METHODOLOGY FOR COLLISION RISK ASSESSMENT OF AIRCRAFT WITH DIVERSE COLLISION AVOIDANCE CAPABILITIES

by

Seungwon Noh

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Systems Engineering and Operations Research

Committee:

_____	Dr. John Shortle, Dissertation Director
_____	Dr. George Donohue, Committee Member
_____	Dr. Lance Sherry, Committee Member
_____	Dr. Duminda Wijesekera, Committee Member
_____	Dr. John Shortle, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date:_____ Summer Semester 2020
George Mason University
Fairfax, VA

Methodology for Collision Risk Assessment of Aircraft with Diverse Collision Avoidance Capabilities

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Seungwon Noh
Master of Science
George Mason University, 2013
Bachelor of Science
Korea Aerospace University, 2005

Director: John Shortle, Professor
Department of Systems Engineering and Operations Research

Summer Semester 2020
George Mason University
Fairfax, VA

DEDICATION

This is dedicated to my parents, my wife and my children.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere thanks to my advisor, Dr. John Shortle, for all the guidance, encouragement and persistent help he provided during my study at George Mason University. This dissertation would not have been possible without his support.

I would like to thank my committee members, Dr. Lance Sherry, Dr. George Donohue and Dr. Duminda Wijesekera, for their invaluable advices and suggestions to enhance my research and for all the support at Center for Air Transportation Systems Research.

I would like to thank all my colleagues, Zhenming, Azin, Sasha and Chaitanya, at Center for Air Transportation Systems Research. I would also like to thank all my friends, Jungho, James, Cheolyoung, Wook and Byunghwa at GMU and Kyeongsu, Taeho, Hyeonmi and Woon from KAU.

Finally and most importantly, I want to thank my family for their endless and unconditional support and encouragement during my graduate study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

METHODOLOGY FOR COLLISION RISK ASSESSMENT OF AIRCRAFT WITH DIVERSE COLLISION AVOIDANCE CAPABILITIES

Seungwon Noh, Ph.D.

George Mason University, 2020

Dissertation Director: Dr. John Shortle

This dissertation proposes a general dynamic event tree (DET) framework and evaluation methodology to assess collision risk for a variety of aircraft types and collision avoidance capabilities. The proposed DET framework consists of three levels – a high-level dynamic event tree that models multiple conflict detection and resolution (CD&R) systems that operate in a sequence to prevent a collision, a generic sub-tree modelling more specific sequences of events within each CD&R phase to resolve a conflict, and fault trees which model the component-based failure logic of each CD&R system. A solution approach is proposed combining analysis methodologies for dynamic event trees, phased-mission systems, and binary decision diagrams. The approach captures several different behaviors influencing collision risk such as time-varying conflict detection rates, pilot delays, component failures, and conflict geometry. The approach allows for ease of creating and modifying a model as well as quick evaluation.

To illustrate the methodology, case studies are developed for collision risk between various types of aircraft with different collision avoidance capabilities in a hypothetical future airspace, e.g., Autonomous Flight Rules (AFR) and the Advanced Airspace Concept (AAC). In addition, sensitivity analyses on the model parameters including component failure probabilities, detection range of the sensors, and error rates of the CD&R systems are conducted.

Case studies indicate that the reliability of aircraft transponders significantly drives collision risk since the CD&R systems and concepts considered highly rely on the transponders for surveillance. In addition, integrating unmanned aircraft with a limited CD&R system into the airspace would increase collision risk significantly.

CHAPTER 1: INTRODUCTION

1.1 Background

Air transportation passenger demand in the U.S. is forecasted to increase by 1.9 percent annually for the next 20 years [FAA 2018]. In order to accommodate the increasing demand and provide safer, more efficient and predictable air transportation service, the Federal Aviation Administration (FAA) has been implementing the Next Generation Air Transportation System (NextGen), which is the modernization of the air transportation system introducing several new technologies through 2025 and beyond [FAA 2018]. The system will need to accommodate a large growth in Unmanned Aircraft Systems (UAS) as well as commercial spacecraft eager to access the National Airspace System (NAS). Wieland [2016] estimated a demand of over 25,000 UAS flights per day in the NAS (above 2,000 feet above ground level).

In addition to the growth in the number of flights, the diversity of aircraft types in the NAS will also increase significantly. Various types of Unmanned Aerial Vehicles (UAVs) have a wide range of specifications, such as dimension and weight, and performance characteristics, such as cruise speed and maximum operating altitude that can differ significantly from manned aircraft. They may also have different collision avoidance technologies. Since UAVs have no pilot on board, various sensors (e.g., optical, thermal, or laser) have been proposed to detect and avoid nearby aircraft.

Furthermore, UAVs are intended to conduct completely different missions from those of current commercial aircraft. Monitoring air quality, weather data collection, and tactical fighting of wildfires are common examples of UAV missions. Some of the missions require flying continuously over a certain area so that capacity of that airspace is affected critically.

1.2 Motivation

The air transportation system currently provides an extremely safe mode of transportation. According to Lin [2009], the actual level of safety for fatal mid-air collision risk in 47 years (1959-2006) was 2.17×10^{-8} per flight hour. Integrated Safety Assessment Model (ISAM) developed by FAA provides a similar rate of mid-air-collision accidents in an order of 10^{-8} per flight hour (i.e., one mid-air-collision accident in 100 million flights given an assumption of 2-hour flying time on average). Maintaining enough separation between aircraft is the key to avoiding mid-air collisions and achieving a high level of safety. However, it is also a constraint to increasing throughput of the airspace. Intuitively, there is a trade-off between safety and capacity – placing more aircraft in a given region of airspace reduces the level of safety within that region.

The capacity within a sector of the NAS is currently governed by the Monitor Alert Parameter (MAP), which specifies a numerical trigger value for the maximum number of aircraft that should be in a sector [FAA 2019]. The MAP value is sector-specific, depending on factors such as the average time for aircraft to traverse the sector and the average time required for a controller to manage each aircraft. With increasing

demand, the sector capacity may be a significant constraint on the capacity of the airspace.

As the system evolves, new procedures and technologies have the potential to increase system capacity, but also to alter the level of safety. For example, Erzberger [2001] proposes the Advanced Airspace Concept (AAC), in which a ground-based system automatically detects conflicts and provides automated resolutions to properly equipped aircraft. AAC eliminates manual separation monitoring and control, which can increase airspace capacity, but may also change the level of safety.

The introduction of UAVs can also impact the safety of the system. For example, since there is no pilot on board, the pilot's see and avoid procedure in 14 CFR Part 91.113 (right-of-way rules) is not applicable to UAVs, which can significantly impact the level of safety. Naturally, such changes must be rigorously evaluated from a safety standpoint prior to being implemented.

A number of analyses have been conducted assessing the collision risk for these new procedures and technologies. For example, several papers have examined the collision risk associated with AAC using a variety of different analysis methodologies, e.g., fault trees [Andrews 2005], Monte Carlo simulation [Blum 2010], and dynamic event trees [Shortle 2012]. The free flight concept, in which the flight crew has the freedom to select a trajectory and the responsibility to resolve a conflict with other aircraft, has been analyzed in Blom [2006]. Collision risk for the flow corridor concept, which is a NextGen concept to better accommodate high levels of traffic, is considered in Zhang [2015]. A common limitation of these papers is that they consider only current

commercial aircraft operating in a system similar to today's NAS and assume that all aircraft have the same level of conflict detection and resolution (CD&R) equipage.

Similarly, a number of analyses have been conducted to evaluate collision risk for UAVs. These have been conducted in terms of technology, concept of operations, conflict detection and resolution algorithms, and so forth [Kuchar 2004; Muñoz 2015; Ferreira 2018; Jenie 2018]. Most papers focus on evaluating how successfully a UAV's collision avoidance technology can detect a collision with a manned aircraft and how reliably it can perform a collision avoidance maneuver. The results of these papers are useful for a given technology or aircraft type, but may not readily extend to other aircraft or technologies. In particular, if there are n different aircraft types in a region of airspace, then there must be n^2 separate safety analyses to certify that each pair is able to safely operate together.

Many of these analyses take similar approaches, but they are developed separately in a problem-specific manner, so there is an opportunity to generalize some of the methods under a common unified framework.

1.3 Research Objective

The objective of this research is to propose a general framework and methodology to assess collision risk for an airspace with a variety of aircraft types and collision avoidance capabilities. The methodology accounts for the inaccuracies of the collision avoidance algorithms and trajectory prediction capabilities as well as discrete failures in system elements, considering both human and hardware failures. Several case studies involving diverse kinds of aircraft and/or future automated conflict detection and

resolution systems are given to illustrate the methodology. In addition, sensitivity analysis on the model parameters for the case studies is conducted to identify parameters that significantly impact collision risk and to provide insights into improving safety.

Collision risk between aircraft can be divided into two parts (Figure 1): 1) the risk that two aircraft are on a collision course, 2) the risk that the collision avoidance systems fail to resolve the conflict. The former risk mostly depends on the number of aircraft in a region of airspace, while the latter risk depends on aircraft equipage. This research focuses only on the second component– namely, the probability that the collision avoidance systems fail, given that two aircraft are already on a collision path. Therefore, a term of collision risk/probability used in this dissertation is limited to the risk/probability that the collision avoidance systems fail, given two aircraft on a collision course.

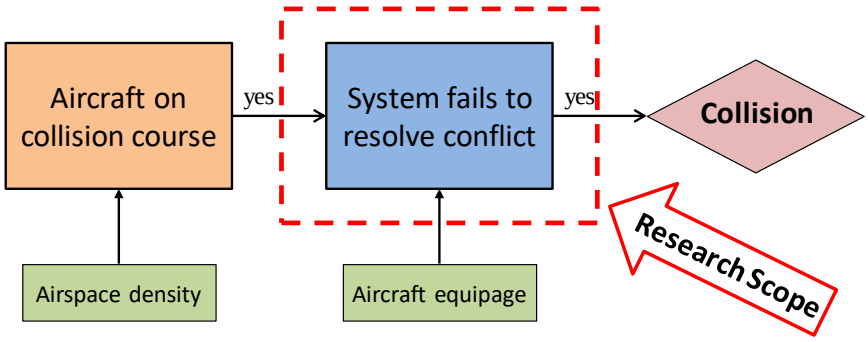


Figure 1 Comparison of collision risk models

Major questions that this research would answer are as follows:

- What existing methods can be applied for evaluating collision risk of an airspace?
- Can a systematic approach be developed to evaluate collision risk for an airspace with a variety of aircraft types and collision avoidance capabilities?
- How well do methods perform for estimating collision risk in terms of speed and accuracy?
- What is the change in collision risk when various aircraft types (with different collision avoidance capabilities) are allowed to operate at the same time?

Given limited general approach for evaluating collision risk between diverse aircraft types and/or collision avoidance capabilities, the unique contributions of this research are as follows:

- Proposal of a general framework using dynamic event tree (DET) structure to model collision risk between various aircraft types and/or collision avoidance capabilities.
- Suggestion and comparison of several methods that can be used to evaluate the proposed DET framework.
- Development of several case studies using the framework, where collision risk between different types of aircraft with different collision avoidance capabilities in a hypothetical future NAS operations.

1.4 Potential Applications of Research

This research proposes a general approach to evaluate collision risk between various types of aircraft with different collision avoidance capabilities. The primary

application concept of the approach is to support the Safety Management System (SMS) of the FAA, which is a framework to identify, analyze, assess, manage, and monitor safety risk of the NAS [FAA 2019]. The approach of this research is particularly able to support to analyze risk of any NAS change in terms of operation, procedure, or equipment (i.e., hardware and software) in the process of the Safety Risk Management (SRM) and Safety Assurance of the SMS (top of Figure 2). More specifically, the method supports to define risk by determining likelihood of potential harmful effects of a hazard during the risk analysis process. The identified controls are used to model a risk through the approach of the this research. For example, Air traffic controller (ATC) and Traffic Alert and Collision Avoidance System (TCAS), which are something to prevent mid-air-collision accident, are used as safety layers to reduce collision risk in the method.

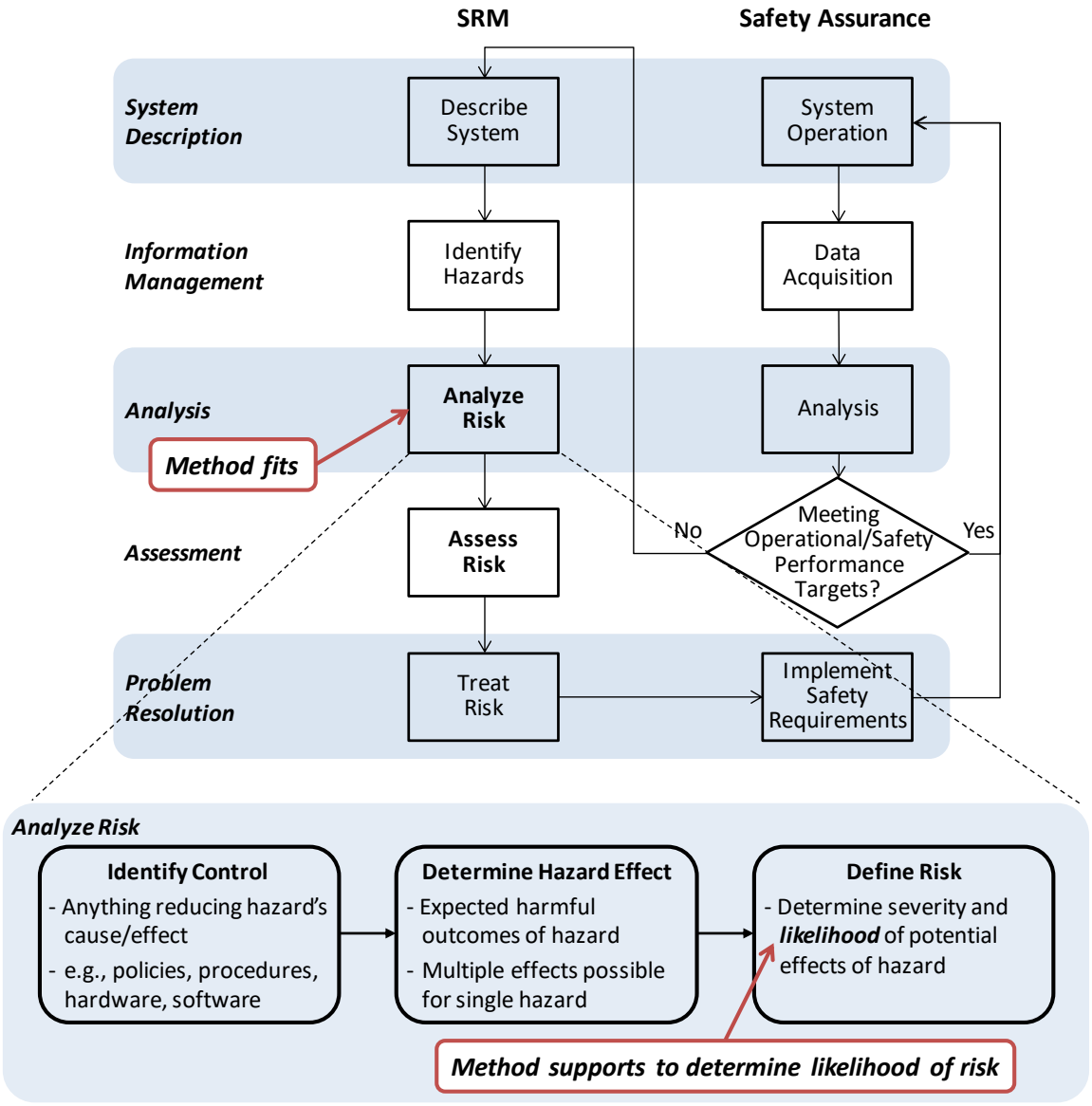


Figure 2 SRM / Safety Assurance process flow [FAA 2019]

Another application concept is to use the approach of this research in the development stage of the system life-cycle, which defines and realizes a system that meets its stakeholder requirements through specifying, analyzing, architecting, and designing the system [Walden 2015]. Figure 3 shows a typical Vee model illustrating

System Engineering (SE) activities during the development stage and where the method of this research fits. System developers can use the approach to design a system, e.g., a collision avoidance system, an aircraft, and even the NAS architecture, and to specify sub-systems and system elements with given system safety requirements (left side of the Vee model in Figure 3). The developer would have a benefit to quickly check if an alternative of the system design would be met a set of requirements for safety before realization of the system design alternatives. Fidelity is not as good as more specific models, but this method allows the analyst to narrow the decision space and to identify candidate designs for more detailed analysis.

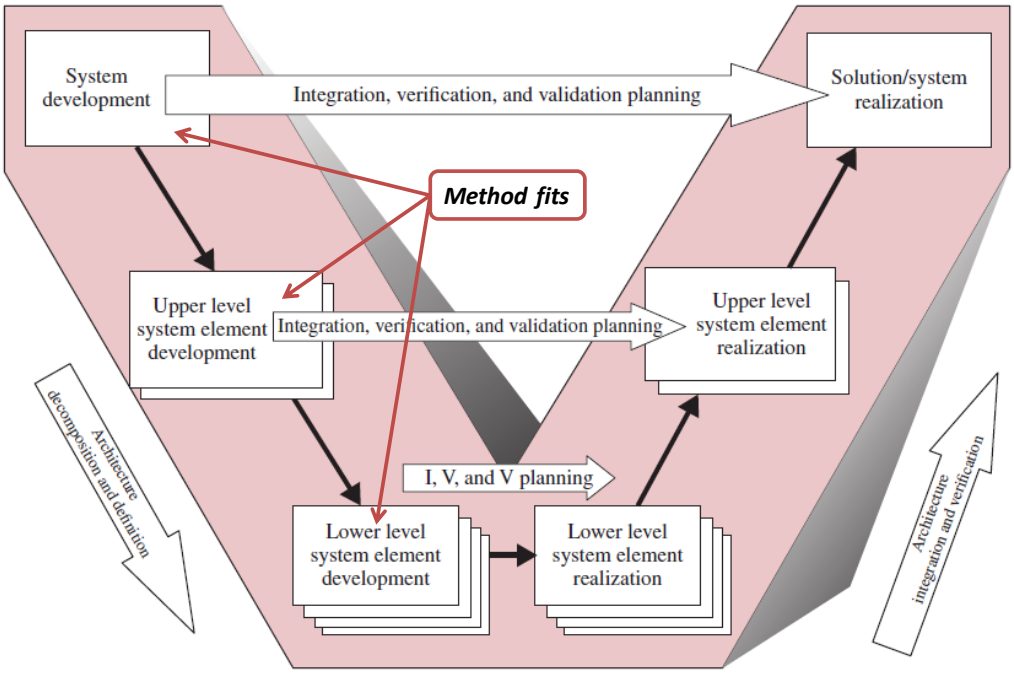


Figure 3 Vee model of development stage [Walden 2015]

CHAPTER 2 LITERATURE REVIEW

This chapter reviews literature related to evaluating mid-air collision risk in air transportation systems. Section 2.1 discusses various collision risk models in the literature. Section 2.2 discusses tree-based methods as a risk assessment model and its solution methodology. Section 2.3 describes future concepts of operations for the National Airspace System (NAS).

