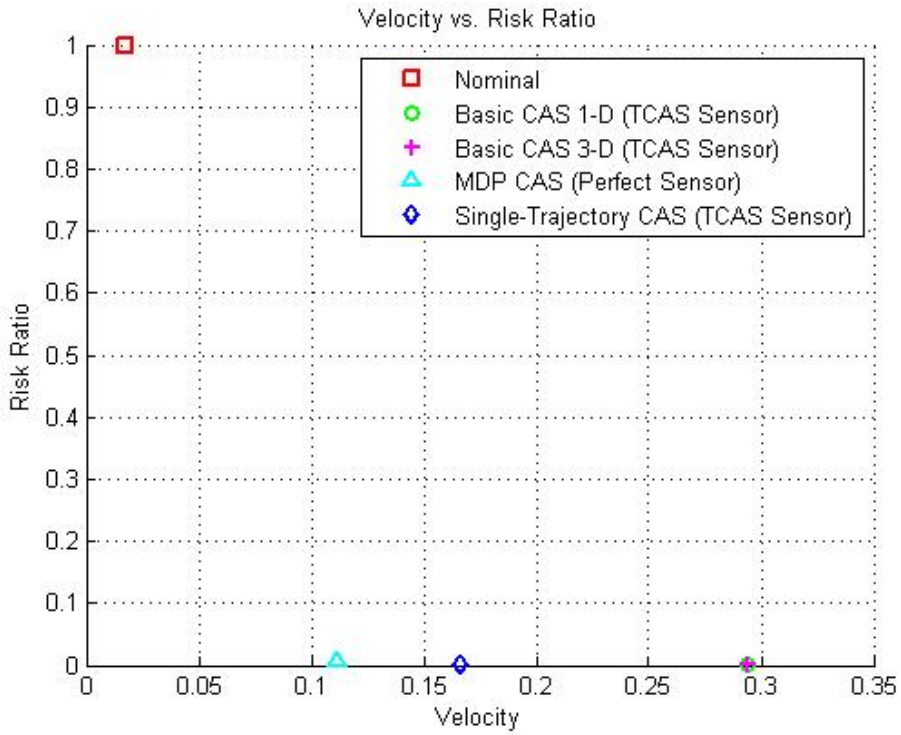


Figure 4-15: Velocity vs. Risk ratio (TCAS sensor).

achieve very small to zero risk ratios and still be able to have mean vertical velocity values that are lower than Basic systems.

4.3 Single Branch-Point Collision Avoidance System

In this section, we will build a planner with the goal of bringing down mean vertical velocity values further, without sacrificing risk ratios. The basic idea that we will employ is depicted in Figure 4-17 and is described below:

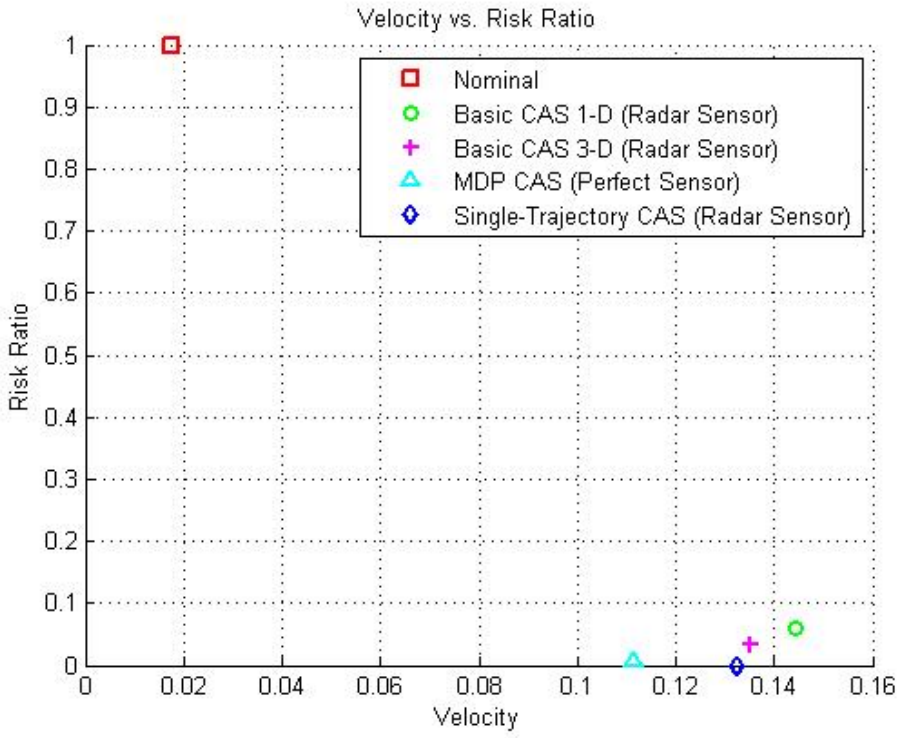


Figure 4-16: Velocity vs. Risk ratio (radar sensor).

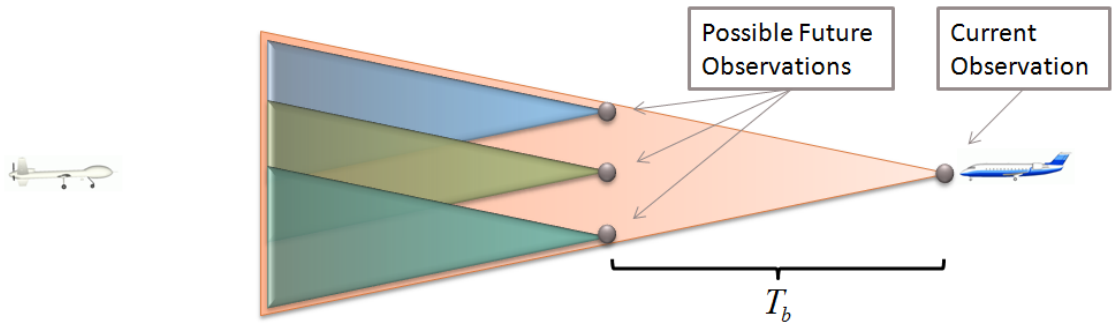


Figure 4-17: In the Single Branch-Point algorithm, we make use of the idea that the next observation that will be received after a certain time T_b will localize the intruder and its estimated future trajectory into a region (one of blue, brown or green cones) that is actually smaller than the region we are currently planning to avoid (red cone).

In the Single-Trajectory CAS, we start by estimating the current state of the intruder based on current and possibly past observations. Then we estimate the trajectory, which corresponds roughly to "all" of the possible locations that it can occupy in the future, and we construct a plan to avoid the whole estimated trajectory. This is the best we can do if no further observations will be received. However, in our simulation environment, we receive an observation every second. The planner that we will build below will take advantage of the fact that the next observation will help us localize the intruder to a smaller region than we are trying to avoid with the Single-Trajectory CAS. For that purpose, the planner will decide among candidate escape plans based on estimates of what the next observation will be. This idea brings us closer to the way the MDP/POMDP solvers work internally, giving rise to both advantages and disadvantages: On one hand, we will be benefiting from making better decisions by looking ahead further and trying to estimate future observations, but on the other hand, we will need to choose our escape maneuvers from a discretized set.

In the following sections, we will first describe the structure and the details of implementation for our Single Branch-Point collision avoidance system, and then we will present results using TCAS and radar sensors. We will not consider perfect sensor in this section since, first, we are already able to achieve zero risk ratios with mean vertical velocity values less than even the nominal flight, and second, with the use of the perfect sensor, the next observation is usually expected to come from a very small (condensed) region that does not provide much benefit from being partitioned and having its sub-regions examined as possible candidates.

4.3.1 Structure and Implementation

We will start by adding a *branching point* in time, T_{branch} or T_b, to the Single-Trajectory collision avoidance system. Let us assume that we have just received an observation at time T_0, we will receive another observation at branching time $T_{\text{branch}} > T_0$, and our planning horizon runs until time $T_{\text{end}} > T_{\text{branch}}$. With that in mind, a Single Branch-Point planner can be structured as follows:

- Construct k sufficiently different partial plans ($t^o_{0-\text{branch}}$) for the time frame T_0-T_{branch}. We can use different strategies for coming up with those k different plans: For example, one of them will usually be the scripted flight plan for ownship (the portion until branching time). To generate other partial escape plans, we can sample actions uniformly from the control space and generate partial plans each of which is generated by applying one of the selected actions repeatedly until T_{branch}. Other methods can be invented to heuristically come up with useful partial plans.

- Compute the state estimate e^i_{branch} for the intruder aircraft at time T_{branch}. The amount of positional uncertainty in the estimated state depends on how noisy the sensor is and how the intruder aircraft might behave between T_0-T_{branch}.

- Generate a set of sample observations $o_{\text{branch}} = \{\phi_1, \phi_2, \ldots, \phi_n\}$ that "cover" e^i_{branch} (in the sense that the union of observation uncertainties cover the positional uncertainty in the state estimate). The probability of observation ϕ_j is written $\Pr(\phi_j)$, $j \leq n$.

- For each partial plan do the following

  - For each $\phi_j \in o_{\text{branch}}$, construct a trajectory

$$t^i_{j,\text{branch-end}} = G(F(o \cup \{\phi_j\}) \text{ , } j \leq n \text{ .}$$

  - Compute a partial plan $t^o_{j,\text{branch-end}} = H(t^s_{\text{branch-end}}, \{t^i_{j,\text{branch-end}}\})$ for the rest of the planning period against each $t^i_{j,\text{branch-end}}$, $j \leq n$.
  - The expected cost of each of these partial plans is equal to

$$\Pr(\phi_j)\,\text{Cost}(t^o_{j,\text{branch-end}}, t^s_{\text{branch-end}}, \{t^i_{j,\text{branch-end}}\}) \text{ , } j \leq n \text{ .}$$

- Finally, pick the partial plan for T_0-T_{branch} that has the lowest expected cost.

The decision process is outlined in Figure 4-18 and a 2-D version of the planner is depicted in Figure 4-19.