2.1 Collision Risk Analysis

Much research has been conducted to measure collision risk in the NAS. Table 1 shows a summary of example studies in which collision risk is evaluated for a specific concept of operation and/or region using a given methodology. The list is not exhaustive but meant to show some illustrative types of methods.

Table 1 Literature summary on collision risk analysis

Category	Paper	Collision risk model	Context
Analytical Models	Reich (1966)	Reich model	North Atlantic
	Endoh (1982)	Gas model	Random flights flying in a straight line
Event tree / Fault tree	Borener (2012)	Event tree / Fault tree	NAS
	Andrews (2005)	Fault tree	Advanced Airspace Concept (AAC)
Dynamic event tree (DET)	Shortle (2012)	DET	
	Zhang (2015)	DET + Simulation	Flow corridor concept

Simulation	Blum (2010)	Discrete Simulation (Monte Carlo)	AAC
	Belle (2012)	Continuous Simulation (FACET, ACES)	Chicago ARTCC
	Farley (2007)		Cleveland ARTCC
	Blom (2003)	Hybrid Simulation (TOPAZ)	Converging approaches
	Shortle (2004)		Nontowered airports
	Blom (2006)		Autonomous free flight

2.1.1 Analytical Models

One of the well-known analytical collision models is the Reich model [Reich 1966] which was developed to estimate collision risk for flights over the North Atlantic. A collision might occur because of the differences between the true values and intended values in three-dimensional speed and position (i.e., along-track, across-tack, and vertical direction). Both the probability distributions of the error magnitudes and the probability distributions of the rates of change of these errors are considered. The collision risk (per time) in a given dimension is the expected rate that two aircraft are within the collision distance in that dimension multiplied by the proportions of time that the aircraft are within the collision distances in the other two dimensions. The overall collision risk is the sum of the collision risk in each dimension.

Another widely known collision model is the gas model which describes the expected collision frequency under several simplifying assumptions – for example, that aircraft fly in a straight line with a constant speed and a uniformly distributed heading [Endoh 1982]. In addition, the gas model assumes that an aircraft is represented by a circular cylinder and that no collision avoidance maneuver is conducted, similar to the

Reich model. The gas model, which was originally developed with only two dimensions (i.e., on a horizontal plane), estimates the expected rate of collisions using the number of aircraft in a region, the area of the region, the expected relative velocity of a pair of aircraft, and the size of aircraft. The gas model is useful because it is a simple equation and it can provide some other risk metrics like losses of separation (LOS) with a simple modification of the aircraft size (reinterpreted as conflict dimension). Endoh [1982] also provided some extensions of the gas model including the vertical dimension and analyzing special cases like overtaking.

Analytical models are expressed by simple and clear mathematical equations to compute collision risk, and they can provide flexible risk measures such as loss of separation (LOS). However, analytical models have serious weaknesses: 1) They rely on several critical assumptions such as independent random deviations of aircraft positions and speeds (the Reich model) and straight line trajectories with uniformly distributed direction (the gas model), 2) they do not consider collision avoidance action by the pilots or the controller, and 3) equipment failures are not included.

2.1.2 Event Trees / Fault Trees

Event trees and fault trees (EF/FT) are commonly used methods in reliability and safety analysis. The methods are used in various industries such as nuclear power plants ([Vesely 1981]), air transportation, and the chemical industry ([Podofillini 2012]). ET/FT have been successfully used not only to evaluate risk, but also to identify scenarios representing risk and to suggest means to reduce the risk. The Integrated Safety Assessment Model (ISAM) developed by the Federal Aviation Administration (FAA), for

example, uses event trees and supporting fault trees to model all possible accident and incident scenarios for the current National Airspace System (NAS) [Borener 2012]. ISAM contains 35 event trees and hundreds of associated fault trees [Noh 2015], including one to model mid-air collisions. The model assesses current (baseline) risk of the NAS based on historical data. It can also estimate future risk based on proposed operational and procedural changes to the current system with projected traffic trends. However, it may be limited to incorporate new concepts/technologies introduced in the future since all the elements modeled in the ISAM are based on the current NAS. In addition, the model identifies a general sequence of events and associated causes of the events that leads to an undesired event, but it is not considering specific time horizon of the events that can change the consequence.

Andrews [2005] use fault trees to evaluate collision risk in a scenario where a highly automated separation assurance system is in place in the NAS. Safety functions designed in the Advanced Airspace Concept (AAC, [Erzberger 2001]) are assumed as the basic architecture for the NAS. Four types of faults (nominal conditions, information faults, control faults, and service interruptions) are identified, and a fault tree for each type is constructed and analyzed.

While ET/FT approaches are easy to understand from their graphical representations, they have weaknesses in treating the dimension of time, e.g., dynamic behaviors of the system and different operator actions as response to the system in time. In a collision scenario between aircraft, for example, timing of a conflict detected, when pilots are requested to take collision avoidance maneuver, and timing of the conflict

detection system failures all affect to the collision risk, and ET/FT approaches are limited to model these events appropriately.

2.1.3 Dynamic Event Tree

Dynamic Event Trees (DET) were proposed in order to include dynamic responses of the system, i.e., branching probabilities that vary as a function in time. For example, it is more difficult to correctly detect a conflict 20 minutes prior to a loss of separation compared to 3 minutes ahead. The conflict-detection probability is not a static number. Dynamic event trees extend standard event trees by including the time dimension to deal with dynamic occurrences of events in time without losing the capability of analytical evaluation.

Shortle [2012] use a dynamic event tree combined with reliability block diagrams to evaluate mid-air collision risk in the NAS under AAC operation. In this analysis, reliability block diagrams are first evaluated to determine whether various subsystems of AAC are functional or failed based on combinations of the component states. Then, several DETs are defined based on various combination of available functions. The generated DETs are evaluated to determine the conditional probability that a collision occurs. The collision probability is then computed by the weighted sum of the conditional probabilities with functional availabilities as weight, which are obtained by evaluating reliability block diagrams.

Zhang [2015] use a DET with Monte Carlo simulation to estimate the collision risk for a flow corridor concept with dedicated flight paths across the U.S. First, a Monte-Carlo simulation models aircraft movements in the flow corridors (e.g., passing,

overtaking, corridor lane changes) and estimates frequencies of potential mid-air collisions (i.e., conditions in which aircraft would collide in the absence of any conflict avoidance maneuver). DETs with reliability block diagrams are used to evaluate the performance of the onboard conflict detection and resolution functions to prevent a collision under the Autonomous Flight Management concept (AFM, [Wing 2011]), which is one of the automated separation assurance concepts for the future NAS.

One critical assumption of the DET method is that all components supporting a system fail only at the beginning of the time horizon of the analysis. In other words, the component failures are decoupled from the analysis of the dynamic event trees. Another drawback is that all combinations of component states need to be evaluated, which can be computationally expensive. Finally, the methods given in these papers are each developed in a somewhat ad-hoc manner for the specific problem being addressed and there is no formal description of an overarching framework for the methodology.

2.1.4 Simulation-based Models

The simulation-based methodology is one of the most commonly used approaches to assess collision risk. Simulation-based models can be divided into three types – discrete-time, continuous-time, and hybrid models. Discrete-time Monte Carlo (MC) simulation is often used to evaluate collision risk in the NAS where new procedures and/or technologies are implemented. Blum [2010], for example, used MC simulation to see what happens to two aircraft that are on a collision course under the Advanced Airspace Concept (AAC) environment. In each replication, the simulation first determines the failure state of each AAC component. From this, the failure states of AAC

subsystems are determined through fault-tree logic. Then, the simulation is run until the AAC system detects and resolves a conflict within a specified time (e.g., 8 minutes) and records whether or not two aircraft experience a near mid-air collision (NMAC) as well as the time when the AAC system resolves the conflict. The failure probability of AAC (i.e., the probability that two aircraft on a collision course eventually collide) multiplied by the rate that aircraft would be on a collision course in the absence of air traffic control (obtained from a gas-law model) provides the estimated rate of mid-air collisions in en route airspace.

The continuous simulation approach has been used to provide a base of collision risk analysis for the NAS. Several continuous time simulation tools have been developed to model the current NAS as well as to evaluate future concepts of operation for the NAS. One of the well-known simulation tools to analyze the NAS is the Future Air Traffic Management Concepts Evaluation Tool (FACET), which is developed by NASA [Bilimoria 2001]. FACET aims to provide a NAS-level simulation environment for exploration, development and evaluation of advanced ATM concepts such as distributed air traffic management and a decision support tool for controllers. FACET models four-dimensional (4D) aircraft trajectories using either flight plans or direct (great circle) routes considering winds. Another widely used tool for simulation and modeling the NAS is the Airspace Concept Evaluation System (ACES) which is intended to evaluate new system-level operational concepts of the NAS [Sweet 2002]. En route simulation with ACES is mainly based on FACET, and ACES also provides a flexible and extensible

simulation framework assembling a variety of models for physical systems, human operators, and rules and procedures.

From a collision risk analysis perspective, Belle [2012] use FACET to simulate flights trajectories and estimate rates of four types of proximity incidents (loss of separation (LOS), critical loss of separation (CLOS), near mid-air collision (NMAC), and mid-air collision (MAC)) under the assumption of no conflict resolution, i.e., without air traffic control. The resulting conflict rates, which are interpreted as the rates at which aircraft would get within a specified proximity without ATM, are shown as a function of flight count so that the rates can be used as the initiating event frequencies of other methods for collision risk analysis, e.g., ET/FT, DET or discrete simulation approaches (e.g., [Andrews 2005; Borener 2012; Shortle 2012; Blum 2010]).

Farley [2007] use a continuous simulation (i.e., ACES) more comprehensively to evaluate performance of an automated conflict resolution algorithm in several increasing traffic demand levels. The conflict resolution algorithm is implemented in the ACES environment, and it provides a resolution trajectory for detected aircraft on a collision course. Twenty-four hours of traffic are simulated to generate the number of conflicts detected and the number of conflicts successfully resolved to evaluate the safety performance of the conflict resolution algorithm.

The hybrid simulation methodology combines discrete failure events and continuous systems. An example is the TOPAZ (Traffic Organization and Perturbation AnalyZer) methodology, which was developed at the National Aerospace Laboratory NLR [Blom 1999]. The methodology aims to provide safety feedback of developing

concepts of operation for air traffic management (ATM) during the design stage. First, a qualitative safety assessment is conducted by identifying potential hazards and non-nominal scenarios. Then a quantitative assessment estimates accident risk through simulation of a stochastic dynamic system for the situation considered. Dynamically Colored Petri Nets (DCPN) are used to model stochastic differential equations of the hazardous scenarios selected on a hybrid state space (i.e., a state space where both discrete and continuous states exist). Since accident risks in aviation are rare, for a numerical evaluation, the methodology decomposes the problem into several conditional problems to which an appropriate evaluation method, e.g., MC simulation, is applied.

The TOPAZ methodology has been applied in several accident risk studies, where various ATM concepts of operations were analyzed. For example, Blom [2003] apply the TOPAZ approach to evaluate accident risk of simultaneous instrument approaches on converging runways at Amsterdam Airport Schiphol, Netherlands. Several operational scenarios are identified for missed approaches based on published procedures, operation modes and controllers' actions. Through simulation and mathematical modeling, the most critical scenarios are identified. Other examples where the TOPAZ methodology is applied include Shortle [2004], in which collision risk of landing airplanes at non-towered airports is evaluated, and Blom [2006] where the collision risk of the free flight concept is estimated.

Each type of simulation-based approach has unique pros and cons. The discrete simulation approach in Blum [2010] can deal with discrete component failures, but requires a lot of computing time – about 10 billion replications are needed to achieve

statistically meaningful results. The continuous NAS-wide simulation methods have an ability to accurately model the whole NAS, but require more than several hours to run a replication for one day of flights [Farley 2007]. These NAS-wide models also do not consider discrete component failures. Lastly, the hybrid simulation approach can model continuous processes (e.g., flight dynamics) combined with discrete event processes (e.g., runway occupancy sensor failures). Simulation methods, however, generally take much computation time to have statistically reasonable results for rare events via a direct simulation. In order to improve efficiency of the method, the hybrid models may need several other models such as the Reich model in Shortle [2004] since collision events are rare.

2.1.5 Summary

Four types of collision-risk methodologies (i.e., analytical models, ET/FT, DET, and simulation) have been described in this chapter. Each method has different features in terms of model complexity and computation time (see Figure 4). Analytical methods are simple and relatively easy to evaluate, but require a number of potentially unrealistic assumptions. Simulation-based approaches can model much more detail of the real system, but require a lot of computation time due to the rare-event nature of the problem. For example, to evaluate an event that occurs with probability 10^{-9}, 10^{9} replications are required to observe, on average, one event; many more replications are required to obtain a reasonable confidence interval. ET/FT and DET methodologies reside between the analytical and simulation approaches, where the DET methodology may model collision accidents more accurately than ET/FT.

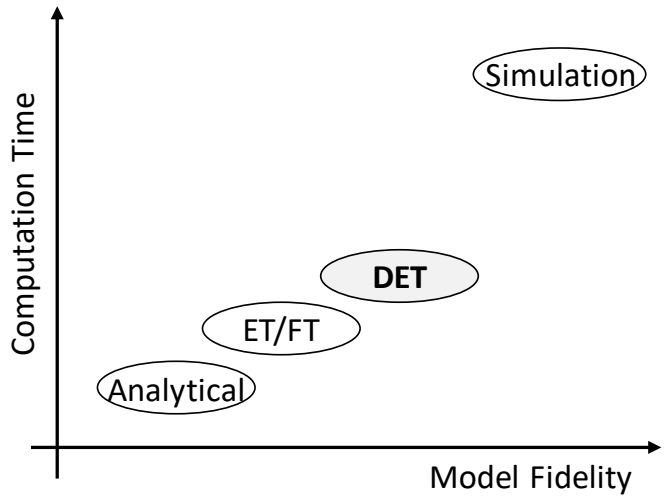


Figure 4 Comparison of collision risk models

A methodology to evaluate collision risk in this research should meet the following requirements:

- Correcting actions by pilots, air traffic controller and/or collision avoidance systems should be considered in the model.
- Discrete failures such as component failures in collision avoidance systems and mistakes by humans should be considered in the model.
- Time-varying system dynamics, such as trajectory prediction errors and encounter geometries, should be considered in the model, at least at an approximate level.
- Computation effort to assess collision risk should be reasonable, e.g., computation shall be done in minutes.

- Time to develop the model should be reasonable, e.g., creating a model shall be done in hours.

The goal is not to provide detailed system dynamics modeling, as some simulation models do. Rather, the objective is to include a comprehensive set of modeling elements as described previously in a manner that is relatively easy to model and compute. Therefore, this research focuses on a DET-based framework to assess collision risk for an airspace.

2.2 Tree-based Risk Models

A large number of systematic methods have been proposed to model and evaluate risk in systems, and the two most commonly used techniques for system reliability and safety analysis are event trees (ET) and fault trees (FT). Both ET and FT have a dynamic version, i.e., dynamic event trees (DET) and dynamic fault trees (DFT), to overcome a weakness in treating time element. The rest of this section discusses event tree analysis including more general dynamic event tree methods that have been studied in the literature (Section 2.2.1), static and dynamic fault trees (Section 2.2.2), and an existing computation method that can be applied to assess a DET in the context of collision avoidance (Section 2.2.3).

2.2.1 Event Tree (Static / Dynamic)

Because the analysis in this research focuses on event-tree-based methods, a more detailed literature review of event trees and their extensions is provided. Event tree analysis (ETA) is one of the most commonly used techniques for system reliability and safety analysis. According to Siu [1994], ETA can be used to evaluate the sequences of

system failures that can lead to undesired consequences. Event trees start with a single hazard or initiating event and model the sequence of events through safety layers that mitigate risk. An event tree can result in many different outcomes, and the tree provides a probability for each outcome. A path from the initiating event to each outcome indicates a possible scenario in terms of the occurrence or non-occurrence of various intermediate events. The probability of each outcome is calculated as the product of the branching probabilities along the path leading to each outcome. In Figure 5, for example, there are three different end events. The most severe event, a mid-air collision, occurs when air traffic control fails to resolve the conflict *and* the flight crew fails to resolve the conflict. The frequency of a mid-air collision is the product of the frequency of the initiating event and the probabilities of the two intermediate events.

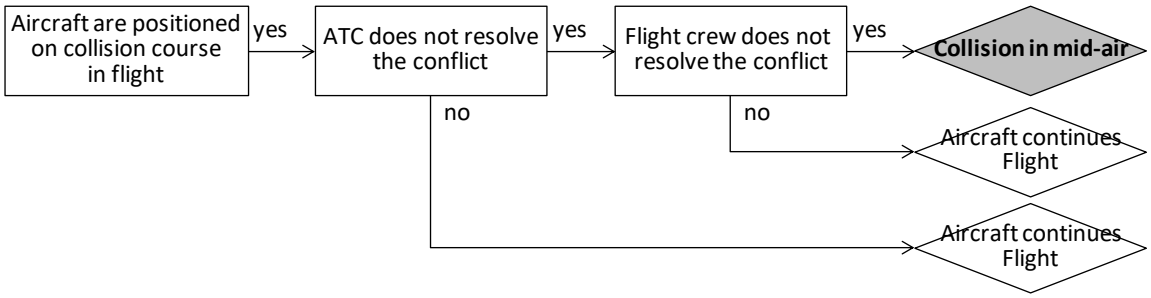


Figure 5 Example event tree for mid-air collision in ISAM

Conventional ETA, however, has a weakness in treating the dynamic responses of the system in time. To account for this, Dynamic Event Trees (DET), which is an extension of a static event tree adding the dimension of time, have been proposed. There are many DET methods. First introduced in the nuclear industry, these methods deal with

complex and dynamic behaviors of hardware, operators and the physics of the system in time. Some other DET methods deal with relatively simple dynamic behaviors, such as modeling success/failure probabilities that vary in time.

Table 2 Literature summary on dynamic event tree

Paper	Method	Branching	System State	Generated DET	Note
Cojazzi (1996)	DYLAM-3	Fixed points in time	Hardware, operator, and process variables tracked and simulated by problem-specific model	One big DET	Allow several kinds of probabilistic behaviors
Acosta (1993)	DETAM				Emphasize on operator crew states
Devooght (1996)	Probabilistic dynamics	Randomly chosen points in time	Vector of process variables and components status	Analytical equations	Continuous random transition, Analytical solution
Hofer (2004)	MCDET		Hardware, operator, and process variables	Many big DETs	Treat continuous and discrete random transition
Shortle (2012)	DET+RBD	Fixed points in time	Grouped combinations of hardware states treated as scenarios	Several small DETs	Appropriate for high-level system safety analysis
Zhang (2015)	DET+RBD				

Table 2 shows a summary of several studies using dynamic event trees. DYLAM (Dynamic Logical Analytical Methodology) is one of the earliest methodologies to assess the reliability of systems characterized by dynamic interactions and has evolved to version 3 (DYLAM-3) [Cojazzi 1996]. DYLAM is basically a tool that is a simulation driver able to generate branching scenarios of the system evolution at user specified time

intervals and to coordinate the simulation, which is a numerical model for the physical evolution of the system, at every branch. DYLAM-3 offers a framework to take into account six different kinds of probabilistic behaviors, 1) constant probabilities, 2) stochastic transitions, 3) functional dependent transitions, 4) stochastic and functional dependent transitions, 5) conditional probabilities, 6) stochastic transitions with variable transition rates.

Acosta and Siu [1993] propose the Dynamic Event Tree Analysis Method (DETAM), which is one of several variants of the DYLAM approach. DETAM allows a more general treatment of the integrated response of a nuclear power plant and its operators to an initiating event. The approach treats the time-dependent evolution of plant hardware states, process variable values, and operator states over the course of a scenario. DETAM especially has an emphasis on dynamic behaviors of the operator crew.

Devooght [1996] provide a mathematical formulation of the probabilistic dynamics of the system allowing for continuous random transitions. This approach can capture the possible dependencies among failure events due to process/component/human interactions in a single integral equation, but the computation effort is impractically large. Monte Carlo Dynamic Event Tree (MCDET) [Hofer 2004], a combination of Monte Carlo simulation and dynamic event tree analysis, is one approximation method to the analytical solution of the probabilistic dynamics. Monte Carlo simulation provides sets of random values, like times for components to fail, then MCDET generates samples of the dynamic event tree, and evaluates conditional probabilities and outcomes of all paths in each tree.

Shortle [2012] and Zhang [2015] use a dynamic event tree to evaluate mid-air collision risk for the Advanced Airspace Concept (AAC, [Erzberger 2004]) and for the flow corridor concept of operation, respectively. Both papers evaluate reliability block diagrams at first to determine whether various system functions are working or failed based on combinations of components states. Then, they generate several DETs, including a baseline DET and variants of the baseline DET, based on the combination of available functions. Lastly, they evaluate the DETs to determine the conditional probability that a collision occurs given two aircraft are on a collision course.

Researchers can use a DET to comprehensively model a system with details, while it may require a separate system simulation model and it generates a complicated and huge event tree which may need a lot of computation effort. Or, a DET can also be modeled by simply incorporating time-dependent branching probabilities to take advantages of easy implementation and solving a problem quickly, but the results may not be as accurately detailed as one from the former.

2.2.2 Fault Tree (Static / Dynamic)

Fault trees are generally used to identify all combinations of component failures (e.g., Conflict alert system fails) that can lead to system failures (e.g., ATC fails to resolve the conflict) [Siu 1994]. An undesired event, e.g., system or subsystem failure, is defined as the top event of the fault tree. The causes that can lead for the undesired event to occur are articulated and placed under the top event of the fault tree at a certain level with a logical relationship gate, e.g., AND gate, OR gate. A fault tree thus represents

logical relationships between the undesired event and the causes, i.e., the basic fault events, graphically [Xing 2008].

Fault trees are logical models of fault combinations that could cause a mitigating system to fail to perform. An analysis of fault trees is studying minimal cutset. A cutset is defined as a set of basic events whose occurrence ensures that the top event occurs. A minimal cutset is a cutset without redundancy, i.e. a cutset is said to be minimal if the set cannot be reduced without losing its status as a cutset [Rausand 2004]. Fault trees are also used to determine the probability of the undesired event occurrence, given estimated or measured occurrence probability of each basic event. Cutset-based methods are conventionally used to compute the top event probability, however, they are computationally challenging without approximations. Another method to compute the top event probability of a fault tree is to use Binary decision diagrams (BDDs). The BDD method is a relatively recent method to solve a fault tree model for the system reliability analysis [Rauzy 1993]. Introduction of BDD in the reliability analysis has improved accuracy and efficiency in fault tree analysis [Sinnamon 1997].

Another reasonably used approach to model system reliability/risk is dynamic fault trees (DFT), which extends static fault tree to include dynamic system behavior [Dugan 1990]. With several dynamic gates DFT appropriately models complex behaviors and interactions between components such as sequence dependences, spares, and priority of events. Markov model is commonly used to solve DFTs, however Markov model has the significant disadvantage that its size grows exponentially as the size of the system increases. The modular approach, which is a combination of solutions from combinatorial

and Markov model, solves DFT more efficiently [Gulati 1997]. DFTs may be used to model the second risk in Figure 1 that system fails to resolve a conflict, but it could be complicated to model sequence of both failure events and success events, which occur in mid-air collision scenario. Furthermore, Markov model used to solve a dynamic fault tree is limited to deal with time-varying failure/success rates.

2.2.3 Computational Method (Phased-Mission Systems)

Phased-mission systems (PMSs) are systems in which multiple non-overlapping phases of operations (or tasks) are accomplished in sequence for a successful mission [Xing 2013]. In these systems, the system configuration, success criteria, and component behavior may vary from phase to phase [Zang 1999]. One example of PMS is a flight of an aircraft, which consists of taxing, take-off, cruise, approach, and landing phases. In each phase, different reliability criteria and behaviors are required. Figure 6 shows a simplified example in which a flight "mission" is accomplished if an aircraft successfully performs all three phases, i.e., take-off, cruise and landing. Different combinations of components are required in different configurations in each phase.

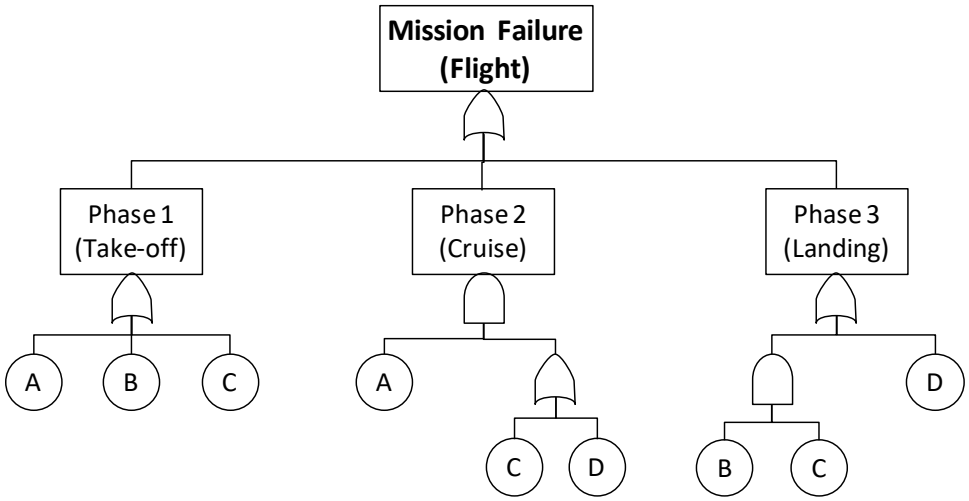


Figure 6 Example phased-mission system (Flight)

PMSs have similar operational structure with conflict detection and resolution (CD&R) systems (Figure 7). To perform collision avoidance, several CD&R systems usually operate in sequence to prevent a collision. For example, there is usually a strategic CD&R system that looks several minutes ahead to identify conflicts. There is also a tactical system that avoids imminent collisions at the last moment. These systems may involve different sets of components, software, and system configurations. Collision avoidance fails if every CD&R system in the sequence fails. This is analogous (but opposite) to the structure of phased-mission systems, which are successful if every phase of the mission is successful.

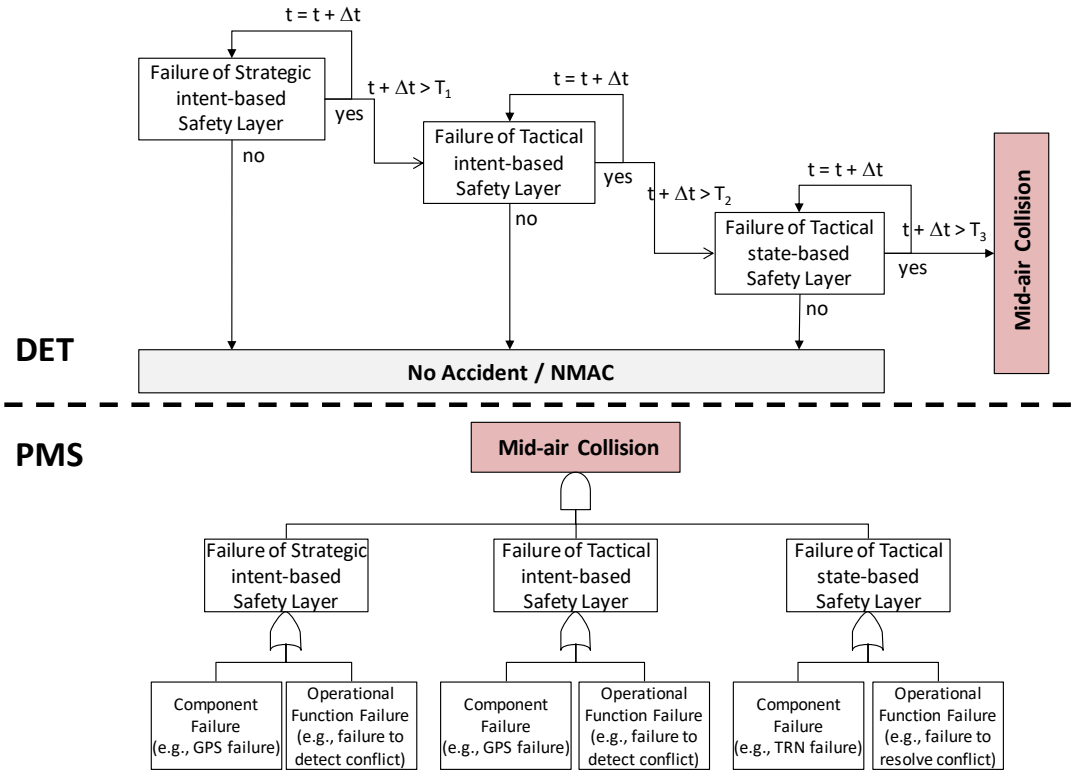


Figure 7 DET and PMS representations for an example CD&R process

There are two commonly used approaches to evaluate PMSs (Table 3): State space oriented models ([Kim 1994]) and combinatorial methods ([Zang 1999; Xing 2002; Xing 2013]). State space oriented approaches are mainly based on Markov chains. The methods are flexible and powerful in modeling complex dependencies in system components. One assumption in these approaches, however, is that failure (and repair in some research) times of components are exponentially distributed. Another drawback of the methods is that the number of states included in the model can easily explode [Xing 2008].

Table 3 Literature summary on Phased-mission systems

Paper	Method	Components	Common Cause Failure	Failure
Kim (1994)	Markov model	Repairable	-	Exponential
Zang (1999)	PMS-BDD	Non-repairable	-	General
Xing (2002)	PMS-BDD	Non-repairable	Yes	General
Xing (2013)	PMS-BDD	Non-repairable	Yes	General

The combinatorial methods include the mini-component technique, which was introduced by Esary et al. [1975], to deal with the dependence across the phases using a set of independent mini-components to replace the component in each phase. Zang et al. [1999] proposed a binary decision diagram (BDD) based algorithm for reliability analysis of phased-mission systems (PMS-BDD). PMS-BDD uses phase algebra to deal with the dependence across the phases. PMS-BDD has been extended to include any combinatorial phase requirement and to incorporate common cause failures [Xing 2002; Xing 2013]. One limitation in the combinatorial methods is that they can only deal with non-repairable components.

This research uses PMS-BDD method (or a combination with another method) to evaluate a collision risk modeled in a dynamic event tree structure, since 1) non-repairable components are assumed to support conflict detection and resolution systems; 2) PMS-BDD is an analytical approach so that computation would be fast.

2.3 Future NAS

The air transportation system is predicted to face significant increase in terms of both the demand of flights and the diversity of aircraft. To accommodate the increasing demand of various aircraft types and to provide safer air transportation service, researchers have studied about the future NAS. This research would be more properly applied for the future NAS environment than the present because the approach models a collision risk between various types of aircraft with a different level of collision avoidance capabilities, which is not available now, but expected to be placed in the future. The future NAS has been investigated in terms of new concepts of operations for the NAS (e.g., automated NAS, Section 2.3.1), collision avoidance systems (Section 2.3.2), and even new architecture of the NAS (Section 2.3.3).

2.3.1 Concept of Operations (Automated NAS)

Two widely studied concepts of operations for the future NAS are the Advanced Airspace Concept (AAC, [Erzberger, 2001]) and Autonomous Flight Management (AFM, [Wing 2011]). AAC was proposed to improve the capacity of the NAS by reducing controller workload. To do this, AAC has a ground-based central computer that automatically monitors aircraft separation and sends trajectories to resolve conflicts directly to aircraft via an air-ground data link. Figure 8 shows the system architecture of AAC. The central system on the ground has two separate CD&R systems, Autoresolver (AR) and Tactical Separation Assured Flight Environment (TSAFE). The two systems are designed to detect and resolve conflicts in different ranges of time, 2-20 min prior to predicted conflicts for ATS and 0-3 min for TSAFE. While air traffic controllers are able

to devote more time to solving strategic control problems under AAC operation, they still have separation assurance responsibilities for aircraft improperly equipped or facing failure like loss of data link or on-board system failures.

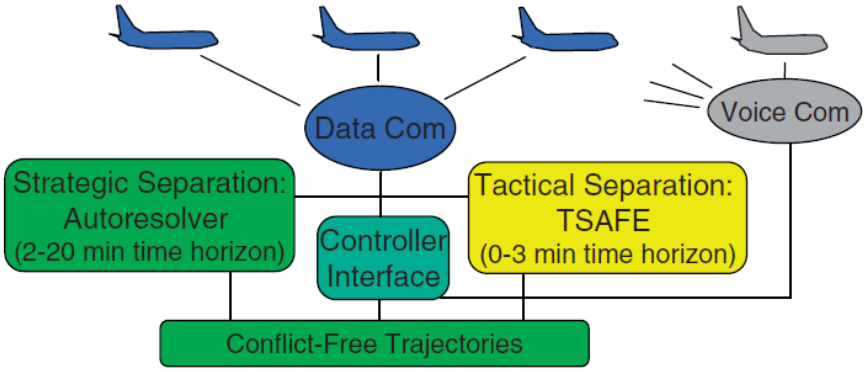


Figure 8 Most current system architecture of AAC [Erzberger 2012]

AFM has been developed from the free flight concept of operation, which is defined as the operators have the freedom to select their path and speed in real time [RTCA 1995]. A key concept of separation assurance for free flight called "self-separation", which is the ability to remain in a safe distance to all other aircraft by the pilot themselves, was enabled by emerging technologies, and it allows the responsibility for separation to be distributed among ground and airborne elements. Wing [2011] defines Autonomous Flight Rules (AFR) related to AFM based on the self-separation concept. Under AFR operations, aircraft and flight crew can maintain separation from all other aircraft, terrain, and obstacles without the ground-based air traffic management system.

AFR operations are enabled by several technologies, which are emerging and/or already in use. Automated Dependent Surveillance Broadcast (ADS-B) is the primary resource of the position information (including altitude and velocity) of other aircraft. As a backup system to provide surveillance information, the ground-based Traffic information Service Broadcast (TIS-B) can be used. An Airborne Separation Assurance System (ASAS), i.e., onboard collision detection and avoidance (CD&R) system, is also required for an AFR flight to maintain a safe distance from the other aircraft. ASAS includes physical components (e.g., a processor) as well as logic that performs conflict detection, resolution, and prevention functions. In addition, a data communication link between air and ground as well as the Traffic Alert and Collision Avoidance System (TCAS) are required to support AFR operations.

2.3.2 Collision Avoidance Systems (Conflict Detection and Resolution)

In the current airspace system, maintaining separation between aircraft has usually been conducted by humans, i.e., air traffic controllers, so that the capacity of the airspace relies on the ability of humans. However, not only air traffic demand, but also the diversity of vehicle types in the NAS, are expected to increase substantially in the future. In order to accommodate these increases, automated air traffic control systems are needed to help air traffic controllers to detect and resolve conflicts between aircraft. Much research effort has been placed to develop methodologies for automatic conflict detection and resolution (CD&R) [Kuchar 2000]. Most of the CD&R research proposes a new method and evaluates its performance, i.e., how well the CD&R method detects and resolves conflicts in a certain environment.

According to Kuchar [2000], the CD&R process begins with collecting current state information of aircraft in the airspace through sensors such as surveillance radar. A dynamic trajectory model calculates the states in the future to predict whether a conflict will occur. The current and predicted states can then be combined to derive metrics, e.g., estimated minimum separation, to make traffic management decisions such as whether action by controllers or pilots is needed. When action by a human is required, the conflict resolution function determines an appropriate series of actions and informs the controller and/or pilots.

With increasing demand of integration of UAVs into the NAS, a large number of studies have been conducted related to collision risk of UAVs. Kuchar [2005] proposes a safety analysis framework for UAV collision avoidance systems that may be used for certifying such systems. The proposed methodology estimates collision avoidance system performance through a combination of airspace encounter modeling, fast-time simulation of the collision avoidance system for numerous encounter scenarios, and fault tree analysis of system failure. The approach considers various encounter geometries, aircraft dynamics, CAS logic failure, and pilot response delays through the simulation, then component-based system failures with the simulation results are used to model a fault tree for the overall collision risk.

Weibel [2011] presents an analytical approach to defining a well clear threshold to evaluate self-separation performance, which is the ability to remain in a safe distance from other aircraft. “Well clear” is framed as a relative state, defined by the time to closest point of approach (CPA) and distance, between aircraft for which the risk of

collision is acceptable. Through encounter simulation conditional probabilities of collision are shown as a function of each relative state variable, i.e., time to CPA and distance.

Ferreira [2018] presents a risk analysis of integration of UAS (specifically, Remotely Piloted Aircraft System, RPAS) into non-segregated airspace. Mid-air-collision and ground collision risk caused by two new types of hazard (Command and Control link failure and jamming attack), which are introduced due to insertion of RPAS, are evaluated through the fault tree analysis. All the operational failure events (e.g., ATC failure generating conflict) are modeled like a component failure in the fault tree even though a simulation is used to compute the probability that a conflict occurs.

Unlike the researches explained above, collision risk between unmanned aircraft is assessed using Monte Carlo simulation [Jenie 2018]. Small UAVs with an onboard CD&R system are assumed to fly in a hypothetically dense airspace to compute Near Mid-air-collision (NMAC) and Mid-air-collision (MAC) frequencies efficiently. Then, the NMAC and MAC frequencies in a realistic (less dense) airspace are derived using the gas model. Two types of CD&R procedures, uncoordinated and implicitly coordinated (with right-of-way rule) resolution maneuver, as well as a case without CD&R systems are analyzed. The simulation takes into account various conflict geometries, aircraft dynamics, and CD&R algorithmic failure due to uncertainty of trajectories, while component-based system failures and pilot (human or autonomous) delays are not considered.