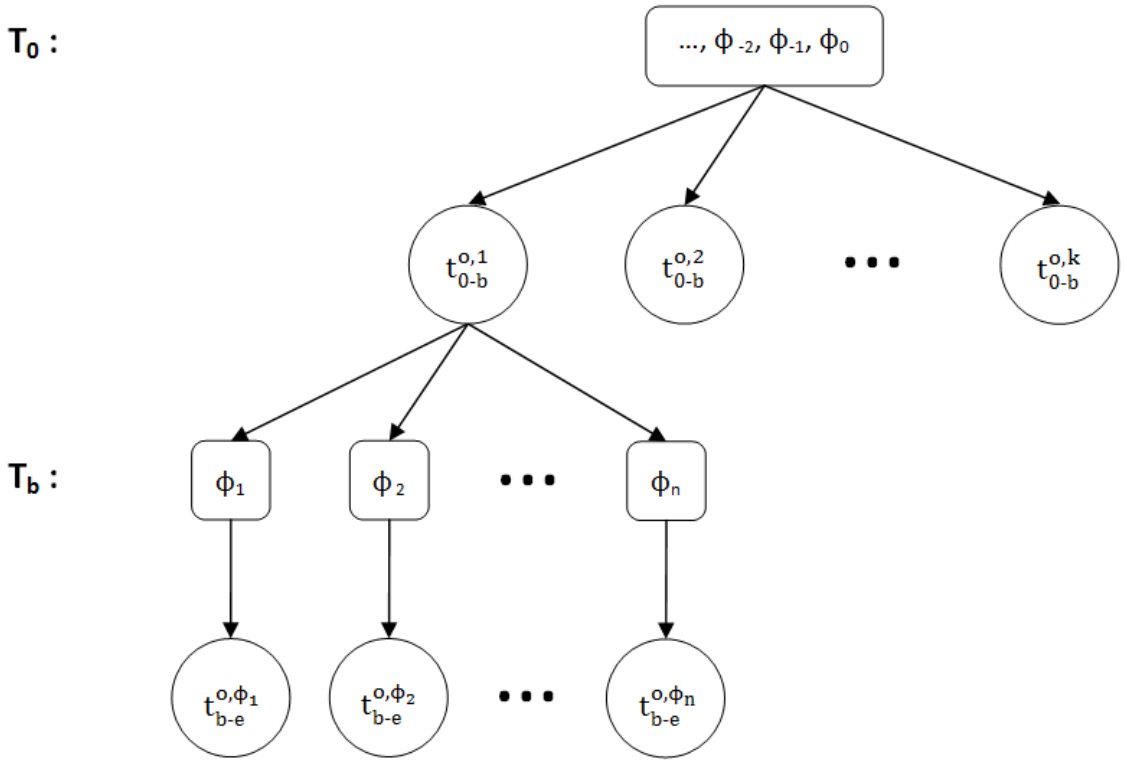


Figure 4-18: Single Branch-Point planner. First, partial evasion plans until branch time T_b are generated. Then, they are evaluated against estimated intruder trajectories based on available observation history at time T_0. The partial plan that scores the minimum expected cost is selected for execution.

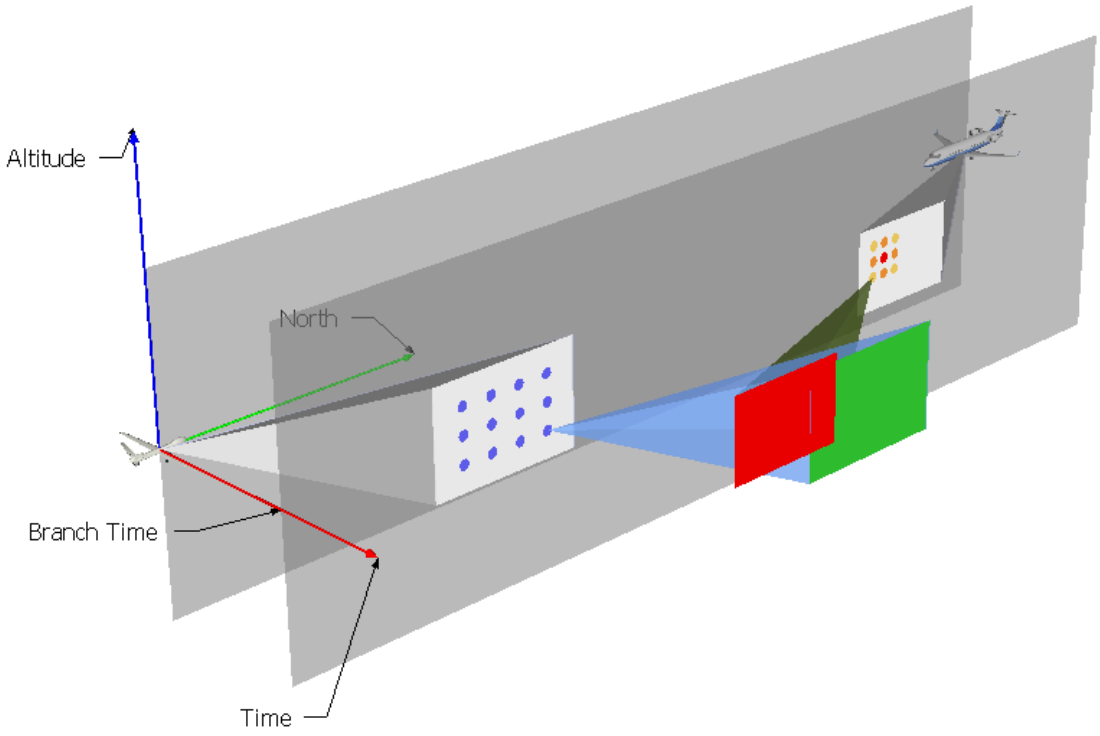


Figure 4-19: Single Branch-Point planner, 2-D Example. In the figure, ownship is located at the origin at time T_0, and we assume that all motion is constrained to *north-altitude* plane. The 2-STAR representations for both aircraft at branch time T_b show set of all possible locations that could be occupied. We first sample from ownship 2-STAR and build a list of candidate partial plans (blue dots). We then compute a list of possible observations of intruder aircraft at T_b (colored dots). Evaluation of candidate plans against possible observations yields which plan has the minimum expected cost.

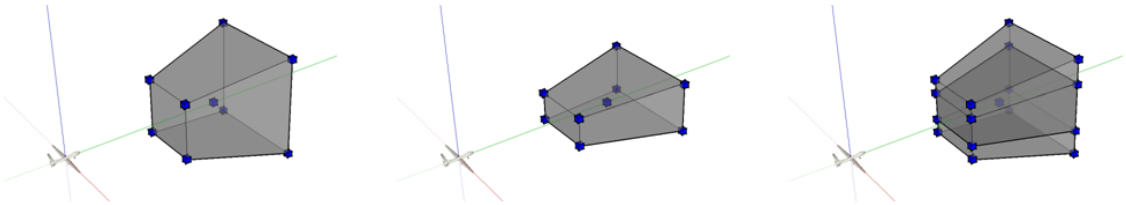


Figure 4-20: In the figures, the reachable regions for ownship at branch time are shown as transparent boxes. The candidate partial plans (indicated by little blue cubes) correspond to the 8 corners of those boxes, and a ninth location that is reached by following the nominal flight plan, which falls somewhere inside the boxes. The figure on the left shows the generated partial plans when we use minimum/maximum vertical acceleration values, and the figure in the middle shows the generated partial plans when half of vertical acceleration values are used. Two sets of partial plans are shown overlapped in the figure on the right for comparison.

This planner allows us to, for example, follow the scripted flight plan for a while and then branch based on the possibilities of observations at time T_{branch}. Given a rich set of partial plans to choose from, it is possible to lower mean vertical velocity values with this planner without affecting risk ratios.

4.3.2 Results

In order to test Single Branch-Point CAS, we set T_{branch} = 1 s, and used same parameter values from Table 4.1 with the exception of working with increased rate of growth of uncertainties (Unc_v^{Δ} = 100 ft and Unc_h^{Δ} = 900 ft) that were used to compute results presented in Table 4.5. The other main functionalities are also the same as in the Single-Trajectory CAS, where applicable.

For the first set of experiments that we would like to present in this section, we generated 8 partial plans that correspond to 8 extreme corners of ownship 3-STAR at time T_{branch} that are computed by applying all combinations of minimum/maximum vertical acceleration, turn rate, and airspeed acceleration values between T_0-T_{branch}. We then added $t^s_{0-\text{branch}}$ as a ninth alternative. Generation of candidate partial plans is shown in Figure 4-20.

We generated 6 estimated observations to optimize against: 4 of them were as-

Table 4.6: Results with TCAS and radar sensors.

Algorithm	Sensor	Ratio	Velocity	Acceleration
Single Branch-Point CAS	TCAS	0.000375	0.181029	0.049002
Single Branch-Point CAS	Radar	0	0.136107	0.054050

signed 0.2 probability each and their origins were computed by adding $\text{Unc}_h^{\Delta} \times T_{\text{branch}}$ to the current location estimate of the intruder in $+/-east$ and $+/-north$ directions. The remaining 2 of them were assigned 0.1 probability each and the origins were computed by adding $\text{Unc}_v^{\Delta} \times T_{\text{branch}}$ to the current location estimate of the intruder in $+/-altitude$ directions. Generation of estimated observations is shown in Figure 4-21.

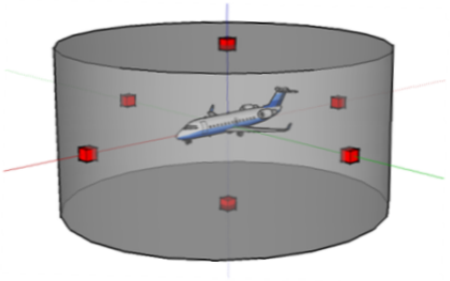


Figure 4-21: We used 6 estimated observations (indicated by red cubes); 4 of them are located horizontally around the current observation with 0.2 probability each, and 2 of them are located vertically below and above current observation with 0.1 probability each.

The results with the above settings are shown in Table 4.6 and Figures 4-22 and 4-23. When we compare this table to Table 4.5, we see that we have comparable risk ratios, but we were not able to lower mean vertical velocity values. This is an expected outcome, since 8 of the partial plans were constructed by applying minimum/maximum vertical acceleration values. The results from this experiment emphasize the importance of having enough variety in the candidate partial plans.

For our second and final set of experiments, we used half of minimum/maximum vertical acceleration values in generating 8 partial plans. The rest of the settings remained the same as in the first case. The results given in Table 4.7 and Figures 4-24 and 4-25 show that a careful selection of candidate partial plans can in fact

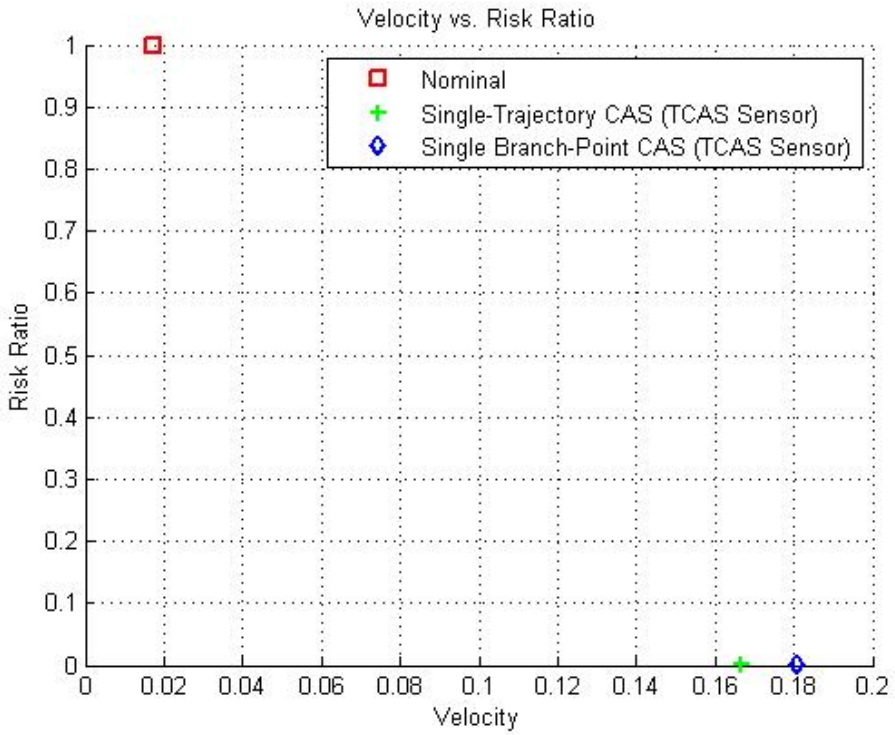


Figure 4-22: Velocity vs. Risk ratio (TCAS sensor).

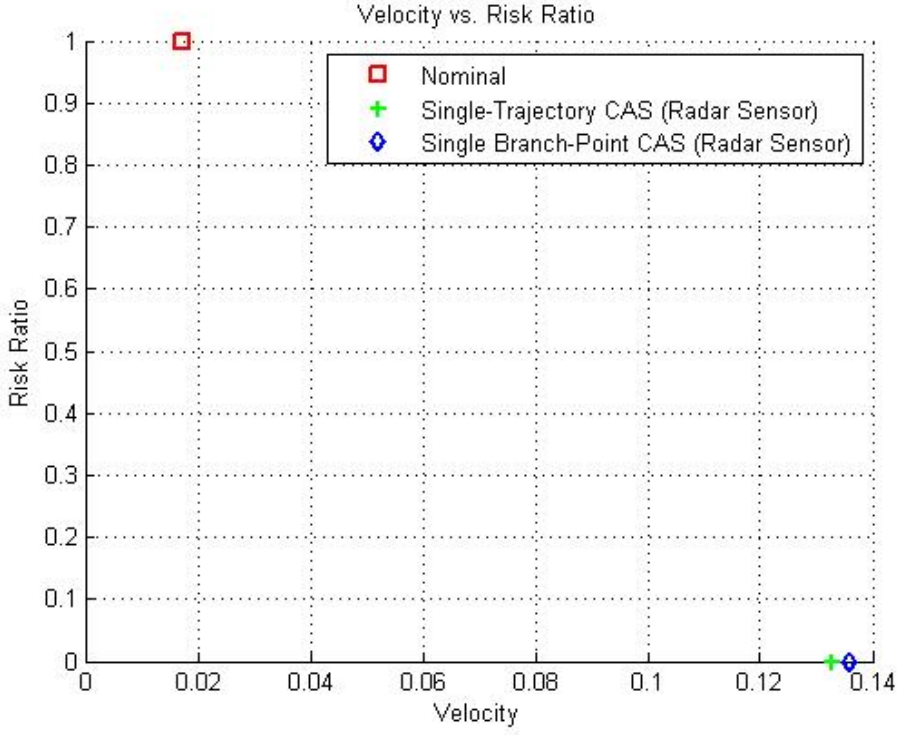


Figure 4-23: Velocity vs. Risk ratio (radar sensor).

Table 4.7: Results with TCAS and radar sensors using half of minimum/maximum vertical acceleration values.

Algorithm	Sensor	Ratio	Velocity	Acceleration
Single Branch-Point CAS	TCAS	2.6179e-07	0.160484	0.029998
Single Branch-Point CAS	Radar	0	0.107135	0.033706

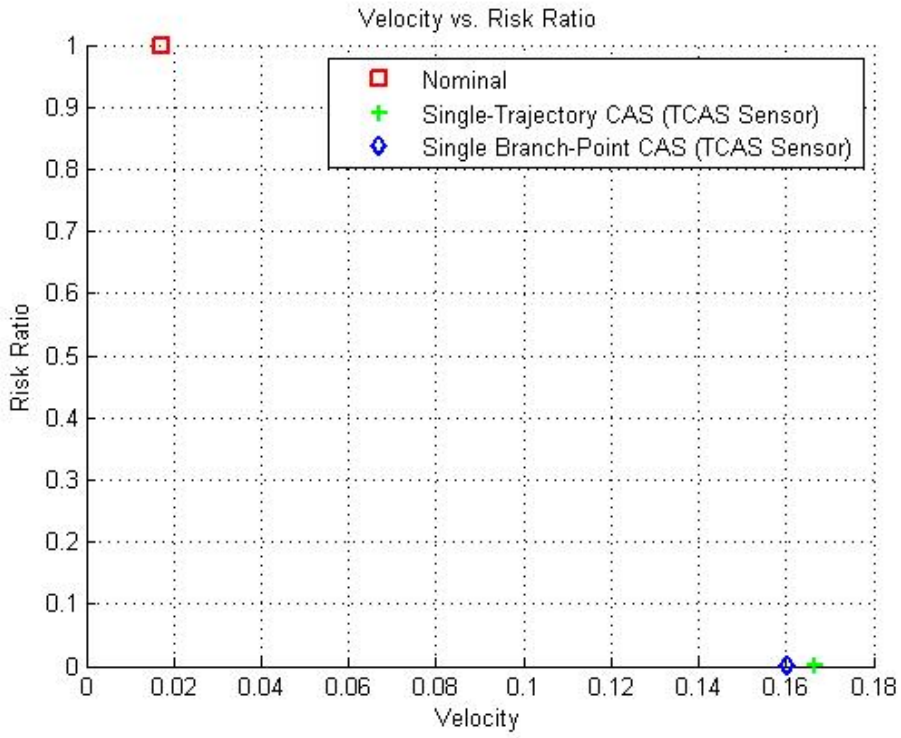


Figure 4-24: Velocity vs. Risk ratio (TCAS sensor).

enable us to reach our goal of reducing the mean vertical velocity values further by maintaining comparable risk ratios (compared to Single-Trajectory CAS).

4.4 Discussion

In this section, we first describe major inherent limitations of path-modification based collision avoidance models and suggest ideas on how to further improve their performance. Then we present a short assesment in general and also in relation to the MDP/POMPD framework.

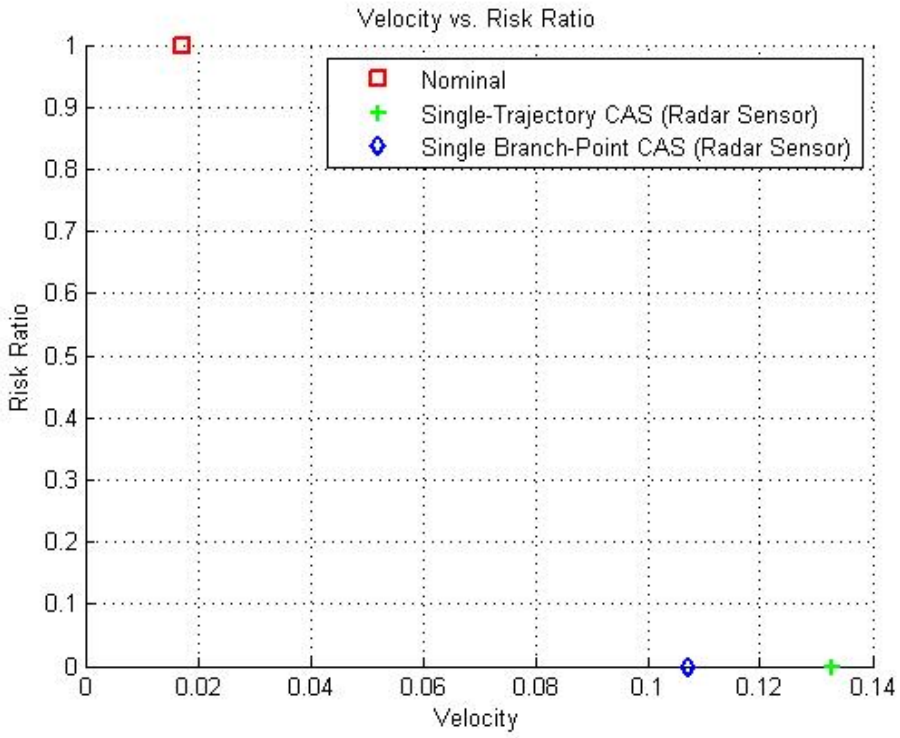


Figure 4-25: Velocity vs. Risk ratio (radar sensor).

4.4.1 Model Limitations

An important motivation for us to shift from the MDP/POMDP framework—an offline optimization defined over enumerated lists—to path modification—an online iterative improvement process over continuous spaces—was to work around the problems caused by the need to use discretized sets of values. Even though we tried to limit the discretizations to a minimum, there are still 2 areas where working with sets of enumerated items finds its place in the path-modification process:

- An ideal representation for a continuous trajectory would be a closed-form expression that is differentiable in the variables that we would like to optimize, but due to the difficulty of deriving such expressions in the collision avoidance problem domain, we chose to simply define our trajectories using ordered sets of waypoints in position-time space. This representation requires additional care in protecting the links (segments) between successive waypoints against collisions as we mentioned in Section 4.1.2. An alternative approach would

be to represent trajectories in terms of ordered sets of segments (with linear or possibly more complex shapes). Even though such a representation has its immediate benefits, most of the functionalities described above have to be altered in non-trivial ways to allow for collision testing between segments, cost accumulation along segments, modification of segments instead of points, etc.

- The Single Branch-Point planner chooses the evasion maneuver among a set of candidate partial plans that are optimized against a set of estimated future observations of the intruder aircraft. Both sets should contain enough and sufficiently different elements for satisfactory performance, and also as few elements as possible for fast computation at the same time. Good heuristics need to be developed and/or algorithms need to be revised for parallel execution in order to both improve the results and keep the overall computation time below the frequency at which observations are received.

In Section 4.1.2 we have also mentioned that path-modification can be very time-demanding depending on values of certain parameters, and we have presented two more limitations that we will just remind here without details: The local minima problem that arises when we let some trajectories to be frozen, and the need to ensure path feasibility when we work in position space.

Another aspect that we would like to mention in this section is; path-modification process has many internal and external parameters like the MDP/POMDP models, and the performance can significantly be improved by systematic study and tuning of parameters with more experiments than we were able to conduct.

And lastly, we would like to emphasize the direct relation between the sensor accuracy and the effectiveness of evasive maneuvers. This is in fact a common statement that applies to all collision avoidance algorithms, but since we are not discretizing our observations in path-modification based methods, we benefit much more from using better sensors with path-modification based systems than with MDP/POMDP based systems.