Table 4 summarizes literature that perform risk/safety analysis related to UAV. Kuchar [2005] considers all factors that affect to the risk using a combination of fault tree and simulation, while the others take into account only limited factors as applying a single method. Conflict geometries are considered through simulation in most literature, however, they are all aggregated to calculate a single risk. In addition, most literature analyze a risk between a specific combination of aircraft, e.g., manned and unmanned aircraft or unmanned and unmanned aircraft, or unspecified ([Ferreira 2018]). The approach of this research takes into consideration all the factors but aircraft dynamics to evaluate collision risk between any combination of aircraft.

Table 4 Literature summary on UAS-related risk/safety analysis

Paper	Collision risk model	Context	Considered Factors
Kuchar (2005)	Fault tree + Simulation	Collision avoidance system of UAV (to manned aircraft)	- Conflict geometries - Aircraft dynamics - Algorithm failure - Human behavior - Component failure
Weibel (2011)	Simulation	Establishing risk-based separation standard	- Conflict geometries - Aircraft dynamic
Ferreira (2018)	Fault trees	Collision risk due to C2 link failure / jamming	- Component failure
Jenie (2018)	Simulation	UAV CD&R system (between UAVs)	- Conflict geometries - Aircraft dynamic - Algorithm failure
Zhang (2018)	Gas model (Simulation)	Collision risk of UAV to manned aircraft	- Airspace density - Speed of aircraft

2.3.3 New Architecture for NAS (Risk-based Sector Capacity)

In order to accommodate future increasing demand of both traffic and diversity of aircraft for the NAS, a new architecture for the future NAS has recently been proposed by Wieland [2017]. The architecture that could be a basis of autonomously controlled airspace in the far future has been developed by a 'clean sheet' approach, in which the architecture may apart from the current NAS system. One of the high level requirements considered for the architecture is that all flights including any type of UAVs have the same procedure for access to the NAS, i.e., "file and fly" today where a manned aircraft files a flight plan and flies in 45 minutes. This requirement may be the most different from the current system since all UAV flights need to go through a long Certificate of Authorization process to access to the current NAS.

The architecture can be summarized as dynamic and risk-based sectors. The architecture is based on dynamic sectors since the airspace is divided into sectors that are not fixed and may divide and merge as traffic densities change throughout the day. Traffic density, which causes the sectors divided and combined, is considered as collision risk because as the number of aircraft in the sector increases, the number of times that a loss of separation occurs as well as collision risk increase. Sectorization in the architecture is based on the Required Collision Avoidance Probability (RCAP), which is the probability that the entire collision avoidance functions including strategic through tactical functions has failed. Whether or not an aircraft is allowed to enter a sector is depending on the RCAP of the sector as well as the performance of collision avoidance functions equipped by the aircraft.

In order to support the new architecture, Shortle [2017] provided a framework to evaluate collision risk and thus to issue admittance based on the estimated collision risk. Figure 9 illustrates the concept of airspace admittance function, which controls admittance into a sector. If an aircraft wishes to enter a sector, the collision risk function within the admittance function is evaluated including existing flights in the sector and a new flight, and admittance is given when the resulting collision risk in the sector remains at or less than a target level of safety (TLS). A key aspect of the proposed framework is to consider not only the traffic density but also the collision avoidance capabilities to compute collision risk. Therefore, frequently evaluating collision risk between various types of aircraft (including different types of collision avoidance equipage) would be a main function to support the architecture.

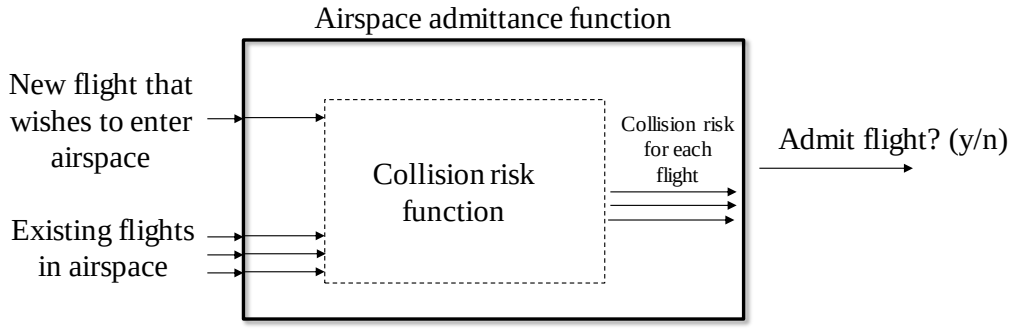


Figure 9 Airspace admittance function [Shortle 2017]

Collision risk can be decomposed into two parts, the risk that two aircraft are on a collision course and the risk that separation assurance and collision avoidance functions fail given two aircraft on a collision course ([Wieland 2017; Shortle 2017]). The first risk

depends on traffic density, while the second risk relies on the capability of collision avoidance functions on aircraft. Shortle [2017] used an analytical method and a simulation method, particularly the gas model and ACES simulation, to evaluate the first risk, i.e., the probability that two aircraft are on a collision course given a traffic density. Then the admittance function was evaluated for several cases where various combinations of relatively different collision avoidance capabilities are assumed, i.e., the second risk, which is the probability that collision avoidance functions fail, was not evaluated explicitly.

CHAPTER 3: METHOD FOR COLLISION RISK ASSESSMENT

Static event trees model the sequences of failure events that can lead to undesired events. Figure 10 shows a static event tree that models collision risk assuming three types of conflict detection and resolution (CD&R) systems in operation. A mid-air collision occurs when all three CD&R functions fail to perform, starting from an initiating event in which two aircraft are positioned on collision course in flight. A basic assumption of static event trees is that the event probabilities are fixed regardless of when the events occur. In reality, the performance of each CD&R function changes in time, e.g., the probability of successfully detecting a conflict increases as the aircraft get closer together. The geometry of the conflict also impacts the event probabilities.

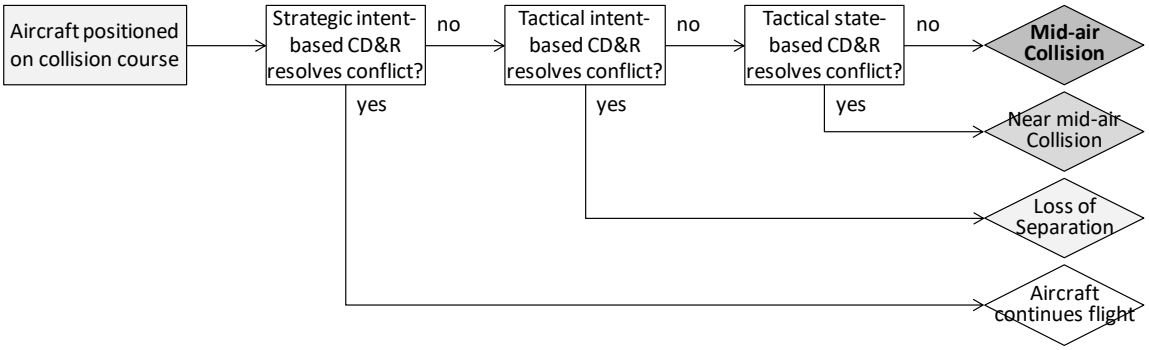


Figure 10 Example static event tree of mid-air collision

To account for time dependence, dynamic event trees (DETs) provide an extension to static event trees in which the dimension of time is added to include dynamic performance of collision avoidance systems [Shortle 2012; Zhang 2015]. This chapter presents a general framework for modeling mid-air collision scenarios using dynamic event trees (DET) and describes several methods that can be applied to evaluate the framework. An example DET is used to illustrate each method.

3.1 Canonical Form of Collision-Risk Dynamic Event Tree

The general framework for a collision-risk DET consists of three levels – a high-level dynamic event tree, a generic sub-tree, and fault trees (Figure 11). The high-level tree (Figure 11, left) has multiple phases of conflict detection and resolution (CD&R) that operate in sequence to prevent a collision. (The figure shows three CD&R systems, though an arbitrary number can be modeled.) The generic sub-tree (Figure 11, middle) provides a more detailed template for the sequence of events within each CD&R phase. Fault trees (Figure 11, right) model logical relationships between failure of a CD&R system and failure of physical components supporting the CD&R system. Figure 11 shows specific examples, but the structure of these fault trees can be general.

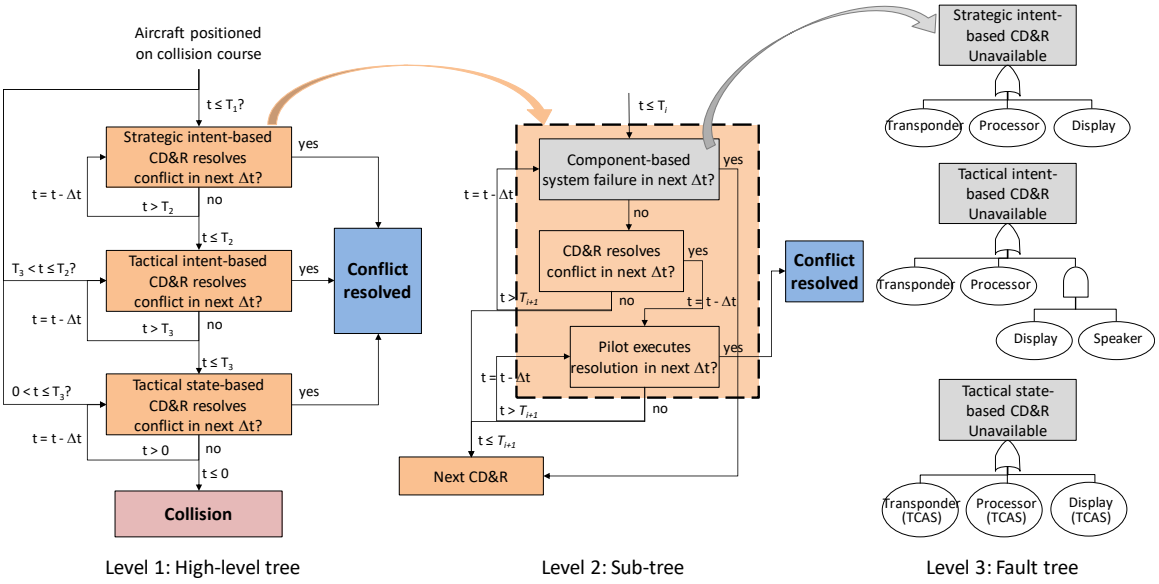


Figure 11 General **f**ramework of **d**ynamic **e**vent **t**ree for **m**id-**a**ir **c**ollision with example conflict detection and **resolution systems**

As an illustrative example, *Autonomous Flight Rules* (AFR) [Wing 2011], proposed by NASA, is a framework for maintaining safe separation without controllers. Aircraft flying under AFR are equipped with three CD&R systems. The first system manages conflicts in a strategic manner up to twenty minutes ahead using flight plans and intent information. The second system manages conflicts up to three minutes ahead and uses intent information, but also considers aircraft dynamics (e.g., turn radii) to resolve conflicts in a more tactical manner. The final system uses current state information (without intent, simply projecting aircraft locations forward based on their current speeds and directions) to avoid imminent collisions.

High-level tree: The high-level event tree (Figure 11, left) captures the following logic. The initiating event is a situation in which two aircraft are on a collision course; *t* is defined as the time remaining to a collision. This value is decremented by a small amount

Δt in an iterative manner until either a collision occurs or is avoided. Each CD&R system attempts to detect and resolve the conflict until either the conflict is resolved or t reaches a designated time point T_i, at which point the next CD&R system takes over from the previous one.

The following parameters are required to specify the high-level dynamic event tree:

- Time horizon T
- Number of phases r
- Time horizon for each phase, T_1, T_2, …, T_r (with $T > T_1 > T_2 > \ldots T_r > 0$)
- Time step Δt

Generic sub-tree: The generic sub-tree (Figure 11, middle) shows a template for evaluating a sequence of events within each CD&R phase. In the example, the CD&R function requires (a) correct functioning of the physical components, (b) successful detection of the conflict via the conflict detection algorithm (e.g., correctly predict trajectories under uncertainty), and (c) correct pilot execution of the resulting resolution. The sub-tree is structured as a dynamic event tree. Some of the transition probabilities are dynamic. For example, the probability of successfully detecting a conflict increases as the aircraft get closer together, since there is less uncertainty in the trajectory predictions at shorter time horizons.

The following inputs are required to specify the generic sub-trees:

- Set of state transitions (k, l) and associated time advancement for each transition (TA_{kl}, where $TA_{kl} = 0$ if transition from state k to l occurs instantly; $TA_{kl} = 1$ if

transition from state k to l takes a time step).

- Time-dependent conflict-detection rate $\mu(t)$. This function defines the infinitesimal rate of detecting a conflict t minutes prior to a mid-air collision. In a short time interval $[t, t - \Delta t]$, the probability of detecting a conflict is approximately $\mu(t)\Delta t$.
- Execution rate γ. The average time for a pilot to execute the resolution is exponentially distributed with mean $1 / \gamma$.

Fault trees: Fault trees (Figure 11, right) model logical relationships between physical components and the functional failures of the CD&R systems. Some components may support multiple CD&R systems. For example, a transponder broadcasts and receives position information of aircraft and may be used to locate aircraft in multiple CD&R systems.

The following parameters are required to specify the fault-tree portion of the model:

- Number of components m
- Failure rate λ_i of component i ($i = 1, 2, \ldots, m$)
- A fault tree that maps the component states to the working status of each CD&R system. Let X_i be the status of component i where $X_i = 0$ if component i is failed; $X_i = 1$ if component i is working. Let $f_j(X_1, \ldots, X_m)$ denote the state of CD&R system j (0 is failed, 1 is working), where $j = 1, 2, \ldots, r$.

In addition to the structure of the dynamic event tree, sub-trees, and fault trees, several assumptions are made: All components are statistically independent of each other.

All components are unrepairable, so that once a component fails, it remains in a failed state for the time horizon of the analysis.

3.2 Solution Methods

In order to solve the proposed general form of the DET, several methods from the literature are applied. The first method was proposed in Shortle [2012]. We call this the conditional dynamic event tree (cDET) method. The second method is based on a binary decision diagram approach to analyze phased mission systems (PMS-BDD) that is popular in the PMS reliability analysis [Zang 1999]. The third approach is a combination of the first two methods. We call this method cDET-PMS. Lastly, a Monte-Carlo simulation is applied to provide an approximation of the true solution. The methods have different assumptions on the timing of component failures as well as the internal logic of the sub-trees.

The model in Figure 11 is used to illustrate the algorithm of each methodology. Three CD&R systems – strategic intent-based, tactical intent-based, and tactical state-based – are respectively activated at times T_1, T_2, and T_3 and operate in consecutive and non-overlapping time periods. Each CD&R phase successfully resolves a conflict when all three conditions are met within the given time period: i) the CD&R system is working properly, ii) the CD&R function detects and resolves the conflict, and iii) the flight crew executes the resolution in the specified time horizon.

3.2.1 Solution Method 1: Conditional Dynamic Event Tree (cDET)

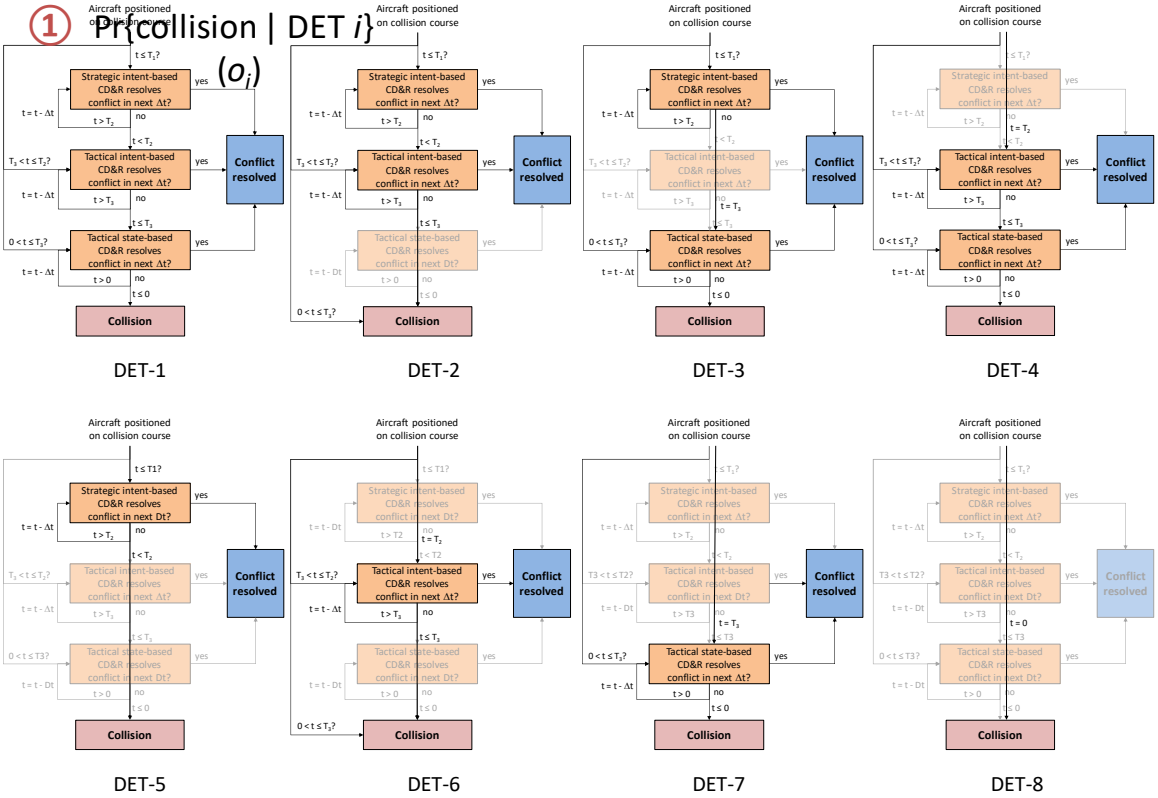
The conditional DET approach is based on ideas in the literature ([Shortle 2012; Zhang 2015]). The main assumption is that component failures occur at the start of the

analysis time window – i.e., T minutes prior to a potential collision. Each combination of component states corresponds to a particular working state of each CD&R system. When all components are working (the typical case), then all CD&R systems are working. The accident logic is given by a baseline DET where each CD&R system, in succession, attempts to detect and resolve the conflict until either the conflict is resolved or a collision occurs (DET-1 in Figure 12).

When some components are failed, then one or more of the CD&R systems may fail. A failed CD&R system no longer provides any capability to prevent a collision, so the DET logic can be modified by skipping over that phase in the event tree. In a worst-case scenario, when all CD&R systems are failed, there is nothing to prevent a collision, so the initiating event (two aircraft are lined up on a collision course) results in a collision with probability 1 (DET-8 in Figure 12).

With r CD&R phases, there are at most 2^r distinct DETs (Figure 12). Each DET provides a *conditional* risk – namely, the probability that a collision occurs given specific availability states of the CD&R systems (and conditional on the initiating event that two aircraft are lined up on a collision course). The conditional risk for each DET can be evaluated using the methods described in Shortle [2012]. The overall collision risk is then the weighted average of the conditional risk from each DET, where the weights are the probabilities of being in a particular CD&R functional state.

Figure 12 shows a graphical representation of the algorithm applied to the example in Figure 11. For the numerical results, it is assumed that time horizon $T = 8$ min and $\lambda_i = 10^{-6}$ / min for each component.