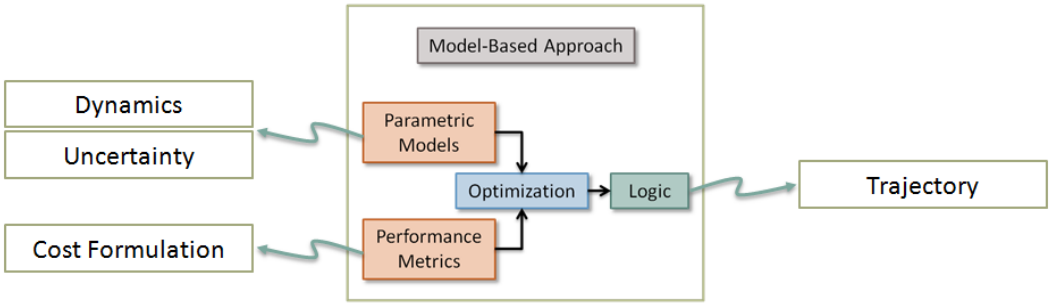


Figure 4-26: Model-based approach, path-modification based collision avoidance logic.

4.4.2 Assessment

Based on experimental results we conclude that:

- Like our MDP/POMDP models, a parametric implementation of the path-modification method is flexible enough to accomodate a variety of sensor modalities, intruder behavior, aircraft dynamics, and cost functions as shown in Figure 4-26.

- Using a short time horizon that can be represented by trajectories with a small number of waypoints, collision avoidance maneuvers for encounters involving 2-3 aircraft can be computed in real time on a single processor with path-modification based techniques. In our experiments, we used a 30 second look-ahead time that was represented by trajectories with 30 waypoints each for encounters between ownship and a single intruder, and we were able to run our Single-Trajectory planner within the time period between the reception of two successive observations. Using a more sophisticated planner such as the Single Branch-Point planner, increasing planning horizon, representing trajectories with more waypoints, and/or including more than 2-3 intruders can very quickly take planning outside real-time computation boundaries. However, path-modification lends itself naturally to parallelization, and usually, as long as a single ownship trajectory can be optimized against a single intruder trajectory in real time, any number of candidate plans can be optimized against any number of estimated trajectories of any number of intruders in real time,

as well, using parallel computation.

- Computation of evasive maneuvers using path modification scales linearly in the number of intruder aircraft. This is a very big advantage that places path-modification method among the techniques that can actually find practical use in the aviation domain.

- Thinking of our path-modification based systems as black boxes, we can say that: we worked with realistic input and output (our observations came from realistic simulations of various sensors and we computed full 3-D aircraft control command vectors), we tested our systems on a very high fidelity simulation/evaluation software (CASSATT), and real-time performance can be achieved with the help of parallel processing. As a result, the systems we built are practically close to turn-key modules that can be deployed on actual aircraft with little effort.

- Similar to our MDP/POMDP development, sophisticated tools to graphically visualize, analyze and optimize many internal and external parameters and intermediate computation results are likely to be necessary, especially for rigorous verification of algorithms.

Chapter 5

Conclusions and Recommendations for Future Research

In this chapter, we will summarize the research presented in this thesis, point out important contributions, and suggest directions for future research for further improvement of the performance of the developed collision avoidance systems.

5.1 Summary

Throughout the course of this study, we worked with realistic aircraft state vectors, realistic aircraft control command vectors, and four different sensor modalities (implementation of TCAS was provided by MIT Lincoln Laboratory, we implemented the perfect, radar and EO/IR sensors based on realistic specifications). We tested our algorithms on a very high fidelity simulation and evaluation platform that was also used in prior TCAS and UAV sense-and-avoid studies. Our simulated encounter scenarios were based on actual radar data collected and analyzed at MIT Lincoln Laboratory, and our algorithms demonstrated real-time performance with realistic aircraft dynamics and 1-to-1 scaled world dimensions in our simulations (with the exception of Single Branch-Point planner, which requires parallelization to actually run in real time).

We demonstrated the feasibility of two approaches to aircraft collision avoidance

problem:

- In the first part of this thesis, we applied the MDP/POMDP framework to collision avoidance domain. Our models were fully in charge of commanding ownship to maintain planned flight, avoid incoming traffic, and gather information to further reduce risk of collision, without the need for additional lower-level or higher-level planners. This approach to collision avoidance planning was based on offline optimization using discretized sets of states, observations and actions. The evasive maneuvers were planned in the vertical plane, similar to TCAS.

- In the second part, we introduced the path-modification technique and two collision avoidance models based on that technique. The nature of our second approach was to iteratively improve a given initial flight plan online, and to work in continuous state, action and observation spaces. Path-modification technique also allowed us to plan full 3-D evasive maneuvers.

The results of simulated experiments with our algorithms showed that our collision avoidance systems are comparable to/better than TCAS and some hand-crafted baseline systems.

We designed parametric models and employed a model-based optimization approach to make it easy to accomodate various aircraft and sensor pairs in our collision aviodance systems.

We built a versatile software application, called the Encounter Analyzer (described in Appendix D), which is an initial step in the design of sophisticated visual analyzer/debugger systems for large POMDPs. We also implemented two other software packages: the POMDP Generator (described in Appendix B) automatically generates parametrized POMDPs, and the POMDP Processor (described in Appendix C) converts a POMDP specification into a PPOMDP for very fast belief-state updating.

5.2 Contributions

Major contributions of presented research are as follows:

- Our MDP/POMDP based algorithms are some of the first examples of application of the original POMDP formulation to a realistic UAV collision avoidance problem. We experimented with compact representations and careful designs of state, action and observation spaces in order to be able to model collision avoidance systems with sufficient details while still staying within the reach of feasible computation capabilities of state-of-the-art solvers.

- We devised a novel method that processes POMDP formulations offline, and converts them to PPOMDPs for very fast belief-state updates. The speed-up gained from this approach allowed us to run our POMDP collision avoidance models in real time.

- We generated very large POMDP models with varying characteristics (the use of different sensor modalities ranging from hypothetical perfect sensing to very noisy and limited field-of-view sensing lead to interesting observation models that produce very-focused to highly-smudged-out belief-states). Some of our POMDP models were used in the testing and improvement of the SARSOP solver.

- We took the first steps in designing sophisticated visual analyzer/debugger software for POMDPs by developing Encounter Analyzer as an example, which provided enormous benefits to us in especially visualizing the belief-states at every step of an encounter. Being able to grasp the evolution of belief-states was very helpful to us in quickly improving our models and tuning parameter values.

- We introduced the path-modification technique, a novel approach to dynamic collision avoidance, and we built two collision avoidance systems that are based on the path-modification idea. We demonstrated the feasibility of both systems by experimental results. Path-modification method allows planning full 3-D evasive maneuvers, and the cost formulation is linear in the number of intruders, which make it a very promising and practical idea for actual deployment on real

platforms.

- We employed a model-based approach to developing optimized collision avoidance logic in both of our MDP/POMDP and path-modification frameworks, and our results can be referred to in demonstrating the feasibility of and promoting the model-based optimization approach.

5.3 Recommendations for Future Research

Below is a list of some directions that could be followed to further improve the performance of our collision avoidance systems:

- Both sets of our MDP/POMDP and path-modification based collision avoidance models have many internal and external parameters. We believe that much better performance could be achieved by systematic experimentation and optimization of parameter values. System performance curves generated for each parameter individually would be very helpful in optimizing their values. During the optimization phase, it is likely to be necessary to test the models using a lot more encounters than we were able to do, but running experiments separately for each parameter could be carried out in parallel.

- We have successfully identified some major shortcomings in our state space design for some of our POMDP models (such as the lack of pitch/roll angles, and the benefit that could be achieved by finer discretizations of important dimensions), but we have not been able to take steps in improving them within the timeframe of this research. It would be well worth trying to apply our findings as we have reason to believe that following this suggestion would lead to better performance delivered by our models. Also, POMDP solvers have been improving to accomodate larger models (for example, by building on the fact that in various problem domains some components of states are actually fully-observable rather than partially-observable, and leveraging this idea by separating those two groups of components in distinct sets that are to be handled

differently). These improvements might provide room for being able to work with larger state spaces that include the missing components we have identified.

- The performance of the Single Branch-Point planner could be improved by better selection of the candidate partial plans and estimated future observations. The branching time is also a very important key parameter, the optimization of which could have great impact on the performance.

- In our description of the Single Branch-Point planner we stated that the pairwise optimizations of candidate partial plans versus estimated future observations could be carried out in parallel, but we have not provided implementations and test results of an actually parallelized system due to the way our test hardware/software were set up. Even though we are confident that a parallelized implementation of the Single Branch-Point planner could run in real time as long as a single pairing of partial plan/estimated observation could be optimized in real time, this claim should be backed up by an actual working implementation. This is due to the fact that even though important computations could be done in parallel, some internal data should first be distributed among the computation nodes, and some intermediate results should be brought together after the computation nodes finish their tasks. It needs to be demonstrated that this overhead that is common in parallel computing will not break the real-time performance claim.

- We have formulated our path-modification based collision avoidance systems to handle multiple threats, but the encounter scenarios that we used to test our systems on CASSATT were generated for a single intruder aircraft. It needs to be demonstrated by simulations against multiple intruders that our models can actually scale up easily to more complex scenarios.

- Finally, we utilized importance sampling to be able to evaluate our collision avoidance systems on up to 15,000 encounter scenarios at a time, but ideally they need to be evaluated and optimized on hundreds of thousands of scenarios

before they can mature enough for actual deployment on real platforms.

Appendix A

Pseudocode

Algorithm 1 Simulation of Sensor Readings

Input: Aircraft state vectors for ownship and intruder from CASSATT
Output: Sensor reading

```
Find coordinates of intruder aircraft in local coordinate system of own aircraft
Compute true reading (value without noise)
if returning a false positive reading then
    return a randomly generated reading that complies with sensor specifications
else if returning a false negative reading then
    return no intruders inside sensing range
else
    if other aircraft is outside sensing range then
        return no intruders inside sensing range
    else
        return sensor reading (i.e., true reading modified according to error model)
    end if
end if
```

Algorithm 2 Perfect Sensor - Observation Model

Input: End state, Action, Set of all observations
Output: List of probabilities

```
if end state is a START state, or a DONE state, or is outside sensing range then
    Add to the list the single observation that reports only V_Y^Ownship
else
    Locate the bin in the Relative Coordinate System that corresponds to the end
state and add to the list the single observation that corresponds to this same bin
end if
```

Algorithm 3 TCAS Sensor - Observation Model

Input: End state, Action, Set of all observations

Output: List of probabilities

if end state is a START state, or a DONE state, or is outside sensing range **then**

Add to the list the single observation that reports only V_Y^{Ownship}

else

Locate the bin in the Relative Coordinate System that corresponds to the end state and enlarge the boundaries of that bin by a *margin* determined by the sensor error model

for each observation in the observation set **do**

if the bin that corresponds to the observation overlaps with the enlarged bin **then**

Add the observation to the list with probability proportional to the overlap area

end if

end for

if the false negative measurement probability of the sensor is greater than zero **then**

Add the observation that reports only V_Y^{Ownship} (with false negative measurement probability of the sensor)

end if

end if

Algorithm 4 Radar Sensor - Observation Model

Input: End state, Action, Set of all observations
Output: List of probabilities

if end state is a START state, or a DONE state, or is outside sensing range **then**
  if the false positive measurement probability of the sensor is zero **then**
    Add to the list the single observation that reports only V_Y^{Ownship}
  else
    Add to the list all observations with equal probability
    Add to the list the single observation reporting only V_Y^{Ownship} (with probability 1.0 – false positive measurement probability)
  end if
else
  Locate the bin in the Relative Coordinate System that corresponds to the end state and enlarge the boundaries of that bin by a *margin* determined by the sensor error model
  for each observation in the observation set **do**
    if the bin that corresponds to the observation overlaps with the enlarged bin **then**
      Add the observation to the list with probability proportional to the overlap area and weighted by the sensor error model (discretized Gaussian)
    end if
  end for
  if the false negative measurement probability of the sensor is greater than zero **then**
    Add the observation that reports only V_Y^{Ownship} (with false negative measurement probability of the sensor)
  end if
end if

Algorithm 5 EO/IR Sensor - Observation Model

Input: End state, Action, Set of all observations
Output: List of probabilities

if end state is a START state, or a DONE state, or is outside sensing range **then**
  if the false positive measurement probability of the sensor is zero **then**
    Add to the list the single observation that reports only V_Y^{Ownship}
  else
    Add to the list all observations with equal probability
    Add to the list the single observation reporting only V_Y^{Ownship} (with probability 1.0 – false positive measurement probability)
  end if
else
  Locate the bin in the Relative Coordinate System that corresponds to the end state and determine the minimum and maximum angles that 'see' this bin
  for each observation in the observation set **do**
    $d \leftarrow 0$
    if the angle that corresponds to the observation is outside the minimum and maximum angles that 'see' the end state **then**
      $d \leftarrow$ angular distance to the end state
    end if
    Add to the list all observations with probabilities that are proportional to the density at d of a Gaussian PDF with a zero mean and standard deviation σ given by the elevation error.
  end for
  if the false negative measurement probability of the sensor is greater than zero **then**
    Add the observation that reports only V_Y^{Ownship} (with false negative measurement probability of the sensor)
  end if
end if

Algorithm 6 State Transition Model

Input: Start state, Action, Set of all states

Output: List of probabilities

Locate V_Y^{Ownship} bin corresponding to the given start state

Predict new V_Y^{Ownship} bin boundaries according to the given action (if 'accelerating up', bin boundaries increase by the applied acceleration times ΔT, if 'maintaining', no change occurs, and if 'accelerating down', bin boundaries decrease); add a fixed, small margin when enlarging boundaries

if given start state is a START state **then**

Add to list the START states that overlap with the predicted V_Y^{Ownship} bin (with probability proportional to the overlap amount and also scaled according to the probability of staying in START state)

Add to list all other states that overlap with the predicted V_Y^{Ownship} bin (with probability proportional to the overlap amount and also scaled according to the probability of appearing in any other state)

else if given start state is a DONE state **then**

Add to list the DONE states that overlap with the predicted V_Y^{Ownship} bin (with probability proportional to the overlap amount)

else

Locate all the bins (X, Y, V_X^{Relative} , V_Y^{Intruder} , V_Y^{Ownship}) corresponding to the given start state

for all possible values of intruder horizontal and vertical acceleration model values **do**

Predict new state (new bin boundaries) using dynamics equations, ΔT, and performance limits of our aircraft

Add to list all the states that overlap with the predicted state (with probability proportional to the overlap amount)

end for

end if

Appendix B

POMDP Generation

In the course of this research, we implemented a software application, called POMDP Generator, that generates textual POMDP formulations for all four types of sensors. We have already provided pseudocodes for major algorithms used by the POMDP Generator in previous sections (reward, observation and state-transition functions) and we have also described important parameters of those algorithms. In this section we will briefly go over the POMDP generation process.

POMDP Generator reads in a couple of text files (specification files) that contain values of various parameters. These specification files are easily editable, and different POMDPs can be generated using different configurations of the parameter values. Collectively, the following data are required and gathered from the specification files:

- Specifications for the respective sensor (given in Table 2.2)
- Controller frequency (ΔT)
- Aircraft dynamic model (vertical velocity limits, and vertical acceleration magnitude)
- Geometry of the desired protected space around own aircraft
- State space
- Action space

- Observation space (for most sensor types, this is automatically derived from the state space)
- Reward model (crash cost, protected airspace violation cost and vertical velocity cost)
- Transition probabilities from START state of the POMDP formulation
- Vertical and horizontal acceleration models for the intruder

After gathering necessary information, the following tasks are performed in the given order:

1. Number of states, actions and observations are determined using the specifications of the state, action and observation spaces. Symbolic names for all states, actions and observations are computed and recorded as part of the POMDP formulation. (Most of the algorithms use integer indices when dealing with states, actions and observations, but there are some algorithms that make use of symbolic names to quickly extract information about a given state, action or observation. Also the symbolic names are good for debugging purposes).
2. Initial belief-state is computed and recorded. For all sensor types, initial belief-state contains a uniform probability distribution over the duplicated START states, but the number of duplicated START states depends on the state space (more specifically, the number of bins in the discretization of V_Y^{Ownship} values).
3. POMDP Generator iterates over all state-action pairs and invokes the transition function (Algorithm 6 in Appendix A) for each pair to compute and record the transition model.
4. POMDP Generator iterates over all states and invokes the relevant observation function (one of Algorithms 3, 4, 5, or 2 in Appendix A) for each state to compute and record the observation model.
5. For each state, reward function is invoked and reward model is recorded.

The output of the POMDP Generator is a POMDP file. The POMDP files we generated and used in our tests have sizes ranging approximately between 45–55 MB.

Appendix C

Processed POMDP

One major contribution of this study is the development of a novel method that drastically reduces the computation time of the belief-state update process. The method involves the following components:

- **An extended transition model**: We combine the transition and observation models of a POMDP formulation into a single model that we call an extended transition model, represented as a large lookup table. Given a start state, an action, and an observation, the table provides a list of end states we can land in and their probabilities. We developed a software package (called POMDP Processor) that reads in a POMDP formulation, computes the extended transition table, and outputs it in the form of a new POMDP formulation that has a state set, an action set, an observation set and an extended transition model. We call this new POMDP formulation a Processed POMDP, or PPOMDP for short. A PPOMDP file is usually larger in size than a POMDP file. PPOMDPs can be computed offline, and their main purpose is time efficiency during belief-state updates rather than memory or storage efficiency.

- **A special data structure for representing belief-states**: Similar to the PPOMDPs, this data structure is designed with time-efficient computation in mind. It occupies more than four times the size of a naïve belief-state representation in the memory. All of the required space is allocated at once during

initialization, and no further dynamic allocation or deallocation is performed. It uses a sparse representation to easily store, fetch and iterate over only the states with non-zero probabilities. It uses the pointer data type (in the C Programming Language) to switch between arrays containing various data and thereby it never requires resetting an array, or copying array values from one place to another in memory. It also keeps running sums of probabilities; therefore it also never requires summing over array elements.

In our tests, we witnessed speed-ups in belief-state updates up to factors of 100 to 1000. The strength of the PPOMDP formulation comes from the fact that many of the combinations (start state, action, and observation triplets) do not lead to any end states. We call such combinations *vanishing combinations*. For the rest of the combinations, called *persisting combinations*, it is usually the case that the number of end states is only a very small fraction of the whole state set. Here, we provide two examples.

The first example, summarized in Table C.1, demonstrates a general reduction pattern observed in our tests. This example is from a POMDP formulation for the TCAS sensor using ±1500 ft/s vertical velocity limits. The processing was done on an Intel Core 2 Duo CPU running at 2.5 GHz, with 4 GB available system memory. Table C.2 contains a histogram showing the frequency of number of end states for the TCAS sensor. When we use the novel belief-state representation and the PPOMDP model, we only iterate over the states with non-zero probabilities in the belief-state, and for each such state, we only consider 5 or 6 end states on average.

Our second example, summarized in Table C.3, demonstrates the efficiency of the PPOMDP formulation. This example is from a POMDP formulation for the perfect sensor using ±1500 ft/s vertical velocity limits. In the POMDP formulation for the hypothetical perfect sensor, the observations are very accurate and they help filter out irrelevant end states very effectively. In a sense, the perfect sensor POMDP model is close to an MDP model. Table C.4 contains a histogram showing the frequency of number of end states for the perfect sensor.

Table C.1: POMDP processing for the TCAS sensor.

Number of states in the POMDP formulation	2886
Number of actions in the POMDP formulation	3
Number of observations in the POMDP formulation	183
Total number of (state-action-observation) combinations	1584414
Number of combinations with no end states (vanish)	1205140
Number of combinations with at least one end state (persist)	379274
Total number of end states reachable from all combinations	8140937
Maximum number of end states reached from a single combination	962
Average number of end states reached from a single combination	5.13814
Time spent computing the PPOMDP formulation (mm:ss)	07:29

Table C.2: Histogram showing the frequency of number of end states for the TCAS sensor.

End states	Frequency	End states	Frequency
0	1205140	28	100
1	17	29	1316
2	1630	30	17850
3	1210	31	582
4	18090	32	11152
5	1048	33	64
6	22500	36	25014
7	82	37	754
8	28480	40	5376
9	6982	41	160
10	16250	43	4
11	896	44	1016
12	54480	45	6564
13	746	46	256
14	320	47	996
15	9550	48	12000
16	22934	49	516
17	1074	54	6560
18	31160	55	480
19	404	64	2442
20	14490	70	16
21	466	71	896
22	120	72	4768
23	80	144	170
24	36784	192	170
25	4644	240	340
26	88	288	170
27	5000	962	17

Table C.3: POMDP processing for the perfect sensor.

Number of states in the POMDP formulation	2886
Number of actions in the POMDP formulation	3
Number of observations in the POMDP formulation	183
Total number of (state-action-observation) combinations	1584414
Number of combinations with no end states (vanish)	1480336
Number of combinations with at least one end state (persist)	104078
Total number of end states reachable from all combinations	581187
Maximum number of end states reached from a single combination	16
Average number of end states reached from a single combination	0.366815
Time spent computing the PPOMDP formulation (mm:ss)	07:11

Table C.4: Histogram showing the frequency of number of end states for the perfect sensor.

End states	Frequency
0	1480336
1	7057
2	2701
3	2500
4	27076
5	256
6	45296
9	18172
16	1020

Appendix D

Encounter Analyzer

The data flow between different modules of our development system can be summarized as follows:

- CASSATT simulations run at 10 Hz. At each simulation step CASSATT computes aircraft state vectors for both ownship and intruder aircraft. Encounter scenarios usually last 50 seconds, therefore, at 10 Hz this corresponds to a total of 1000 aircraft state vectors for a single encounter.

- The software module that implements our collision avoidance algorithms is called the "controller." CASSATT invokes the controller at 1 Hz (at every 10^{th} simulation step). The inputs to the controller are state vectors for both aircraft and an aircraft control command vector for ownship . The control command consists of a vertical rate (or a vertical acceleration), a turn rate, and airspeed acceleration. The aircraft control command vector provided by CASSATT is the scripted maneuver for ownship. It represents the Air Traffic Control (ATC) command that should be followed if there is no danger of collision.