② Case (j)	Component States (0-failed, 1-working)							Pr{case j occurs} (p_j)	System Availability (0-unavailable, 1-available)			DET (i)	Pr{DET i occurs} (q_i)
	trn	pro	dis	spk	TRN	PRO	DIS		SICDR	TICDR	TSCDR		
1	1	1	1	1	1	1	1	0.9999	1	1	1	1	0.99995
2	1	1	1	0	1	1	1	8.00E-06	1	1	1		
3	1	1	1	1	1	1	0	8.00E-06	1	1	0	2	2.40E-5
4	1	1	1	1	1	0	1	8.00E-06	1	1	0		
⋮													
127	0	0	0	0	0	0	1	2.62E-31	0	0	0	8	3.84E-10
128	0	0	0	0	0	0	0	2.10E-36	0	0	0		

Figure 12 cDET method applied to example problem

The cDET algorithm is given by the following logic:

1. Create 2^r DETs, where r is the number of CD&R systems or phases. Each DET

models the sequence of events that could occur when a given combination of the CD&R systems are available. For each combination of CD&R states (i = 1, …, 2^r), calculate the conditional probability o_i associated with DET i. This is the probability a collision occurs given the specified availability of CD&R systems. These probabilities can be computed using the method described in Shortle [2012].

2. For each combination of CD&R states (i = 1, …, 2^r), compute the probability q_i of state i as follows:
    a. For each combination of component states (j = 1, 2, …, 2^m), compute the probability p_j of state combination j.
    b. Sum all of the component-state probabilities p_j that result in CD&R state i, via the fault-tree logic for each CD&R system.
3. The overall collision probability is the weighted sum of the conditional collision probabilities ($\Sigma_i\ q_i * o_i$).

This formalizes the method used in Shortle [2012]. Note that Step 2 is a brute-force approach to evaluate all 2^m component combinations. A more efficient approach is to use binary decision diagrams (BDD) [Rauzy 1993] to evaluate the CD&R state probabilities q_i. The BDD method groups certain combinations of component states together, eliminating the need to enumerate every state combination [Sinnamon 1997]. A revised approach for Step 2 is as follows:

2. For each combination of CD&R states (i = 1, …, 2^r), compute the probability q_i of state i as follows:

a. Create a *combined* fault tree where the top event is state combination *i* of the CD&R systems (e.g., CD&R system 1 is unavailable, system 2 is unavailable, and system 3 is available). The combined fault tree is composed of the individual CD&R system fault trees and/or success trees, depending on the state of each CD&R system (see Figure 13).

b. Convert the combined fault tree to a BDD. Evaluate the BDD to obtain the probability q_i.

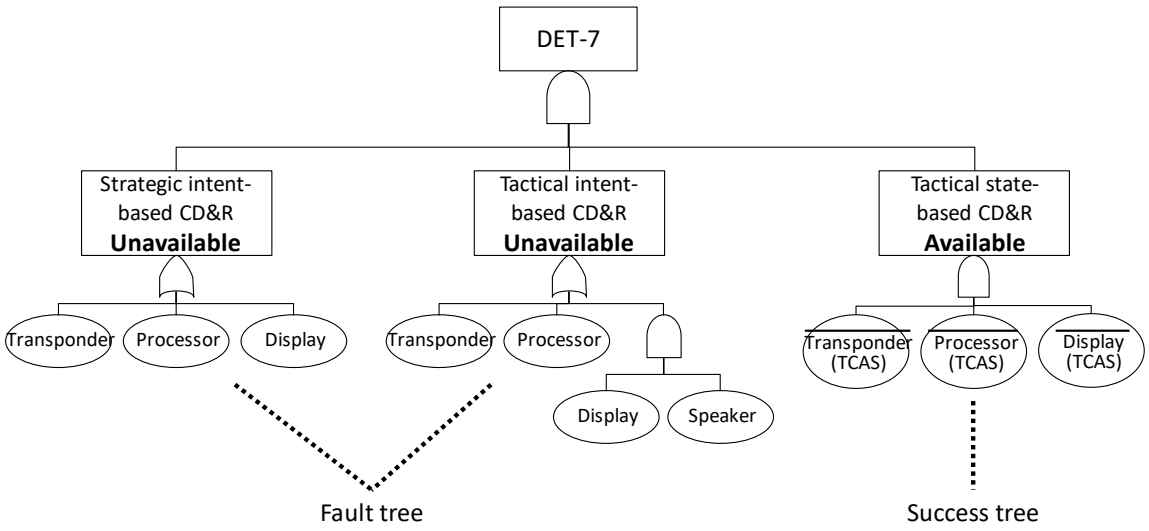


Figure 13 Examples of combining fault trees and/or success trees

Often, fewer than 2^r DETs need to be specified, since some combinations of function states may not be possible. For example, in Figure 11, there are eight possible DETs, but only six are needed, since it is impossible for the strategic intent-based CD&R (SICDR) system to be working/available while the tactical intent-based CD&R (TICDR) system is unavailable, based on the logic of fault trees.

A limitation of the cDET method is that the physical component failures are assumed to occur only at the beginning of the analysis time window. This may over-estimate collision risk, since cases where component failures occur after the conflict is resolved may be counted as collisions when occurring at the start of the time horizon.

3.2.2 Solution Method 2: Binary Decision Diagram based method for Phased Mission Systems (PMS-BDD)

The proposed form of a dynamic event tree for collision avoidance has a similar structure with a phased-mission system (PMS). In a phased-mission system, a mission is accomplished through several phases, and the mission is successful if every phase is successful. In the collision avoidance problem, each phase corresponds to a distinct CD&R system that attempts to prevent a collision. The collision problem is analogous to the PMS problem, but with a "negated" structure – namely, while a phased-mission system is successful if every phase is successful, collision avoidance is *not* successful if every CD&R phase is *not* successful. In either problem, each phase can be supported by a different set of components which may have different failure rates in different phases.

Based on the structural similarity, one approach for analyzing the proposed form of DET is to apply an existing solution method for the PMS problem. One such method is the binary decision diagram methodology for phased-mission systems (PMS-BDD) [Zang 1999] that is applied to analyze the proposed DET framework. This section consists of three sub-sections; Section 3.2.2.1 explains the BDD method to evaluate fault trees, and Section 3.2.2.2 describes an extension of the BDD method for PMS, then Section 3.2.2.3 discusses application of the PMS-BDD to the DET framework.

3.2.2.1 Binary Decision Diagram (BDD) Method

The BDD method is a method to solve a fault tree model for system reliability [Rauzy 1993]. The BDD method in the reliability analysis field has improved accuracy and efficiency in fault tree analysis [Sinnamon 1997]. According to Rauzy [2008], the BDD method changes the analyzing fault trees process significantly: 1) minimal cutsets are not necessary to evaluate a fault tree, 2) BDD provides the exact result of the top-event probability; but it also has a disadvantage that the size of BDD can increase exponentially as the worst case.

A BDD is a directed acyclic graph based on Shannon's decomposition of a Boolean function. A BDD is composed of terminal nodes which indicate system success (value 0) or system failure (value 1) and non-terminal nodes corresponding to basic events of a fault tree. Each non-terminal node has two out-branches: One is called the 0-branch representing the non-occurrence of a basic event (working state). The other is called the 1-branch representing the occurrence of the basic event (failed state). The BDD method converts a fault tree to a binary decision diagram encoding an if-then-else (**ite**) structure [Sinnamon 1997]. '**ite**(*x, f1, f2*)' means that **if** *x* is true, **then** consider function *f1*, **else** consider function *f2*, where *x* is a Boolean variable. Figure 14 illustrates a procedure to convert a fault tree to a BDD [Sinnamon 1997].

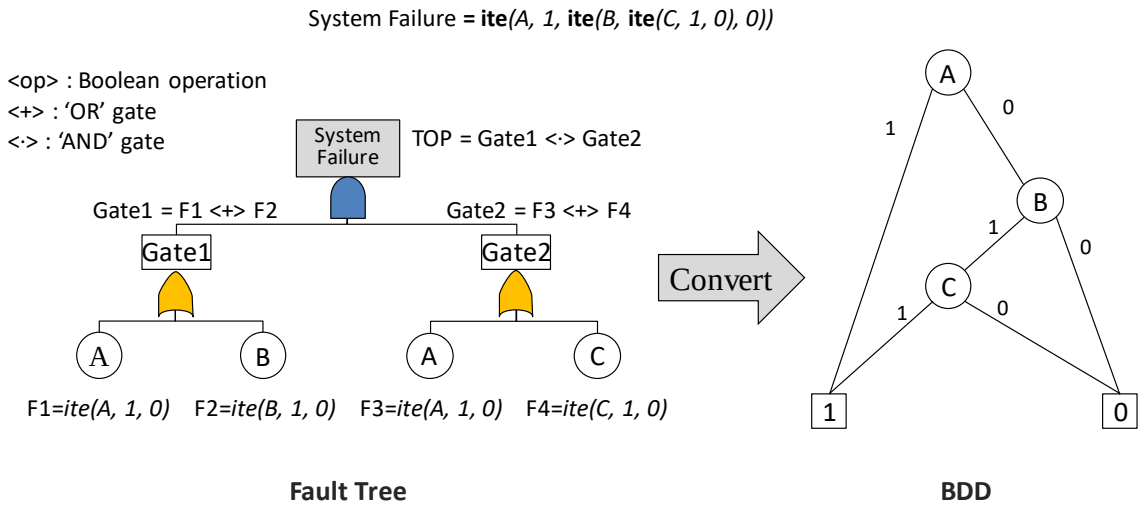


Figure 14 Conversion of fault tree to BDD

The first step of the conversion procedure is to assign each basic event in the fault tree the **ite** structure, **ite**(basic event name, 1, 0), which means that **if** the basic event occurs, **then** the system fails, **else** the system works. These structures are then combined in a bottom-up manner via the following operation rules (where an OR operation is denoted by <+> and an AND operation is denoted by <·>):

- For event A > B let J= **ite**(*A*, *S1*, *S2*) and H= **ite**(*B*, *U1*, *U2*);

  then J <op> H = **ite**(*A*, *S1* <op> *H*, *S2* <op> *H*)

- If A=B, i.e., let J= **ite**(*A*, *S1*, *S2*) and H= **ite**(*A*, *U1*, *U2*);

  then J <op> H = **ite**(*A*, *S1* <op> *U1*, *S2* <op> *U2*)

- 1 <·> H = H, 0 <·> H = 0, 1 <+> H = 1, 0 <+> H = H

Examples of the operation rules applied in Figure 14 are as follows:

- Gate1 = F1<+>F2 = **ite**(*A*, *1*, *0*) <+> **ite**(*B*, *1*, *0*) = **ite**(*A*, *1*, **ite**(*B*, *1*, *0*))
- Gate2 = F3<+>F4 = **ite**(*A*, *1*, *0*) <+> **ite**(*C*, *1*, *0*) = **ite**(*A*, *1*, **ite**(*C*, *1*, *0*))

- Top Event = Gate1 <·> Gate2

  = **ite**(*A, 1*, **ite**(*B, 1, 0*)) <·> **ite**(*A, 1*, **ite**(*C, 1, 0*))

  = **ite**(*A, 1*, **ite**(*B*, **ite**(*C, 1*, *0*), *0*))

The final ite structure for the top event represents the BDD of the fault tree (Figure 14, right). In a BDD, paths from the top event to a terminal node with a "1" represent the conditions for occurrences of the top event. For example, in Figure 14, the occurrence of event A will cause the top event to occur. In order to evaluate the probability of the top event in a fault tree, all disjoint paths leading to a terminal node with a "1" need to be tracked, i.e., {A}, {non A, B, C} in Figure 14. Secondly, the probability of each disjoint path is computed by multiplication of the probabilities of the basic events failure or success in the path. For example, the probability of the path {non A, B, C} is multiplication of the probability of non-occurrence of event A and the probability of event B and C. Lastly, the probability of the top event occurrence is obtained by summing the probabilities of all disjoint paths in the BDD [Andrews 2000].

3.2.2.2 PMS-BDD Method

PMS is a system that consists of multiple phases to accomplish a mission. One unique feature of PMS is phase dependency of components (i.e., a component failed during a phase remains at failed state during all later phases). Zang [1999] proposed PMS-BDD to accommodate phase dependency in BDD method using the phase algebra, which is a set of rules combining component states ('on' and 'off') across phases. A special BDD operation, called phase-dependent operation (PDO), specifically deals with the phase algebra. There are two classes of PDO; 1) Forward PDO: The order of variables

is the same as the phase order, 2) Backward PDO: The order of variables is the reverse of the phase order. In terms of size of the final BDD, backward PDO generates a smaller BDD so that backward PDO is computationally more efficient [Zang 1999]. Backward PDO is as follows:

- For phase $i < j$ let component 'A' used in both phases i and j,

  and let E_i= **ite**(A_i, *G1*, *G2*) and E_j= **ite**(A_j, *H1*, *H2*);

  then E_i <op> E_j = **ite**(A_j, E_i <op> *H1*, *G2* <op> *H2*)

In addition to PDO, a special evaluation rule needs to be applied for the 1-edge linking the variables of the same components because A_i and A_j are phase-dependent. Final equation for that is as follows (see [Zang 1999] for detailed derivation):

- Let G= **ite**(A_j, *G1*, *G2*) and G1= **ite**(A_i, *H1*, *H2*);

  then $\Pr\{G = 1\} = \Pr\{G1 = 1\} + (1-\Pr\{A_j = 1\})\cdot(\Pr\{G2 = 1\}-\Pr\{H2 = 1\})$

3.2.2.3 Application of PMS-BDD to DET

The basic idea is to convert the DET model for collision avoidance to a PMS fault tree, and then solve the PMS fault tree using the PMS-BDD method. This approach is relatively simple to apply to solve the proposed DET framework even though the PMS-BDD implementation itself requires several steps as explained in the two previous sections.

The PMS-BDD algorithm in this research is given by the following logic and is illustrated in Figure 15:

1. Create a fault tree where the top event is a collision event, below which is an AND gate combining the failure of each CD&R phase (Figure 15).

2. Each CD&R phase fails if either there is a component based failure, as defined by the supporting fault trees in the original system description, if the CD&R system fails to detect and resolve a conflict (an algorithmic failure), or if the pilot fails to execute the resolution in a timely manner.
3. Apply the standard PMS-BDD method to compute the overall collision probability that all CD&R phases fail.

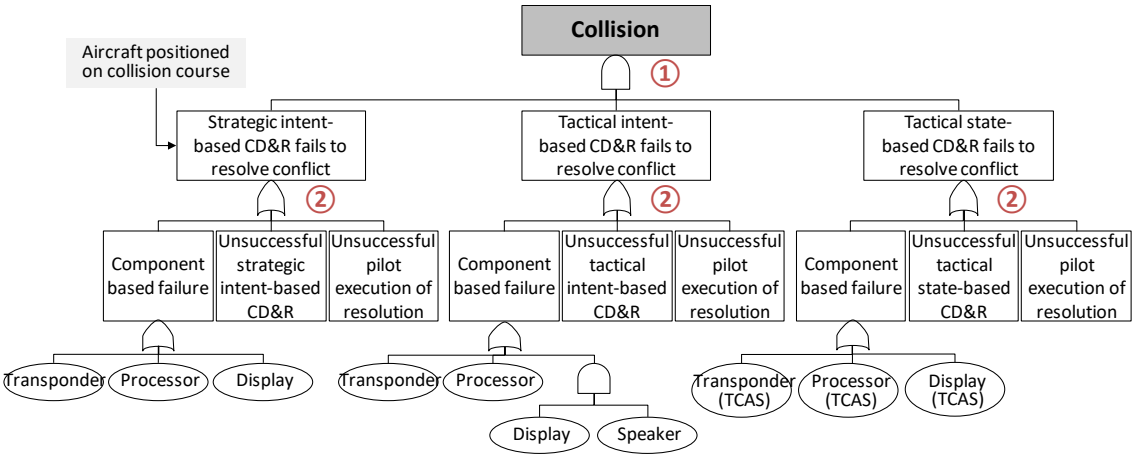


Figure 15 PMS-BDD approach applied to example problem

Note that this solution method introduces several structural differences in the underlying model assumptions. First, the PMS-BDD approach relaxes the assumption that components can only fail at the beginning of the analysis time horizon; now, components can fail at the beginning of each CD&R *phase*. Second, the fault tree in Figure 15 implicitly models the operational failures of detecting the conflict (e.g., "unsuccessful strategic intent-based CD&R") and executing the resolution ("unsuccessful pilot execution of resolution") as independent parallel events. In reality, these two events

occur *in sequence* – first, the CD&R system detects the conflict and provides a resolution to the pilot, and then the pilot executes the resolution. If the CD&R system is late to provide the resolution, there is less time for the pilot to successfully execute the resolution. These dynamics are captured in the sub-tree model of Figure 11 and in the other solution methods described in the research, but not by the PMS-BDD approach. A related limitation is that while the other methods (e.g., the cDET method) can easily generalize the dynamic-event tree logic within each sub-tree – for example, by adding logic to incorporate a backup air-traffic controller who may override the automated CD&R system –the ability to generalize the structure of the fault tree in Figure 15 is more limited.

3.2.3 Solution Method 3: cDET with PMS-BDD (cDET-PMS)

The cDET-PMS approach is a combination of the two approaches, the conditional dynamic event tree and the binary decision diagram methodology for phased-mission systems, explained in the previous sections. The basic concept of the cDET-PMS approach is to follow the computational logic of the cDET approach, but to use the PMS-BDD method to relax the assumption that components only fail at the beginning of the analysis horizon. Instead, they are assumed to fail at the beginning of each *phase*. Because components may be working in one phase, but may fail in another phase, the logic of PMS-BDD must be used.

The computation steps of the cDET-PMS method are below. The main difference between this method and the cDET method is Step 2b, where the BDD approach is

extended to PMS systems. The PMS-BDD method correctly accounts for failures of components that may occur in different phases of the analysis period.

1. [Same as cDET method] Create 2^r DETs, where r is the number of CD&R systems or phases. Calculate the conditional probability o_i associated with DET i, $i = 1, \ldots, 2^r$.
2. For each combination of CD&R states ($i = 1, \ldots, 2^r$), compute the probability q_i of state i as follows:
    a. [Same as cDET method] Create a *combined* fault tree where the top event is state combination i of the CD&R systems.
    b. Convert the combined fault tree to a PMS-BDD. Evaluate the PMS-BDD to obtain the probability q_i. The PMS-BDD method accounts for the phase dependency of components (i.e., a component that fails during one phase remains in a failed state during all later phases). The PMS-BDD method considers which phase a component accommodates phase dependency using the phase algebra, which is a set of rules combining component states ('failed' and 'working') across phases. By the PMS-BDD, the assumption on the timing of component failures is relaxed to that components can fail at the beginning of each phase in the cDET-PMS method compared to that components assume to fail at the beginning of the analysis in the cDET approach.
3. [Same as cDET method] The overall collision probability is the weighted sum of the conditional collision probabilities ($\Sigma_i\, q_i * o_i$).