- The output from the controller is another aircraft control command vector. This command might be a replica of the ATC command, or it might be a different one as a result of the planned collision avoidance maneuver by the controller. This output is captured by CASSATT and used to calculate aircraft state vectors for the following 10 simulation steps (before the controller is invoked again).

- At the end of each encounter, CASSATT outputs the horizontal and vertical miss distances (HMD and VMD) at the time of closest approach. A near mid-air collision (NMAC) occurs if both HMD and VMD are below some thresholds (usually if HMD is less than 500 ft and VMD is less than 100 ft).

In addition to the data flow among modules, at each invocation, the controller processes its inputs and generates the internal data that is necessary to compute its output. The internally generated data is as follows:

- A sensor reading is simulated (computed) using the aircraft state vectors for both aircraft.

- For POMDP based systems, the simulated sensor reading is further processed and converted to an "observation" suitable for the POMDP used by the controller.

- The controller keeps a record of the previous collision avoidance action it took. It also maintains an internal belief-state for POMDP systems. At each invocation, this belief-state is updated (using the previous action and the computed observation). Then, using this updated belief-state, the controller decides what collision avoidance action to take next (which determines the output aircraft control command). For path-modification based systems the action is computed at the end of the iterative improvement process.

- Statistical data is collected inside the controller (such as the total and average times it takes to process all data).

It is very difficult to debug such a system using conventional software debugging techniques. An important part of debugging a POMDP during a simulation run is understanding the current state, which can be challenging using conventional tools because the POMDPs used in our tests have 2000–3000 (sometimes even more) states. To aid in debugging, we created a visualization tool to inspect the encounters. Our visualization tool is called "Encounter Analyzer" (or "Analyzer"). Figure D-1 shows a screenshot of the Encounter Analyzer.

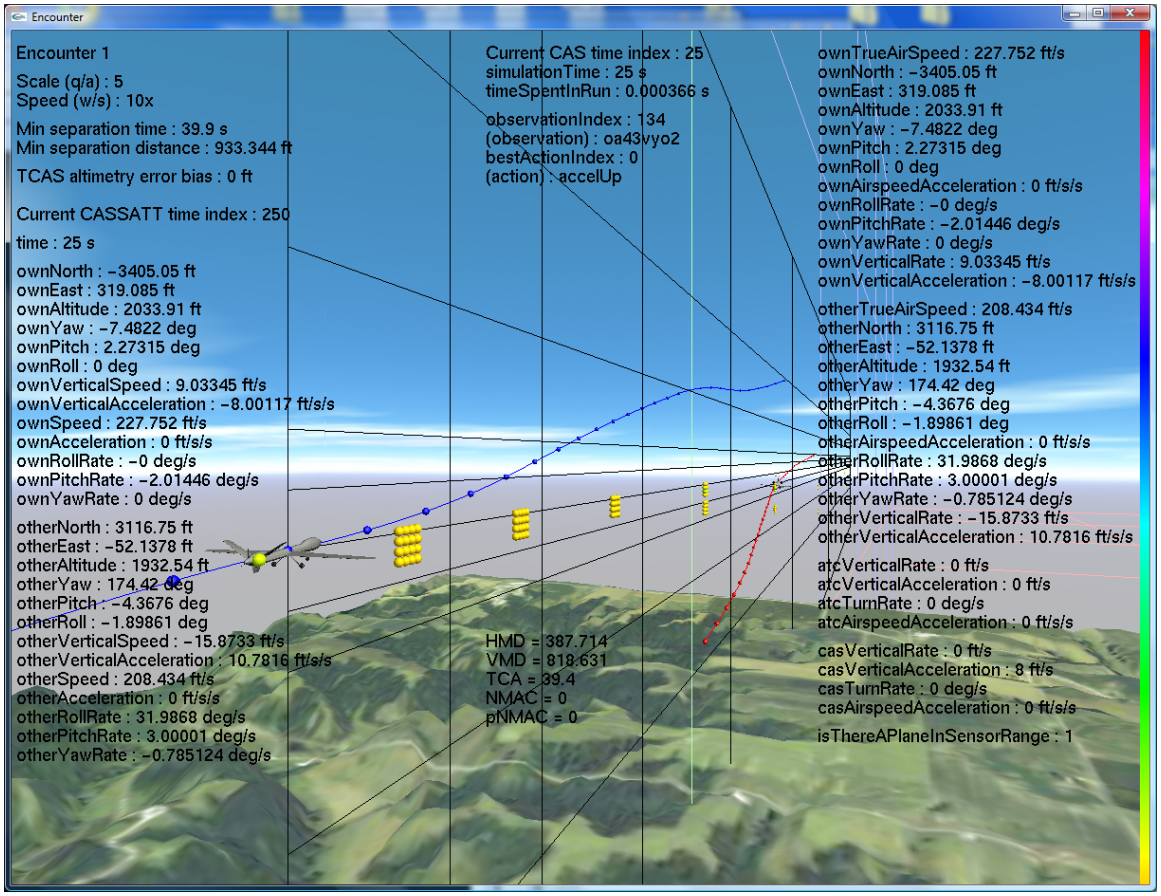


Figure D-1: A screenshot of the Encounter Analyzer showing flight trajectory, belief-state, and debugging information.

The data to be visualized is acquired from two sources:

- At the end of each encounter, CASSATT outputs a special data structure that contains all CASSATT generated data. We prepared a Matlab script that reads in this special data structure, and outputs a text file that Analyzer can parse. The size of this text file is usually around 140–150 KB.

- The controller can be passed an option to turn data logging on or off. If data logging is turned on, the controller dumps all its input, output and internal data to a text file. The only exception is that it does not dump its internal belief-state (since it is large and can be readily recomputed). The size of this text file is usually around 30–40 KB.

Each entry in both of the text files is labeled with the simulation time it belongs to, so Analyzer can synchronize the two sources of data. Analyzer creates its own internal belief-state, and uses the recorded actions and observations to update it, so this belief-state is also synchronized with the actual belief-state used (but not recorded) by the controller.

Analyzer is built on top of the OpenGL library and it has both 3-dimensional graphing and head-up display (HUD) capabilities. The aim of Analyzer is to use graphical visualization (drawn to scale) as much as possible, and use HUD to display the rest of the data that cannot be presented graphically.

The graphical visualizations can be grouped into 3 categories:

1. Visualizations of some of the gathered data

    - Own aircraft (a Predator B model is used)
    - Intruder aircraft (an Embraer Bombardier CRJ-200 model is used)
    - Trajectory of own aircraft for the entire encounter
    - Trajectory of intruder aircraft for the entire encounter
    - The active region for the sensor
    - Sensor reading

2. Other visual enhancements

    - A 3-D coordinate system
    - A tile of terrain
    - A sky-box
    - A bounding box that contains both trajectories
    - Projections of trajectories on the sides of the bounding box
    - A separate window that shows the same (synchronized) visual description as perceived by ownship (there is a small offset that allows our aircraft to be in the view, too)
    - Another separate window that shows the same (synchronized) visual description as perceived by the intruder (there is a small offset that allows the intruder to be in the view, too)

3. Visualizations of data structures and computed variables

    - State space (relative coordinate system)
    - Action space
    - Observation space
    - Belief-state (states are colored, different colors indicate different probabilities, a "color legend" is also displayed on the right side of the main window)
    - Computed observation
    - Computed action

The textual (HUD) visualizations include the following:

- Current simulation time
- Aircraft state vectors for both aircraft (recorded from CASSATT special data structure)

- Aircraft state vectors for both aircraft (recorded by controller)
- ATC command
- Computed observation
- Computed action
- Computed aircraft control command vector
- Quantitative measure produced by CASSATT

Analyzer has a fourth window that displays a list of names of all of the graphical and textual visualizations described above. The user can click any item to toggle its visibility.

Analyzer allows the user to play (forward or backward) a recorded encounter step by step. At any step, one may examine the values of different variables, check the input and output of the controller, and move the camera around in the scene. Additionally, one may visualize the positions and orientations of the aircraft, the sensor readings, and the belief-state.

Bibliography

[1] Bassam Abdul-Baki, Jonathan Baldwin, and Marc-Philippe Rudel. Independent validation and verification of the TCAS II collision avoidance subsystem. In *AIAA* 18^{th} *Annual Digital Avionics Systems Conference*, 1999. (Cited on page 16)

[2] P. Edgar An and Chris J. Harris. An intelligent driver warning system for vehicle collision avoidance. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 26(2):254–261, March 1996. (Cited on page 15)

[3] K. J. Astrom. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965. (Cited on page 20)

[4] J. Andrew Bagnell and Jeff C. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *International Conference on Robotics and Automation*, pages 1615–1620. IEEE Press, 2001. (Cited on page 26)

[5] Tucker Balch and Ronald Craig Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998. (Cited on page 32)

[6] Cesar Bandera, Francisco J. Vico, Jose M. Bravo, Mance E. Harmon, and Leemon C. Baird III. Residual Q-Learning applied to visual attention. In 13^{th} *International Conference on Machine Learning*, pages 20–27, 1996. (Cited on page 26)

[7] Jerome Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, December 1991. (Cited on page 32)

[8] Jennifer Barry, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Hierarchical solution of large Markov decision processes. In *ICAPS Workshop on Planning and Scheduling in Uncertain Domains*, May 2010. (Cited on page 30)

[9] Olivier Baud, Nicolas Honore, and Olivier Taupin. Radar/ADS-B data fusion architecture for experimentation purpose. In 9^{th} *International Conference on Information Fusion*, pages 1–6, July 2006. (Cited on page 16)

[10] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35:407–427, 1999. (Cited on page 35)

[11] Eva Besada-Portas, Luis de la Torre, Jesus M. de la Cruz, and Bonifacio de Andrés-Toro. Evolutionary trajectory planner for multiple UAVs in realistic scenarios. *IEEE Transactions on Robotics*, 26:619–634, August 2010. (Cited on page 36)

[12] John T. Betts. *Practical methods for optimal control using nonlinear programming*, volume 3 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001. (Cited on page 30)

[13] Gollee H. Bevan, G. and J. O'Reilly. Automatic lateral emergency collision avoidance for a passenger car. *International Journal of Control*, 80(11):1751–1762, 2007. (Cited on page 15)

[14] Karl D. Bilimoria. A geometric optimization approach to aircraft conflict resolution. August 2000. (Cited on page 33)

[15] Thomas Billingsley. Safety analysis of TCAS on Global Hawk using airspace encounter models. Master's thesis, Massachusetts Institute of Technology, 2006. (Cited on page 42)

[16] P. H. Borcherds. Importance sampling: an illustrative introduction. *European Journal of Physics*, 21(5):405–411, 2000. (Cited on page 49)

[17] H. Boucard, T. Gach, and E. Sicsik-Pare. CRECUS: A radar sensor for battlefield surveillance UAVs. In *Proceedings of RTO SCI Symposium on Warfare Automation: Procedures and Techniques for Unmanned Vehicles*, Ankara, Turkey, April 1999. (Cited on page 16)