Table 5 shows the computation steps of the cDET-PMS method as applied to the example problem. (Numerical values for the model parameters are given in the next section.) The example has three CD&R systems, so 8 (= 2^3) DETs are constructed (see top of Figure 12). For example, DET-8 corresponds to the case in which *no* CD&R systems are available. From Table 5, the associated conditional collision probability is 1. That is, if two aircraft are on a collision course, there is nothing to prevent the collision, so the conditional collision probability is 1. In contrast, DET-1 corresponds to the case in which all CD&R systems are available. This case has the lowest conditional collision probability, 2.46E-6. A collision in this case would be due to failures of the CD&R systems to detect the conflicts (e.g., due to noise in the trajectory predictions) and/or failures of the pilot to respond to resolutions.

Table 5 Example computation of cDET-PMS method

DET (i)	System Availability (0-unavailable, 1-available)			① Conditional Collision Probability (o_i)	② Weight Probability (q_i)	③ Collision Probability of DET ($q_i * o_i$)
	SICDR	TICDR	TSCDR			
1	1	1	1	2.46E-06	9.93E-01	2.45E-06
2	1	1	0	1.76E-04	3.99E-03	7.01E-07
3	1	0	1	1.06E-04	6.64E-04	7.02E-08
4	1	0	0	7.55E-03	2.67E-06	2.01E-08
5	0	1	1	3.26E-04	8.28E-04	2.70E-07
6	0	1	0	2.33E-02	3.33E-06	7.75E-08
7	0	0	1	1.40E-02	1.66E-03	2.33E-05
8	0	0	0	1.00	6.68E-06	6.68E-06
Overall Collision Probability ($\Sigma_i q_i * o_i$)				3.357E-05		

The weight probabilities (column ②) are computed by applying the PMS-BDD method to the fault trees associated with each DET. DET-1 has the highest weight probability (as expected), since the CD&R systems are available most of the time due to the high reliability of the system components. DET-8 usually has the lowest weight probability, corresponding to the most component failures. In the example problem, however, some other DETs have a lower weight probability than DET-8, because all components supporting SICDR system are also supporting TICDR system with an additional component so that it is less frequent that SICDR system fails while TICDR system is available (DET-6) than both SICDR and TICDR system fail (DET-8). The weight probabilities can sometimes be zero in situations where particular combinations of available CD&R systems are not possible. This can occur because components are non-repairable. As an example, assume that a CD&R system operates different conflict detection algorithms in a different time horizon, then the later phase always fails if the system is unavailable at the former phase.

The overall collision probability is the weighted sum of the conditional collision probabilities weighted by the probabilities of each DET being used. The last column of Table 5 is computed by multiplying conditional collision probability and weight probability for each DET, which shows contribution of each DET to the overall collision probability. DET-7, where SICDR and TICDR systems are unavailable while tactical state-based CD&R (TSCDR) system is working, contributes the most on collision risk, which takes more than 65% of the risk. The most contributing DET on collision risk varies depending on the assumed numerical values for model parameters.

3.2.4 Solution Method 4: Simulation

The last method to compute the collision probability through the proposed DET framework is a simulation, which can be used to approximate the true result. The cases where a collision occurs through the DET framework of the example problem are described verbally as follows:

- A collision occurs if all CD&R phases fail.
- Each CD&R phase fails if each CD&R system is not available due to component-based failure before the CD&R system function successfully detects and resolves a conflict or two events, that each CD&R system function detects and resolves a conflict and that pilots correctly execute a resolution, are not completed in a given time period.

From the verbal descriptions, an analytical equation for collision scenarios of the DET framework may be derived for a simple case, where, for example, a single component supports all CD&R systems. However, it is almost impossible to see the simple case in safety systems. Simulation is one way to approximate the result in the case that the analytical equation is extremely difficult to derive. Figure 16 shows a flow diagram of simulation to compute the collision probability through the DET framework. The simulation follows the verbal descriptions for the collision scenarios. The simulation generates a set of all random times such as the time for each CD&R system to detect a conflict and provide a resolution, and the time for the pilot to correctly execute a resolution in each CD&R phase. Next, it determines whether or not a CD&R phase fails to avoid a conflict due to either operation function failure or component-based failure via

evaluating fault trees until all CD&R phases fail to resolve a conflict. The method provides an approximation of the true result, and may need a lot of computation time to achieve a certain level of confidence interval for rare events like mid-air collision. (Note: Simulation is conducted with 30 runs, and each of which has from millions to billions replications, depending on the assumed numerical values for model parameters, to achieve a small coefficient of variation such as 0.05.)

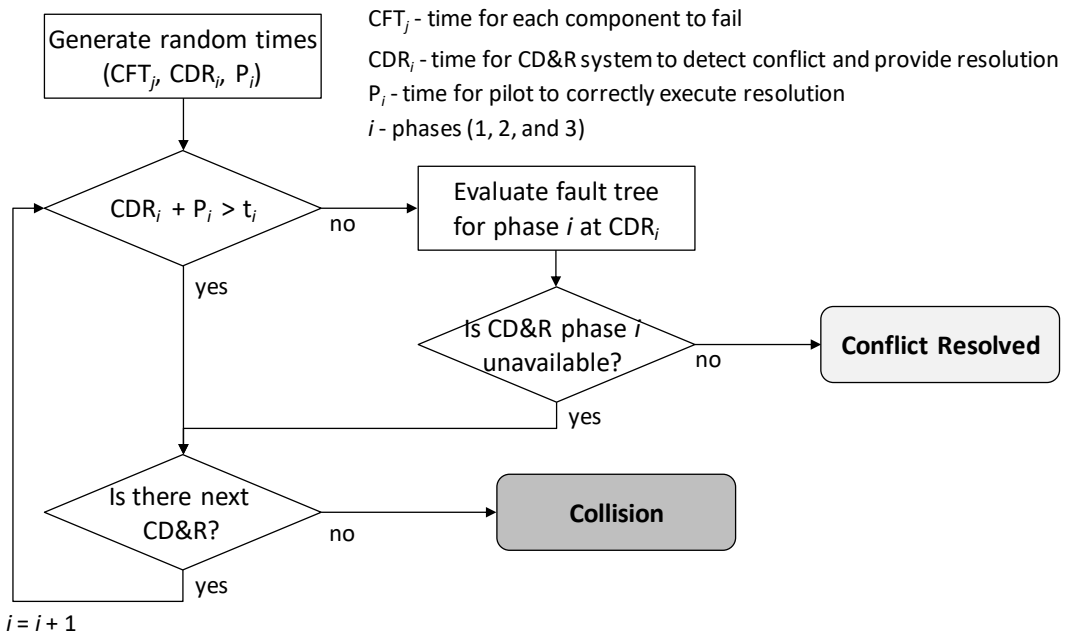


Figure 16 Flow diagram of simulation for example problem

3.3 Numerical Results for Example Problem

This section provides numerical results of the example problem (Figure 11) and comparison between the results from different methodologies, cDET, PMS-BDD, cDET-PMS, and simulation. The different methods use different assumptions, which are

summarized in Table 6. A major difference between the methods is the assumption about when the components can fail. The cDET assumes that components can only fail at the beginning of the analysis. Both the PMS-BDD and the cDET-PMS assume that components may fail at the beginning of each CD&R phase, while the PMS-BDD is not able to appropriately model a sequence of operational events within each phase (e.g., first the system detects a conflict, then the flight crew executes a resolution). The simulation allows components to fail at any time, thus to provide an approximation of the true result even though it can be computationally challenging for rare events.

Table 6 Assumptions and limitations for each method

Methodology	Method specific assumptions and limitations	Common assumptions
Conditional DET (Section 3.2.1)	- Components fail at the beginning of analysis	- Components may be used either in a single CD&R system or in multiple CD&R systems. - Components are unrepairable. - Each CD&R system and pilot not fails but succeeds at a given rate or time-varying rate with a probability function.
PMS-BDD (Section 3.2.2)	- Components fail at the beginning of each CD&R phase - Impossible to model sequence of events	
cDET-PMS (Section 3.2.3)	- Components fail at the beginning of each CD&R phase	
Simulation (Section 3.2.4)	- Each component fails in a random manner according to a time-varying rate. - Can be computationally expensive (Simulation)	

Various combinations of numerical values in component failure rates and rates for each CD&R system to successfully detect and resolve a conflict are assumed to calculate

the collision risk from each approach. Times for each CD&R system to activate are respectively assumed 8 min (Strategic intent-based CD&R), 3 min (Tactical intent-based CD&R), and 1 min (Tactical state-based CD&R) prior to a conflict. One second (i.e., 1s) is used for time step (Δt) for evaluating a dynamic event tree.

Table 7 summarizes the combinations of assumed numerical values for the example problem and associated result collision probabilities from each method. Three levels (high, medium and low) of numerical values for each parameter are assumed to see how each evaluation approach estimates the resulting collision probability in a different combination of various levels of parameters. (Note that the term of 'collision risk/probability' used in the research is actually a conditional collision risk/probability, given that two aircraft are on a collision course.)

Table 7 Collision risk of example problem (Case 1 - without pilot execution event)

Scenario	Component failure rate (/min)	CD&R detection rates (/min)	Collision probability			
			cDET	cDET-PMS	PMS-BDD	Simulation
1	1.67E-04	[2, 4, 10]	1.08E-05	6.77E-06	6.76E-06	7.23E-07
2	1.67E-04	[1, 2, 5]	3.19E-05	2.06E-05	1.94E-05	4.77E-06
3	1.67E-04	[0.5, 1, 2.5]	1.27E-03	1.18E-03	1.11E-03	1.01E-03
4	1.67E-05	[2, 4, 10]	1.21E-07	7.59E-08	7.45E-08	1.96E-08
5	1.67E-05	[0.5, 1, 2.5]	1.00E-03	9.94E-04	9.31E-04	9.22E-04
6	1.67E-03	[2, 4, 10]	1.04E-03	6.57E-04	6.57E-04	5.95E-05
7	1.67E-03	[0.5, 1, 2.5]	4.77E-03	3.57E-03	3.44E-03	2.02E-03

Figure 17 shows the result collision probabilities from each method by the scenarios in Table 7. The results from the simulation approach are approximation of the true collision risk given a combination of numerical values. Overall, the PMS-BDD approach estimates the collision risk better than the other two methods, the cDET and the cDET-PMS, even though the differences on the result between the three approaches are very little, mostly in a same order of magnitude. Accuracy of the estimated risk from the proposed methods varies by scenarios. In scenario 1, 2, 4, and 6, the evaluation methods over-estimate the collision risk in a different order of magnitude whereas the estimates in scenario 3, 5, and 7 are close to the true risk. A common parameter for the scenario 3, 5, and 7 is the low level of rates for each CD&R system to successfully detect and resolve a conflict, therefore it can be concluded that the methods estimate the collision risk better when CD&R system algorithms more likely fail to detect and resolve a conflict (i.e., poor algorithm performance) than when CD&R system algorithms perform detection and resolution function very well.

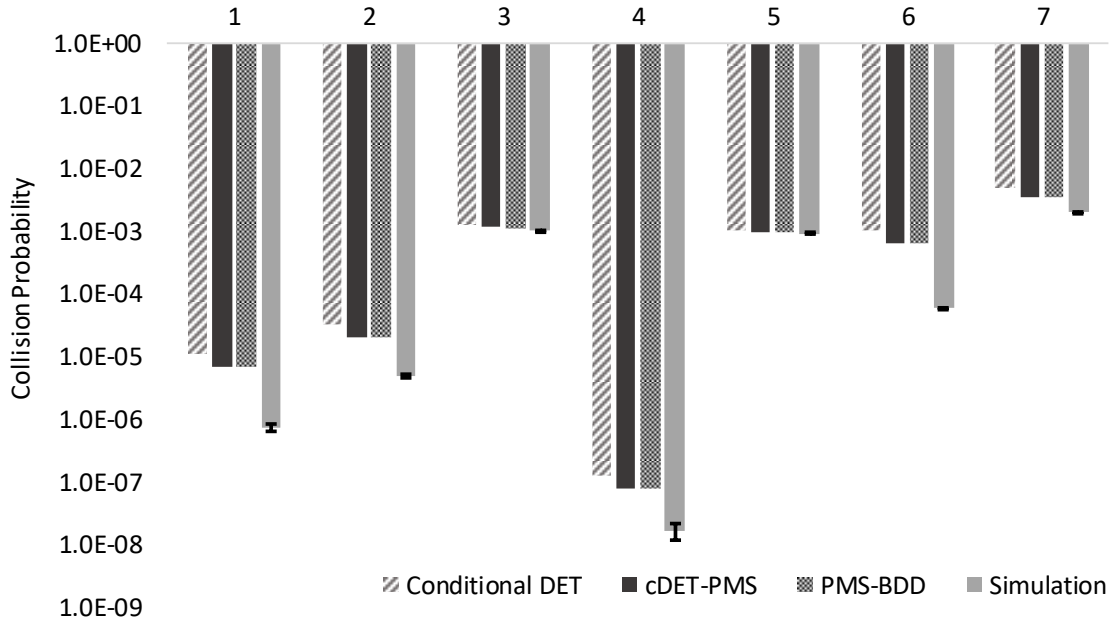


Figure 17 Comparison of collision probabilities between methods (Case 1)

Similar to Table 7 and Figure 17, Table 8 and Figure 18 show the result collision probabilities from each method given combinations of assumed numerical values. Difference between Table 7 (Case 1) and 8 (Case 2) is whether or not the pilot execution event exists in each CD&R phase. Rates for pilot to correctly execute a resolution provided by each CD&R system are simply assumed to be consistent throughout all the CD&R phases to only see how appropriately the proposed approaches model the event.

Figure 18 looks very similar to Figure 17, i.e., the cDET method and the cDET-PMS approach estimate the risk better in scenario 3, 5, and 7 than in the other scenarios. The result collision risk from the PMS-BDD approach, however, shows a different pattern. The estimated collision risk that is lower than the true risk are observed in some of the scenarios. This under-estimated collision probability is due to limitation of the

PMS-BDD approach to model a sequence of operational events within a CD&R phase correctly. The under-estimated result than the true risk is not acceptable from safety perspective since safety analysis must be conservative.

Table 8 Collision risk of example problem (Case 2 - with pilot execution event)

Scenario	Component failure rate (/min)	CD&R detection rates (/min)	Pilot correct execution rates (/min)	Collision probability			
				cDET	cDET-PMS	PMS-BDD	Simula-tion
1	1.67E-04	[2, 4, 10]	[10, 10, 10]	1.21E-05	7.58E-06	6.83E-06	8.01E-07
2	1.67E-04	[1, 2, 5]	[10, 10, 10]	5.16E-05	3.36E-05	1.95E-05	8.70E-06
3	1.67E-04	[0.5, 1, 2.5]	[10, 10, 10]	1.84E-03	1.73E-03	1.11E-03	1.54E-03
4	1.67E-05	[2, 4, 10]	[10, 10, 10]	2.51E-07	1.57E-07	8.21E-08	5.35E-08
5	1.67E-05	[0.5, 1, 2.5]	[10, 10, 10]	1.51E-03	1.50E-03	9.31E-04	1.42E-03
6	1.67E-03	[2, 4, 10]	[10, 10, 10]	1.06E-03	6.65E-04	6.58E-04	6.23E-05
7	1.67E-03	[0.5, 1, 2.5]	[10, 10, 10]	6.03E-03	4.58E-03	3.44E-03	2.73E-03

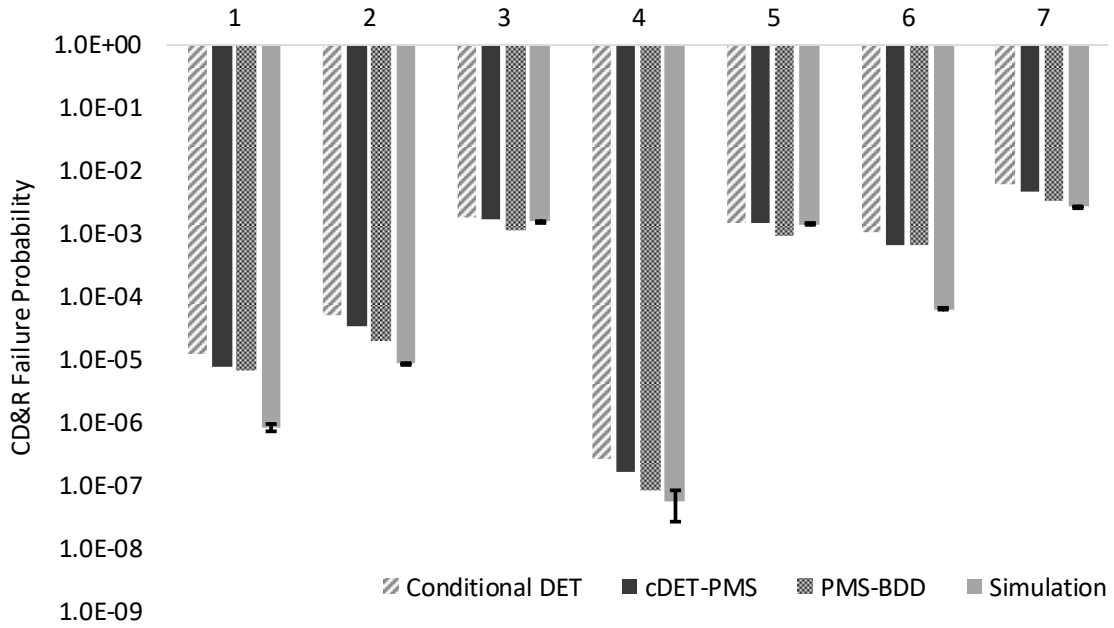


Figure 18 Comparison of collision probabilities between methods (Case 2)

Figure 19 illustrates performance of each method in different scenarios (scenario 1 and 3 in Table 8). Performances of each method are shown in two measures. The first measure is the computation time shown on y-axis in the figure. Computation of the methods except the simulation are quite fast (done in 1 second), and it does not depend on the parameter values (i.e., computation times are almost same in different scenarios), whereas the simulation takes a couple of hours (Scenario-3) to a few days (Scenario-1) depending on the assumed numerical values on the parameters.

The second measure of the performance is the relative risk estimate compared to the simulation result shown on x-axis, where the simulation result always equals to one. If a method has relative risk estimates that are greater than one, the method over-estimates the risk, so it is erroring on the correct side of safety. In addition, the larger an x-axis value of a method is, the more the method over-estimates, i.e., the less accurate the estimate is. The relative risk estimates of each method are quite different in different scenarios as well as between the methods. Relative risk estimates of all methods for Scenario-3 are very close to one, while, for Scenario-1, the relative risk estimates are all greater than 10 meaning that the all methods over-estimates the risk by more than a 10-fold. The methods discussed in this research estimate the collision risk relatively well for Scenario-3, where the CD&R algorithms likely fail to detect and resolve a conflict, i.e., success probabilities (rates) of the CD&R systems are small. In terms of estimation methods, the cDET-PMS approach shows smaller relative risk estimates (i.e., estimates the risk better) in both scenarios than the cDET method, while, as stated in Figure 18, the PMS-BDD under-estimates the risk, which is not allowed in the safety analysis.

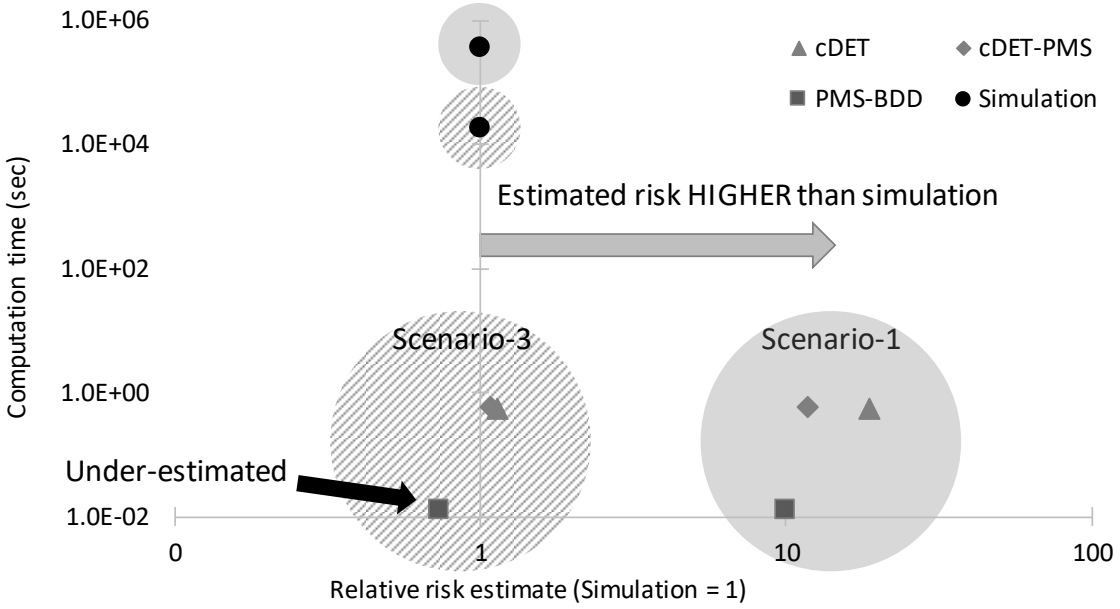


Figure 19 Performance comparison between methods

3.4 Summary

This chapter presented a generic form of dynamic event tree (DET) that consists of three levels, a high-level dynamic event tree, a generic sub-tree, and fault trees, to model the aircraft mid-air collision scenarios. Several approaches, which are existing methods and a combination of them, are applied to evaluate an example problem of the DET framework. The example problem consists of three phases of CD&R systems to detect and resolve a conflict, each of which performs its function in a specified time horizon. A set of components with a specific configuration supports each CD&R system in the example problem. The problem models each CD&R phase in two cases, with one successive event and with two successive events. The first case models an event that

CD&R systems detect and resolve a conflict while the second case additionally includes an event that pilot correctly execute the resolution.

Each method has different assumption and limitation on the timing of component-based system failures and modeling a sequence of events in sub-trees. Table 9 summarizes computation steps and assumptions/limitations of each method. The PMS-BDD approach specifically has a significant limitation to model a sequence of events that causes under-estimation of the true risk. The proposed evaluating approaches (i.e., cDET and cDET-PMS) perform differently in various combinations of numerical values. They estimate the collision risk very well in case that the CD&R system algorithmic failures very likely occur in the first two conflict detection and resolution (CD&R) phases, while they over-estimate the risk in a different order of magnitude for the other case.

The proposed DET framework and evaluating approaches have benefit of creating or modifying a model and of evaluation of the new model. In a typical system design phase, for example, adding a redundant component, replacing a better component, or even a new architecture needs to be evaluated to choose the best design of a system that meets requirements of system reliability/safety. The DET approach proposed in this dissertation can be used to evaluate design alternatives of a system readily with a reasonable fidelity.

Table 9 Summary of the methods

Method	Algorithm	Assumption / Limitation
cDET	i) Enumerate all 2^m (m = # of components) combinations of component states and compute the probability p_i of each state combination ii) Determine availabilities of CD&R systems for each state combination and compute the joint probability q_j (j = 1, …, 2^n, n = # of CD&R systems) for each combination of available CD&R systems iii) Generate at most 2^n DETs based on CD&R system availabilities and evaluate the conditional collision probability o_j given DET-j iv) Total collision risk = $\Sigma_j\, q_j * o_j$	- Components assume to fail at the beginning of analysis. - Enumeration of combinations of components state may require a lot of computation time.
PMS-BDD	i) Create a fault tree with the top level (collision) by combining fault trees for failure of each CD&R phase with 'AND' gate ii) Modify the supporting fault trees to include operational failure events iii) Apply PMS-BDD to compute total collision risk	- Components assume to fail at the beginning of each CD&R system operation. - A sequence of operational events, such as detection of a conflict and correct execution of a resolution by pilot, is modeled parallelly.
cDET-PMS	i) Generate 2^n DETs based on CD&R system availabilities and evaluate the DETs for conditional collision probabilities (o_j) given DET-j ii) Create a fault tree for each DET combining fault tree and/or success tree of each CD&R system iii) Apply PMS-BDD to combined fault trees to calculate weight probability (q_j) for each DET being used iv) Total collision risk = $\Sigma_j\, q_j * o_j$	- Components assume to fail at the beginning of each CD&R system operation.

CHAPTER 4: CASE STUDIES: MODELING COLLISION RISK BETWEEN VARIOUS AIRCRAFT TYPES

The current NAS is expected to experience difficulty in accommodating increasing aircraft demand and diversity of aircraft. In order to resolve the predicted problems, additional automated separation assurance systems are needed, but also a new system architecture for the NAS may be required. Two comprehensively researched concepts of the future NAS operation are Autonomous Flight Management (AFM) [Wing 2011] and the Advanced Airspace Concept (AAC) [Erzberger 2001]. The major difference between AFM and AAC is that AFM distributes responsibility for maintaining safe separation to operators in the air, while AAC has a central system on the ground to provide separation assurance. Wieland [2017] recently proposed a system architecture where the capacity of each sector is determined by the collision risk between aircraft within the sector.

This chapter describes case studies for collision risk between different types of aircraft and/or collision avoidance capabilities (e.g., between a manned aircraft and an Unmanned Aircraft System (UAS)). The case studies are developed in a future NAS environment such as AFM and AAC using the proposed dynamic event tree (DET) framework. In addition, sensitivity analysis on the model parameters including component failure probabilities, maximum detection range of the sensors, and collision geometries are conducted.

4.1 Case Study-1: Autonomous Flight Management (AFM)

4.1.1 Concept of AFM Operations

The AFM concept is originally from the free flight concept of operation, in which operators are free to choose their flight trajectory in real time, which is expected to be the most ultimate environment for the future NAS. The "self-separation", i.e., maintaining a safe distance to all other aircraft by the pilot themselves, is a key concept for a safe free flight environment as well as a concept to be accomplished under AFM operation to prevent a mid-air collision.

Based on Wing [2011], an aircraft operating in the AFM concept has three safety layers that sequentially operate to prevent a mid-air collision. These systems are a strategic intent-based (SI) CD&R system, a tactical intent-based (TI) system, and a tactical state-based (TS) system. The first two safety layers (SI and TI) are implemented via an Airborne Separation Assistance System (ASAS), which is a software automation system onboard the aircraft that performs conflict detection, resolution, and prevention functions. Both systems use state and intent information of other aircraft to suggest resolutions. The final safety layer is the Traffic Alert and Collision Avoidance System (TCAS), which uses state information of the two aircraft to avoid an imminent collision. The three systems are assumed to operate in the following respective time intervals in this case study: Between 8 min and 3 min prior to a conflict, between 3 min and 1 min prior to a conflict, and within 1 min prior to a conflict. Times for each CD&R system to activate are chosen to provide an acceptable trade-off between the benefits of alerting as early as possible and the costs of false alarms [Erzberger 2012]. All commercial (i.e.,

manned) aircraft are predicted to be properly equipped to meet the requirements of AFM flights in the case study.

4.1.2 Conflict Detection and Resolution (CD&R) for UAS

This case study considers the hypothetical introduction of unmanned aircraft systems (UAS) into the AFM framework. In the future NAS, various types of UAS may have different conflict detection and resolution systems onboard. Unlike manned (commercial) aircraft, UAS may not be equipped with all three CD&R systems (i.e., SI, TI, and TS CD&R) due to cost, weight, capacity, or power restrictions.

Table 10 provides a summary of example sensors for UAS in terms of type, information that can be obtained, detection range, and weather conditions in which a sensor operates [Lacher 2007; Yu 2015; Fasano 2016]. Mode A/C or Mode S transponders, TCAS and ADS-B are cooperative sensors because they transmit their position information either by interrogation or on their own. The other sensors are non-cooperative sensors. An aircraft equipped only with a non-cooperative sensor can acquire information of other nearby flights, but the other flights do not have position information of that aircraft. Radar and LIDAR systems locate nearby aircraft by deploying energy, e.g., emitting an electronic pulse, while electro-optical (EO) systems and acoustic systems sense aircraft passively (e.g., by listening to sound made by aircraft). Active non-cooperative sensors require more energy so are typically bigger and heavier. Passive non-cooperative sensors are smaller and lighter, but they do not provide range information directly.

Table 10 Summary of example sensor technologies for UAS

Sensor	Type	Information Acquired	Detection Range	Weather Condition
Mode A/C or S Transponder	Cooperative	Range, Altitude	160 km	VMC / IMC
ADS-B	Cooperative	Position, Altitude, Velocity	240 km	VMC / IMC
TCAS	Cooperative	Range, Altitude	160 km	VMC / IMC
Radar	Non-Cooperative (Active)	Range, Bearing (Azimuth, Elevation)	35 km	VMC / IMC
LIDAR	Non-Cooperative (Active)	Range	3 km	VMC / IMC
Electro-Optical (EO) system	Non-Cooperative (Passive)	Azimuth, Elevation	20 km	VMC
Acoustic system	Non-Cooperative (Passive)	Azimuth, Elevation	10 km	VMC

Note: VMC-Visual Meteorological Conditions, IMC-Instrument Meteorological Conditions

In the case study, the manned aircraft is assumed to be AFM-equipped with three safety levels. But the unmanned aircraft is assumed to have only one safety layer, namely a non-cooperative tactical state-based CD&R system, with an onboard radar to acquire position information of other aircraft. The timings of these safety layers are illustrated in Figure 20. The time interval of the UAS safety phase (T_4) depends on the sensor range, speed of the aircraft, and conflict geometries. The unmanned aircraft is also assumed to have a Mode A/C transponder. This is assumed since the CD&R systems on the manned aircraft require position information of the unmanned aircraft, which a cooperative sensor provides either directly or through ground systems.

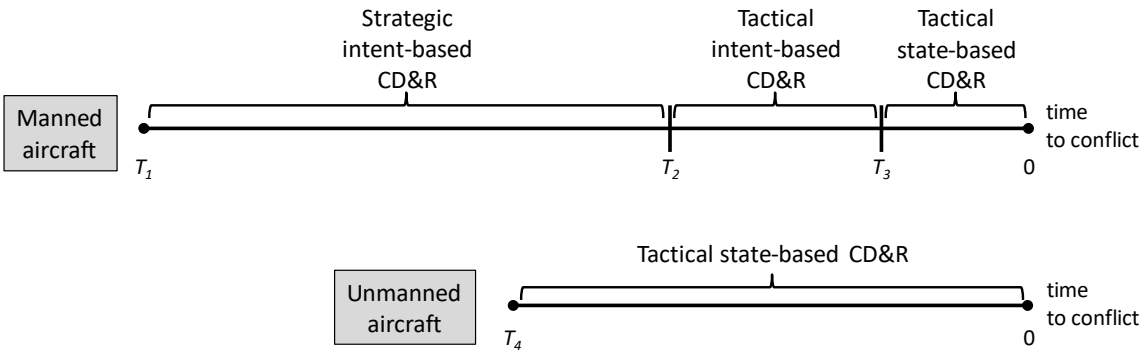


Figure 20 CD&R phases for the case study

An assumed concept of operation of the CD&R system on the unmanned aircraft is as follows: The onboard radar provides relative position information of nearby aircraft. An onboard CD&R processor detects potential conflicts using this information and determines appropriate resolutions. Resolutions are transmitted to a remote pilot via a command and control link. The pilot of the UAS is informed of suggested resolutions aurally through a speaker and visually through a display. The pilot chooses a resolution and gives a command to the UAS to execute the resolution to avoid the predicted conflict.

4.1.3 Fault Trees for CD&R Systems

In order for the CD&R systems to operate, several sub-systems/components must be working. A fault tree for each CD&R system is given to show the failure logic between components and the CD&R functionality. These fault trees are based on the AFM concept in Wing [2010] for the manned aircraft combined with the assumed concept of operation for the CD&R system on the unmanned aircraft. The fault trees for a pair of manned aircraft in AFM flight would be different.

Figure 21 depicts the failure logic of the strategic intent-based (SI) system on the manned aircraft. The SI system can fail either due to the failure of components supporting the system or due to a surveillance failure. On the left side of the figure, the SI system is supported by a processor that runs the conflict detection and resolution algorithm and a display that visually provides conflict information and resolution to the pilot. The failure considered here is a physical failure of the processor. The system can also fail algorithmically (i.e., failure to detect a conflict due to uncertainties in surveillance information), and this is considered later in the chapter.

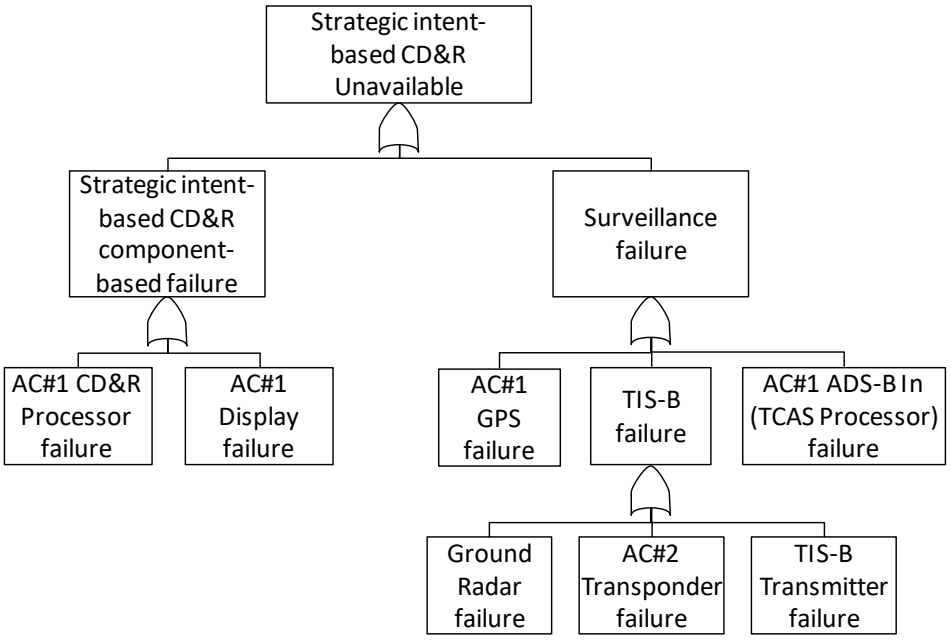


Figure 21 Supporting fault tree for strategic intent-based CD&R system (manned aircraft)

On the right side of the figure, a surveillance failure occurs when the manned aircraft (shown as AC#1) cannot locate either itself or the other aircraft (i.e., the

unmanned aircraft shown as AC#2). The manned aircraft's own location comes from a Global Positioning System (GPS) that is assumed to collect position, velocity, and heading information (from the Global Navigation Satellite System, GNSS) and altitude information from the altimeter. It passes this information to the CD&R processor.

According to Wing [2011], Automatic Dependent Surveillance-Broadcast (ADS-B) is the primary source of surveillance information for the manned aircraft. However, since the unmanned aircraft is assumed *not* to have an ADS-B system, the Traffic Information Service Broadcast (TIS-B) system is used to acquire the location of the unmanned aircraft. In the AFM concept, TIS-B is a ground-based backup system that provides surveillance information of non-ADS-B equipped aircraft. Ground radar locates the unmanned aircraft by interrogating the transponder onboard. A transmitter sends the surveillance information to the manned aircraft in the form of an ADS-B Out message. The ADS-B In system on the manned aircraft receives the message and provides surveillance information to the CD&R systems and/or flight crew. The ADS-B In function is currently implemented in the TCAS processor on most commercial aircraft [Richards 2010].

The tactical intent-based (TI) system begins to operate 3 minutes prior to a potential collision. Figure 22 shows the failure logic of the TI system, which is similar to the logic of the SI system. Failures of supporting components or a loss of location of any aircraft can lead to failure of the TI system. The TI system uses the same source for surveillance information as the SI system does, which is the ground-based TIS-B system.

A key difference is that the TI system uses two means to alert the pilot of conflict detection and resolution – namely, a display and speaker.

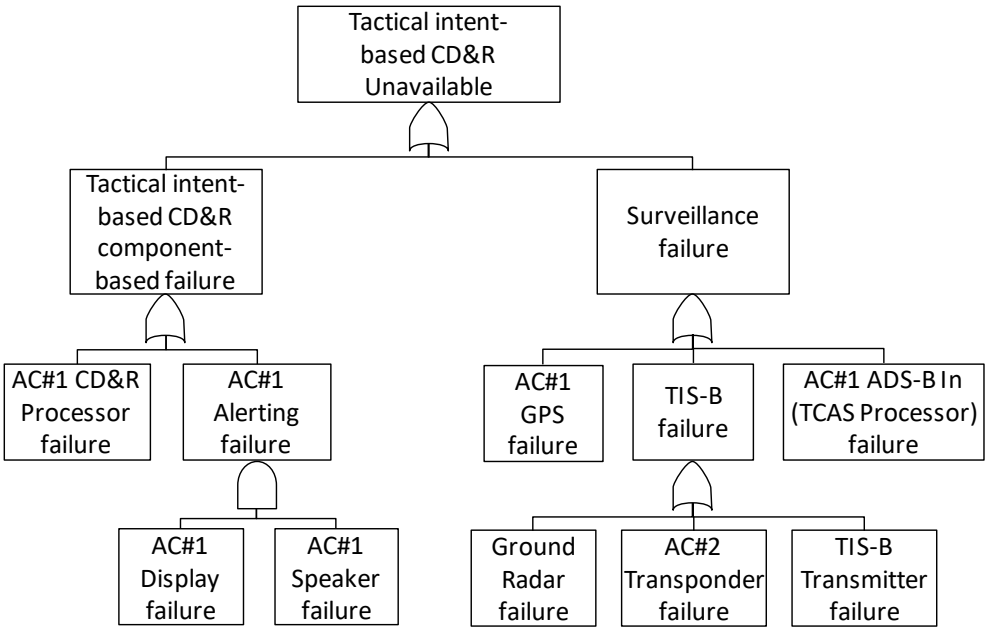


Figure 22 Supporting fault tree for tactical intent-based CD&R system (manned)

The tactical state-based (TS) system is the last CD&R system for the manned aircraft to avoid a midair collision. This system is assumed to be the Traffic Alert and Collision Avoidance System (TCAS). According to FAA [2011], TCAS has a requirement to provide reliable surveillance out to 14 nautical miles (nmi). In this case study, 1 minute is chosen as the activation time of TCAS, which is enough to account for a closing speed up to 840 knots in a head-on collision. Unlike the previous CD&R systems, TCAS obtains surveillance information by direct interrogation of the transponder on the other aircraft [FAA 2011]. Thus TCAS can fail if the transponder on

the target aircraft fails. TCAS can also fail if the transponder on the own aircraft fails, since the TCAS processor is connected to the Mode S transponder and is not available if the transponder fails [FAA 2011]. In addition, the TCAS display and speaker support TCAS to perform its function as depicted in Figure 23.