[18] Craig Boutilier. A POMDP formulation of preference elicitation problems. In *Proceedings of the* 18^{th} *National Conference on Artificial Intelligence*, pages 239–246, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. (Cited on page 26)

[19] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial (*2^{nd} *ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. (Cited on page 100)

[20] Emma Brunskill, Leslie Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Continuous-state POMDPs with hybrid dynamics. In 10^{th} *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, January 2008. (Cited on page 30)

[21] Emma Brunskill, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Planning in partially-observable switching-mode continuous domains. *Annals of Mathematics and Artificial Intelligence*, 58:185–216, April 2010. (Cited on page 30)

[22] Robert L. Bryant. Geometry of manifolds with special holonomy: "100 years of holonomy". In *150 Years of Mathematics at Washington University in St. Louis*, volume 395 of *Contemporary Mathematics*, pages 29–38. American Mathematical Society, Providence, RI, 2006. (Cited on page 18)

[23] Anthony R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Providence, Rhode Island, 1998. (Cited on page 20)

[24] Anthony R. Cassandra. A survey of POMDP applications, 1998. Presented at the AAAI Fall Symposium. (Cited on page 26)

[25] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the* 12^{th} *National Conference on Artificial Intelligence*, Seattle, WA, 1994. (Cited on page 20)

[26] Daniel Challou, Daniel Boley, Maria Gini, and Vipin Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *IEEE International Conference on Robotics and Automation*, pages 709–714, 1995. (Cited on page 32)

[27] Roxaneh Chamlou. Future airborne collision avoidance - design principles, analysis plan and algorithm development. In 28^{th} *Digital Avionics Systems Conference*, Orlando, Florida, October 2009. (Cited on page 86)

[28] Samer Charifa and Marwan Bikdash. Comparison of geometrical, kinematic, and dynamic performance of several potential field methods. In *IEEE SOUTHEASTCON'09*, pages 18–23, March 2009. (Cited on page 32)

[29] X.D. Cheng, Z.Y. Liu, and X.T. Zhang. Trajectory optimization for ship collision avoidance system using genetic algorithm. In *OCEANS 2006 - Asia Pacific*, pages 1–5, May 2006. (Cited on page 15)

[30] B. Chludzinski. Lincoln Laboratory evaluation of TCAS II Logic Version 7. Project Report ATC-268, 1999. (Cited on page 57)

[31] Trevor Darrell and Alex Pentland. Active gesture recognition using partially observable Markov decision processes. In *ICPR96*, pages 984–988, 1996. (Cited on page 26)

[32] Mark Denny. Introduction to importance sampling in rare-event simulations. *European Journal of Physics*, 22(4):403–411, 2001. (Cited on page 49)

[33] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, November 1993. (Cited on page 88)

[34] Bruce R. Donald and Patrick Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, 14:958–963, 1995. (Cited on page 88)

[35] A. Drumm. Lincoln Laboratory evaluation of TCAS II Logic Version 6.04a. Project Report ATC-240, 1996. (Cited on page 57)

[36] Vu.N. Duong and K. Zeghal. Conflict resolution advisory for autonomous airborne separation in low-density airspace. In *Proceedings of the* 36^{th} *IEEE Conference on Decision and Control*, volume 3, pages 2429–2434, December 1997. (Cited on page 31)

[37] M.S. Eby and W.E. Kelly. Free flight separation assurance using distributed algorithms. In *IEEE Aerospace Conference*, volume 2, pages 429–441, 1999. (Cited on page 31)

[38] A. Engelsone, S. L. Campbell, and J. T. Betts. Direct transcription solution of higher-index optimal control problems and the virtual index. *Applied Numerical Mathematics*, 57:281–296, March 2007. (Cited on page 30)

[39] Tom Erez and William D. Smart. A scalable method for solving high-dimensional continuous POMDPs using local approximation. In P. Grunwald and P. Spirtes, editors, *Proceedings of the* 26^{th} *Conference on Uncertainty in Artificial Intelligence (UAI 2010)*. AUAI Press, 2010. (Cited on page 30)

[40] U.S. Air Force. Air combat command manual 3-3, 1992. Department of the Air Force, Washington, D.C. (Cited on page 32)

[41] Kikuo Fujimura. *Motion Planning in Dynamic Environments*, chapter 2, pages 9–26. Springer-Verlag, 1991. (Cited on page 36)

[42] Jiading Gai, Yong Li, and R.L. Stevenson. Coupled hidden Markov models for robust EO/IR target tracking. In *IEEE International Conference on Image Processing (ICIP)*, pages 41–44, October 2007. (Cited on page 16)

[43] Carlos E. Garcia, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice–a survey. *Automatica*, 25(3):335–348, 1989. (Cited on page 36)

[44] Zoubin Ghahramani. Learning dynamic Bayesian networks. In C.L. Giles and M. Gori, editors, *Adaptive Processing of Sequences and Data Structures, Lecture Notes in Artificial Intelligence*, volume 1387, pages 168–197. Springer-Verlag, 1998. (Cited on page 45)

[45] Dowek Gilles, Munoz Cesar, and Geser Alfons. Tactical conflict detection and resolution in a 3-D airspace. Technical report, Santa Fe, New Mexico, 2001. (Cited on page 33)

[46] Roger L. Gray. Method of Kalman filtering for estimating the position and velocity of a tracked object. U.S. Patent No. 5,051,751. Filed February 12, 1991, and issued September 24, 1991. (Cited on page 68)

[47] Ashwin Gumaste, Rahul Singhai, and Anirudh Sahoo. Intellicarts: Intelligent car transportation system. In 15^{th} *IEEE Local Area Networks and Metro Area Networks (LANMAN)*, Princeton, NJ, 2007. (Cited on page 15)

[48] Eric A. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the* 14^{th} *Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 211–219, San Francisco, CA, 1998. Morgan Kaufmann. (Cited on page 28)

[49] Milos Hauskrecht. *Planning and Control in Stochastic Domains with Imperfect Information*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1997. (Cited on page 26)

[50] Milos Hauskrecht. *Planning and Control in Stochastic Domains with Imperfect Information*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1998. (Cited on page 20)

[51] Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000. (Cited on page 30)

[52] Milos Hauskrecht, Nicolas Meuleau, Craig Boutilier, Leslie Pack Kaelbling, and Thomas Dean. Hierarchical solution of Markov decision processes using macroactions. In *Proceedings of the* 14^{th} *Annual Conference on Uncertainty in Artificial Intelligence*, Madison, Wisconsin, 1998. (Cited on page 30)

[53] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, July 1998. (Cited on page 34)

[54] John Hertz, Richard G. Palmer, and Anders S. Krogh. *Introduction to the Theory of Neural Computation*. Perseus Publishing, 1991. (Cited on page 34)

[55] Eric Horvitz and Tim Paek. Deeplistener: Harnessing expected utility to guide clarification dialog in spoken language systems. In *Proceedings of the* 6^{th} *International Conference on Spoken Language Processing (ICSLP 2000)*, volume 1, pages 226–229, Beijing, China, October 2000. (Cited on page 26)

[56] Ronald A. Howard. *Dynamic Programming and Markov Processes*. The MIT Press, Cambridge, Massachusetts, 1960. (Cited on page 20)

[57] D. Hsu, W.S. Lee, and N. Rong. What makes some POMDP problems easy to approximate? In *Advances in Neural Information Processing Systems (NIPS)*, 2007. (Cited on page 28)

[58] D. Hsu, W.S. Lee, and N. Rong. A point-based POMDP planner for target tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2644–2650, 2008. (Cited on page 28)

[59] David Hsu, J.C. Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9(3):495–512, 1997. (Cited on page 32)

[60] International Civil Aviation Organization. Surveillance, radar and collision avoidance. In *International Standards and Recommended Practices*, volume IV, annex 10. 4th edition, July 2007. (Cited on page 75)

[61] D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming*. Elsevier, 1970. (Cited on page 30)

[62] Seungwon Jang and Joongwook Kim. Survey of electro-optical infrared sensor for UAV. *Aerospace Industry and Technology Trends*, 6(1):122–132, July 2008. (Cited on page 16)

[63] James J. Kuffner Jr. and Steven M. Lavalle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 995–1001, 2000. (Cited on page 32)

[64] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998. (Cited on pages 20 and 25)

[65] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Finding aircraft collision-avoidance strategies using policy search methods. Technical Report MIT-CSAIL-TR-2009-043, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2009. (Cited on page 34)

[66] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation*, pages 566–580, 1996. (Cited on page 32)

[67] Oussama Khatib. The potential field approach and operational space formulation in robot control. In *Proceedings of the 4^{th} Yale Workshop on Applications of Adaptive Systems Theory*, pages 208–214, Yale University, New Haven, CT, USA, May 1985. (Cited on page 31)

[68] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 500–505, St. Louis, MO, USA, March 1985. (Cited on page 31)

[69] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986. (Cited on page 31)

[70] Oussama Khatib and J. F. Le Maitre. Dynamic control of manipulators operating in a complex environment. In *Proceedings of the* 3^{rd} *CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, pages 267–282, Udine, Italy, September 1978. (Cited on page 31)

[71] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. A decision-theoretic approach to developing robust collision avoidance logic. In *IEEE International Conference on Intelligent Transportation Systems*, Madeira Island, Portugal, 2010. (Cited on page 31)

[72] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. Partially-controlled Markov decision processes for collision avoidance systems. In *International Conference on Agents and Artificial Intelligence*, Rome, Italy, 2011. (Cited on page 83)

[73] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. Robust airborne collision avoidance through dynamic programming. Project Report ATC-371, Massachusetts Institute of Technology, Lincoln Laboratory, 2011. (Cited on page 31)

[74] Mykel J. Kochenderfer, James P. Chryssanthacopoulos, Leslie P. Kaelbling, Tomás Lozano-Pérez, and James K. Kuchar. Model-based optimization of airborne collision avoidance logic. Project Report ATC-360, Massachusetts Institute of Technology, Lincoln Laboratory, 2010. (Cited on page 31)

[75] Mykel J. Kochenderfer, Leo P. Espindle, James K. Kuchar, and J. Daniel Griffith. Correlated encounter model for cooperative aircraft in the national airspace system. Project Report ATC-344, 2008. (Cited on pages 42, 45, and 48)

[76] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1398–1404, April 1991. (Cited on page 31)

[77] James K. Kuchar. *A Unified Methodology for the Evaluation of Hazard Alerting Systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, January 1995. (Cited on pages 36 and 61)

[78] James K. Kuchar. Methodology for alerting-system performance evaluation. *Journal of Guidance, Control, and Dynamics*, 19(2):438–444, 1996. (Cited on page 61)

[79] James K. Kuchar and Lee C. Yang. A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems*, 1(4):179–189, 2000. (Cited on pages 34, 36, and 86)

[80] H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems*, 2008. (Cited on pages 22 and 28)

[81] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J.P. How, and G. Fiore. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, September 2009. (Cited on page 32)

[82] Chieh-Ping Lai, Yu-Jiun Ren, and Chujen Lin. ADS-B based collision avoidance radar for unmanned aerial vehicles. In *Proceedings of IEEE MTT-S International*, pages 85–88, June 2009. (Cited on page 16)

[83] Terran Lane and Leslie Pack Kaelbling. Toward hierarchical decomposition for planning in uncertain environments. In *Proceedings of the 2001 IJCAI Workshop on Planning under Uncertainty and Incomplete Information*, pages 1–7, 2001. (Cited on page 30)

[84] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991. (Cited on page 31)

[85] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001. (Cited on page 32)

[86] S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing Voronoi bias in RRTs. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004. (Cited on page 33)

[87] S. R. Lindemann and S. M. LaValle. Steps toward derandomizing RRTs. In *IEEE* 4^{th} *International Workshop on Robot Motion and Control*, 2004. (Cited on page 33)

[88] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, pages 362–370, 1995. (Cited on pages 30 and 68)

[89] Jun Liu, Hong Wei, Xi-Yue Huang, Nai-Shuai He, and Ke Li. A bridge-ship collision avoidance system based on FLIR image sequences. In Sio-Iong Ao and Len Gelman, editors, *Advances in Electrical Engineering and Computational Science*, volume 39 of *Lecture Notes in Electrical Engineering*, pages 123–133. Springer Netherlands, 2009. (Cited on page 15)

[90] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, February 1983. (Cited on page 32)

[91] Tomás Lozano-Pérez. A simple motion planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, RA-3(3):224–238, June 1987. (Cited on page 32)

[92] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979. (Cited on page 32)

[93] Brandon Luders. Robust trajectory planning for unmanned aerial vehicles in uncertain environments. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, September 2008. (Cited on page 35)

[94] Emmanuel Mazer, Juan Manuel Ahuactzin, El-Ghazali Talbi, Pierre Bessiere, and T. Chatroux. Parallel motion planning with the Ariadne's Clew algorithm. In Tsuneo Yoshikawa and Fumio Miyazaki, editors, *ISER*, volume 200 of *Lecture Notes in Control and Information Sciences*, pages 62–74. Springer, 1993. (Cited on page 32)

[95] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dayan. Interacting multiple model methods in target tracking: A survey. *IEEE Transactions on Aerospace Electronic Systems*, 34:103–123, January 1998. (Cited on page 68)

[96] Michael P. McLaughlin. Safety study of the Traffic Alert and Collision Avoidance System (TCAS II). Technical Report MTR 97W32, MITRE Corporation, June 1997. (Cited on page 57)

[97] Christoph Mertz, Sue McNeil, and Chuck Thorpe. Side collision warning systems for transit buses. In *IV 2000, IEEE Intelligent Vehicle Symposium*, October 2000. (Cited on page 15)

[98] Nicolas Meuleau, Kee-Eung Kim, Leslie Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Proceedings of the* 15^{th} *Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 417–426, San Francisco, CA, 1999. Morgan Kaufmann. (Cited on page 28)

[99] MITRE. System safety study of minimum TCAS II. Technical Report MTR-83W241, MITRE, 1983. (Cited on page 57)

[100] M. Mukai, Y. Harada, J. Murata, T. Kawabe, H. Nishira, and Y. Deguchi. An automotive collision avoidance control based on a feasible set. In *ICCAS-SICE*, pages 2164–2168, August 2009. (Cited on page 15)

[101] Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 2002. (Cited on page 45)

[102] Andrew Ng and Michael Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the* 16^{th} *Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000. (Cited on page 34)

[103] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In *NIPS*, 2003. (Cited on page 26)

[104] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2004. (Cited on page 34)

[105] Sébastien Paquet, Ludovic Tobin, and Brahim Chaib-draa. Real-time decision making for large POMDPs. In Balázs Kégl and Guy Lapalme, editors, 18^{th} *Canadian Conference on Artificial Intelligence*, volume 3501 of *Lecture Notes in Computer Science*, pages 450–455. Springer, 2005. (Cited on page 31)

[106] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025–1032, August 2003. (Cited on page 28)

[107] Joelle Pineau and Sebastian Thrun. High-level robot behavior control using POMDPs. In *AAAI Workshop Notes*, 2002. (Cited on page 26)

[108] Efstratios N. Pistikopoulos. C.a. floudas, nonlinear and mixed-integer optimization. fundamentals and applications. *Journal of Global Optimization*, 12:108–110, 1998. (Cited on page 35)

[109] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Pérez. Belief space planning assuming maximum likelihood observations. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010. (Cited on page 30)

[110] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994. (Cited on pages 20 and 26)

[111] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In *Proceedings of the International Conference on Robotics and Automation*, pages 802–807, 1993. (Cited on page 94)

[112] A. Raghunathan, V. Gopal, D. Subramanian, L.T. Biegler, and T. Samad. Dynamic optimization strategies for 3D conflict resolution of multiple aircraft. *AIAA Journal of Guidance, Control, and Dynamics*, 27(4):586–594, August 2004. (Cited on page 36)

[113] Bharanee Rathnasabapathy, Prashant Doshi, and Piotr Gmytrasiewicz. Exact solutions of interactive POMDPs using behavioral equivalence. In *Proceedings of the* 5^{th} *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1025–1032, New York, NY, USA, 2006. ACM. (Cited on page 28)

[114] Paul Stephen Rau. Drowsy driver detection and warning system for commercial vehicle drivers: Field operational test design, data analyses, and progress. In *Proceedings of* 19^{th} *International Technical Conference on the Enhanced Safety of Vehicles*, June 2005. (Cited on page 15)

[115] Arthur Richards and Jonathan P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the American Control Conference*, volume 3, pages 1936–1941, 2002. (Cited on page 35)

[116] Cristina Rico Garcia, Andreas Lehner, Thomas Strang, and Matthias Roeckl. Comparison of collision avoidance systems and applicability to rail transport. In Masayuki; Bonnet Christian; Lenardi Massimiliano; Komaki Shozo; Wen GuangJun Finger, Ulrich; Fujise, editor, *ITST 2007*, pages 521–526, Sophia Antipolis, France, June 2007. Imprimerie Pons. (Cited on page 15)

[117] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, 2008. (Cited on page 29)

[118] N. Roy, J. Pineau, and S. Thrun. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the* 38^{th} *Annual Meeting of the Association for Computational Linguistics (ACL2000)*, Hong Kong, 2000. (Cited on page 26)

[119] RTCA, Inc. Minimum operational performance standards for traffic alert and collision avoidance system II (TCAS II) airborne equipment. Technical Report RTCA/DO-185A, SC-147, Washington, DC, December 1997. (Cited on pages 16, 40, and 49)

[120] RTCA, Inc. Minimum aviation system performance standards for automatic dependent surveillance broadcast (ADS-B). Technical Report RTCA/DO-242A, Washington, DC, June 2002. (Cited on page 16)

[121] RTCA, Inc. Safety analysis of proposed change to TCAS RA reversal logic. Technical Report RTCA/DO-298, Washington, DC, November 2005. (Cited on page 42)

[122] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*, chapter 17, pages 613–648. Prentice Hall, 2003. (Cited on pages 20 and 61)

[123] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995. (Cited on page 26)

[124] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973. (Cited on pages 20 and 28)

[125] Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the* 20^{th} *Conference on Uncertainty in Artificial Intelligence*, pages 520–527, Arlington, Virginia, United States, 2004. AUAI Press. (Cited on pages 22 and 28)

[126] Trey Smith and Reid G. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005. (Cited on page 28)

[127] Edward J. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University, Stanford, California, 1971. (Cited on page 28)

[128] Edward J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978. (Cited on pages 20 and 28)

[129] Hisashi Tanizaki. *Nonlinear Filters: Estimation and Applications*. Springer-Verlag, second (revised and enlarged) edition, 1996. (Cited on page 68)

[130] Russ Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009. (Cited on pages 30 and 88)

[131] Selim Temizer, Mykel John Kochenderfer, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and James K. Kuchar. Unmanned aircraft collision avoidance using partially observable Markov decision processes. Project Report ATC-356, MIT Lincoln Laboratory, Advanced Concepts Program, Lexington, Massachusetts, USA, September 2009. (Cited on pages 46 and 83)

[132] Selim Temizer, Mykel John Kochenderfer, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and James K. Kuchar. Collision avoidance for unmanned aircraft using Markov decision processes. In *Proceedings of the American Institute of Aeronautics and Astronautics (AIAA) Guidance, Navigation, and Control Conference*, Sheraton Centre Toronto, Toronto, Ontario, Canada, August 2010. American Institute of Aeronautics and Astronautics. (Cited on page 46)

[133] Sylvie Thiébaux, Marie-Odile Cordier, Olivier Jehl, and Jean-Paul Krivine. Supply restoration in power distribution systems - a case study in integrating model-based diagnosis and repair planning. In *Proceedings of the* 12^{th} *Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 525–532, 1996. (Cited on page 26)

[134] Chuck Thorpe, David Duggins, Sue McNeil, and Christoph Mertz. Side collision warning system (SCWS) performance specifications for a transit bus. In *Final report, prepared for the Federal Transit Administration under PennDOT agreement number 62N111, Project TA-34*. May 2002. (Cited on page 15)

[135] Sebastian Thrun. Monte Carlo POMDPs. In S.A. Solla, T.K. Leen, and K.R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1064–1070. MIT Press, 2000. (Cited on page 31)

[136] Louis Tijerina, Angela Ho, Mary L. Cummings, Dev S. Kochhar, and Enlie Wang. Integrating intelligent driver warning systems: Effects of multiple alarms and distraction on driver performance. In *Transportation Research Board* 85^{th} *Annual Meeting*, January 2006. (Cited on page 15)

[137] Shiu Ming Tsang. Method and apparatus for performing three-dimensional alpha/beta tracking. U.S. Patent No. 6,236,899 B1. Filed August 18, 1998, and issued May 22, 2001. (Cited on page 68)

[138] Li-Chun Tommy Wang and Chih Cheng Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):489–499, August 1991. (Cited on page 101)

[139] H. Paul Williams and Sally C. Brailsford. *Computational logic and integer programming*, pages 249–281. Oxford University Press, Inc., New York, NY, USA, 1996. (Cited on page 35)

[140] R.E. Williams. Importance sampling applied to policy gradient for avoidance of rare events. Master's thesis, Boston University, 2009. (Cited on page 34)

[141] Lee F. Winder and James K. Kuchar. Evaluation of collision avoidance maneuvers for parallel approach. *Journal of Guidance, Control, and Dynamics*, 22(6):801–807, 1999. (Cited on page 61)

[142] Travis B. Wolf and Mykel J. Kochenderfer. Aircraft collision avoidance using Monte Carlo real-time belief space search (in press). *Journal of Intelligent and Robotic Systems*, 2011. (Cited on page 30)

[143] Travis Benjamin Wolf. Aircraft collision avoidance using Monte Carlo real-time belief space search. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, May 2009. (Cited on page 30)