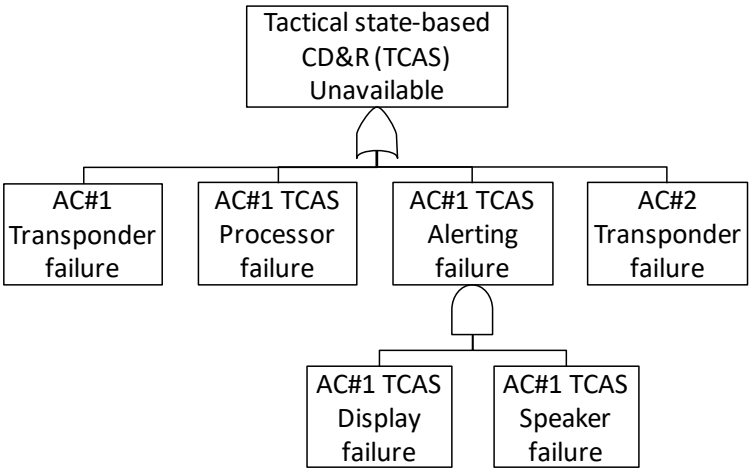


Figure 23 Supporting fault tree for tactical state-based CD&R system (TCAS, manned)

Figure 24 shows the fault tree supporting the CD&R system for the unmanned aircraft. Similar to the CD&R systems for the manned aircraft, the CD&R system for the unmanned aircraft is assumed to be configured with a processor, means of alerting (visual and aural), and sensors that provide state information of the other aircraft. An additional component is a command and control link through which the remote pilot receives resolutions and can direct the aircraft.

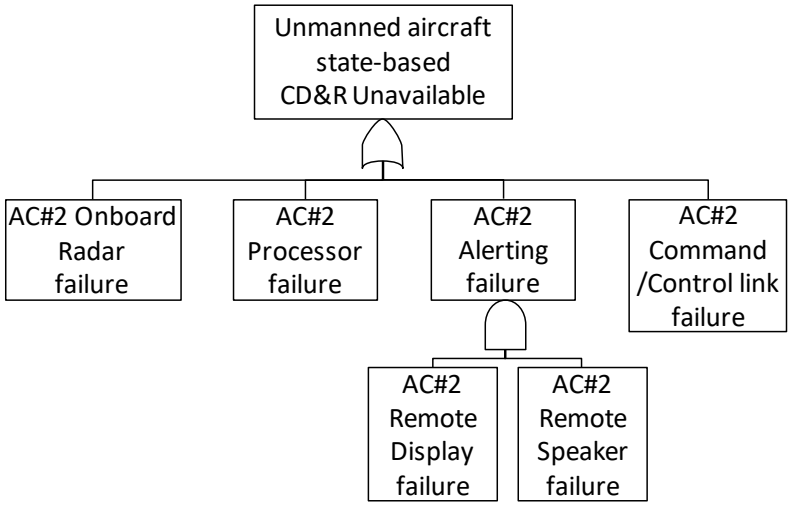


Figure 24 Supporting fault tree for tactical state-based CD&R system (unmanned)

Table 11 summarizes components that support the CD&R systems for both aircraft and their failure rates. Some of the values are assumed, and others are obtained from the literature.

Table 11 Parameters in fault trees for CD&R systems

Component	Failure Rate (/hr)	Description
CD&R Processor	6.25E-5 [Hemm 2009]	- Running CD&R logic using information from ADS-B In, GPS, etc.
Display	6.25E-5 [Hemm 2009]	- Providing traffic/conflict information and resolution trajectory to flight crew
Speaker	6.25E-5 (assumed)	- Providing aural alert to draw flight crew attention to conflicts
GPS	5.0E-5 [Hemm 2001]	- Providing position/velocity, altitude, heading, and air-ground status information
Transponder	8.33E-5 [Hemm 2009]	- Mode C / Mode S transponder including antennas - Providing aircraft state information as response of interrogation
TIS-B transmitter	1.0E-4 [Hemm 2001]	- Providing traffic information from ground to air

Ground radar	2.0E-5 [Hemm 2009]	- Secondary surveillance radar - Gathering traffic information
TCAS Processor / ADS-B In	6.25E-5 [Hemm 2009]	- Antennas included - Transmitting interrogation to / receiving replies from other aircraft - Running TCAS logic - Receiving ADS-B messages from other aircraft or ground facilities - Providing information to flight crew display and to CD&R processor
TCAS Display	6.25E-5 [Hemm 2009]	- Providing traffic/conflict information and resolution trajectory to flight crew
TCAS Speaker	6.25E-5 (assumed)	- Providing aural alert to draw flight crew attention to conflicts
Onboard radar	1.0E-4 (assumed)	- Gathering traffic information
Command/ Control link	1.0E-4 (assumed)	- Providing ability to communicate between aircraft and remote pilot - Providing ability for remote pilot to control aircraft

4.1.4 Algorithm Performance

In order for a conflict to be resolved, three steps need to be completed: 1) an algorithm of the CD&R system detects the conflict, 2) an algorithm of the CD&R system provides appropriate resolutions for the pilot to avoid a conflict, and 3) the pilot correctly executes the provided resolution.

Various studies have been conducted to develop autonomous CD&R algorithms. This research uses an analytic conflict-detection method from Paielli [1997] which gives the probability that a loss of separation ($\leq$ 5 nm) occurs when the system predicts a loss of separation given an assumption of level flights. Trajectory prediction errors are assumed to be normally distributed with a constant root mean square (rms) for the lateral position prediction error and a linearly growing rms in time for the longitudinal position

prediction error. The resulting probability for an actual loss of separation is a function of the time prior to the predicted loss of separation. It is also a function of other parameters such as speed of aircraft, size of the conflict zone, and the path-crossing angle. Figure 25 shows sample loss of separation probabilities for different path-crossing angles based on an implementation of the algorithm in Paielli [1997] (using 5 nmi as a conflict radius).

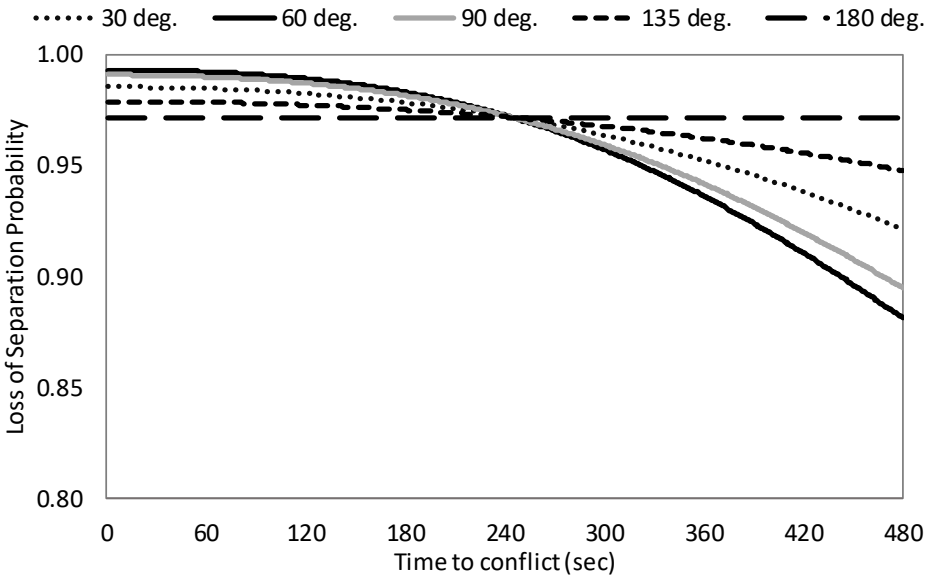


Figure 25 Loss of separation probabilities for different path-crossing angles

As a technical note, values in Figure 25 need to be converted to probabilities used in the DET model. The values in Figure 25 are *cumulative* probabilities, whereas the model uses probabilities associated with detecting conflicts in the next Δt seconds (see level-2 sub-tree in Figure 11). This can be obtained by converting the cumulative probability to an associated hazard rate function. For example, for a 90° path-crossing angle, at 480 seconds prior to a collision, the probability in Figure 25 is about 0.9. This is

interpreted as the cumulative probability of detecting the conflict some time prior to a collision. The associated hazard rate is $-[\ln(1 - 0.9)] / 480 \approx 0.0048$ / sec, meaning there is roughly a 0.0048 probability of detecting the conflict each second. Over 480 seconds, the probability of detecting the conflict yields the desired value of 0.9. Over an interval of Δt seconds, the probability of detecting the conflict is $1-\exp(-0.0048\Delta t)$ which is about $0.0048\Delta t$, assuming Δt is small.

This analysis assumes that the values in Figure 25 can be interpreted as the probability of detecting a conflict, given that a collision will occur. The model in Paielli [1997] gives something slightly different – the probability that a collision will occur given a conflict is detected. By Bayes' theorem, these are approximately the same, so long as the probability of detecting a collision is roughly the same as the probability of a collision (i.e., the detection algorithm is not biased high or low in terms of identifying collisions).

In order to determine the probabilities for the pilot to correctly execute a resolution provided by the CD&R system, results from Consiglio [2010], which assessed the performance of commercial pilots in human-in-the-loop simulation experiments, are used. In the literature, pilot response delays in a self-separation concept were measured when interacting with automated separation assurance tools on board. A CD&R tool was set to provide two different alerting levels depending on the time to a predicted conflict. One alerting mechanism was a display with a chime sound and the other was a display with an aural warning. Average response delays to the two different alerting levels were 32.4 and 20.6 seconds, which are assumed as the pilot response delays for the SI and TI

respectively. Assuming exponential distributions for the response times, these values are converted to pilot response rates for the first two CD&R phases, similar to the previous discussion of Figure 25. The pilot execution rate for the last CD&R phase is based on FAA [2011], where pilots are expected to respond to a TCAS Resolution Advisory in 5 seconds.

Several assumptions for the performance of the CD&R system on the unmanned aircraft are also made. It is assumed that the CD&R system on the unmanned aircraft successfully detects and resolves a conflict with a probability (or rate) that is 30% that of the manned aircraft, This is a time-varying value (e.g., see the conflict detection rate in Table 12). The performance of the remote pilot (i.e., the random time to execute a resolution) is assumed to be the same as for the first CD&R phase of the manned aircraft.

The activation time for the CD&R system of the unmanned aircraft is based on the detection range of the onboard sensors, the geometry of the conflict, and the speed of the two aircraft. Table 12 shows a summary of the parameters for algorithm performance at time *t* prior to a conflict, given a 90° of path-crossing angle. (Note: In this research, it is assumed that the CD&R systems always generates an appropriate resolution once the conflict is detected.)

Table 12 Parameters of CD&R system function and pilot behavior

Aircraft	CD&R Phase	Time to Conflict (min)	Conflict Detection Rate (/hr)	Pilot Execution Rate (/hr)
Manned	Strategic intent-based	8	17	111
		7.5	19	

	CD&R	7	22	
		6.5	25	
		6	28	
		5.5	33	
		5	38	
		4.5	45	
		4	54	
		3.5	65	
	Tactical intent-based CD&R	3	80	175
		2.5	100	
		2	130	
		1.5	179	
	Tactical state-based CD&R	1	276	720
		0.5	560	
Unmanned	Tactical state-based CD&R	2.5	30	111
		2	39	
		1.5	54	
		1.0	83	
		0.5	168	

4.1.5 Result & Sensitivity Analysis

This section provides numerical results and sensitivity analyses of the case study for collision risk between a manned and remotely-piloted unmanned aircraft under an assumption of the AFM environment. The activation time for the CD&R system on the unmanned aircraft varies depending on speed of the aircraft and path-crossing angles between the aircraft (Table 13). All other parameters needed for the DET framework are explained in the previous sections. The case study assumes level flight.

Table 13 Activation times for CD&R system on unmanned aircraft

Angle btw flight paths	30˚	60˚	90˚	135˚	180˚
Activation time (min to conflict)	4.25	3.25	2.60	2.12	1.98

The proposed DET framework models collision risk from the perspective of one aircraft. But the collision avoidance maneuver can be conducted by either aircraft. Only one aircraft needs to execute an avoidance maneuver. If both aircraft are independent in terms of physical components supporting the CD&R systems like the case study, it is possible to independently apply the framework to each aircraft. Then, the overall collision probability is the product of the two collision probabilities from each aircraft (i.e., a collision occurs if both aircraft fail to detect and avoid the other). The evaluation steps using the cDET-PMS was described in Chapter 3.

Figure 26 shows the resulting collision probabilities as a function of the path-crossing angle. These are *conditional* collision probabilities, under the assumption that two aircraft are on a collision course in the first place. As might be expected, the collision probability increases for larger path-crossing angles, since the closing speed increases, thus decreasing the time available to avoid a collision (180° represents a head-on scenario). But the collision risk is not completely monotonic. The collision risk decreases slightly at first and then increases. This is because there is a competing effect where the conflict detection algorithm in Paielli [1997] is more accurate for path-crossing angles between 45° and 90° (at least for the parameters used in this example), so the collision risk improves even though the time to avoid a collision decreases.

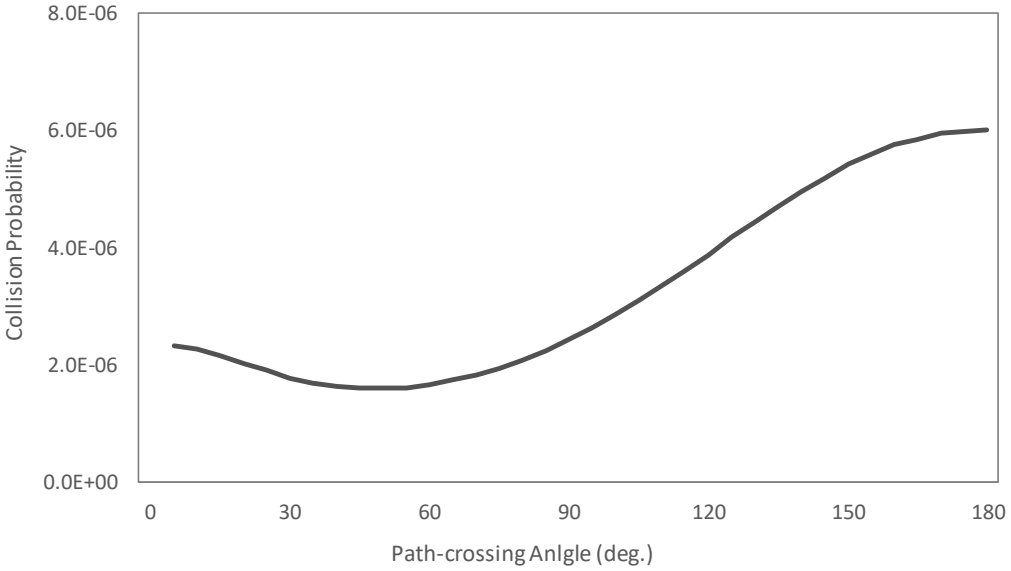


Figure 26 Collision probabilities of case study

Figure 27 shows contribution of failure modes on manned aircraft to collision risk of the case study. 'Algorithm/Pilot failures' indicates contribution of cases where all CD&R systems are available, but the algorithm fails to detect the conflict or the pilot does not respond in time. 'Component-based failures' shows the contribution of cases where all CD&R systems are unavailable due to component failures. Component-based failures are a major cause of collision risk; however, the relative contribution decreases for larger path-crossing angles. This is because the detection algorithm is less successful for larger path-crossing angles, thus the contribution of algorithm/pilot failures increases. For the unmanned aircraft, the algorithm/pilot failure is always the most contributing mode of failure (not shown in the figure).

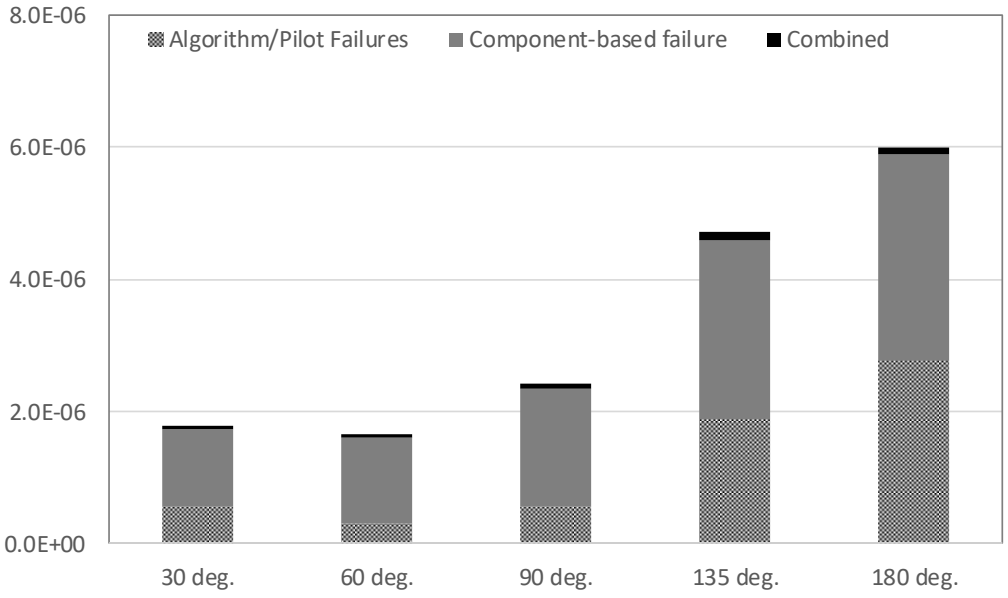


Figure 27 Collision probabilities of case study by failure modes

Figure 28 shows a sensitivity analysis of the failure probabilities of the components supporting the CD&R systems. Note that the first two elements are measured with the scale on the top axis, while the other elements are measured with the scale on the bottom axis. The value associated with each component is the relative change (improvement) in collision risk given a 10% reduction in the failure probability of the given component. For example, the transponder of the unmanned aircraft has a sensitivity of 0.044. This means that if the failure rate of the transponder is reduced by 10%, the collision risk would improve by 4.4%. The transponder on the unmanned aircraft is the most significant component followed by the TCAS processor on the manned aircraft. This is because all CD&R systems on the manned aircraft rely on the transponder to locate the unmanned aircraft.

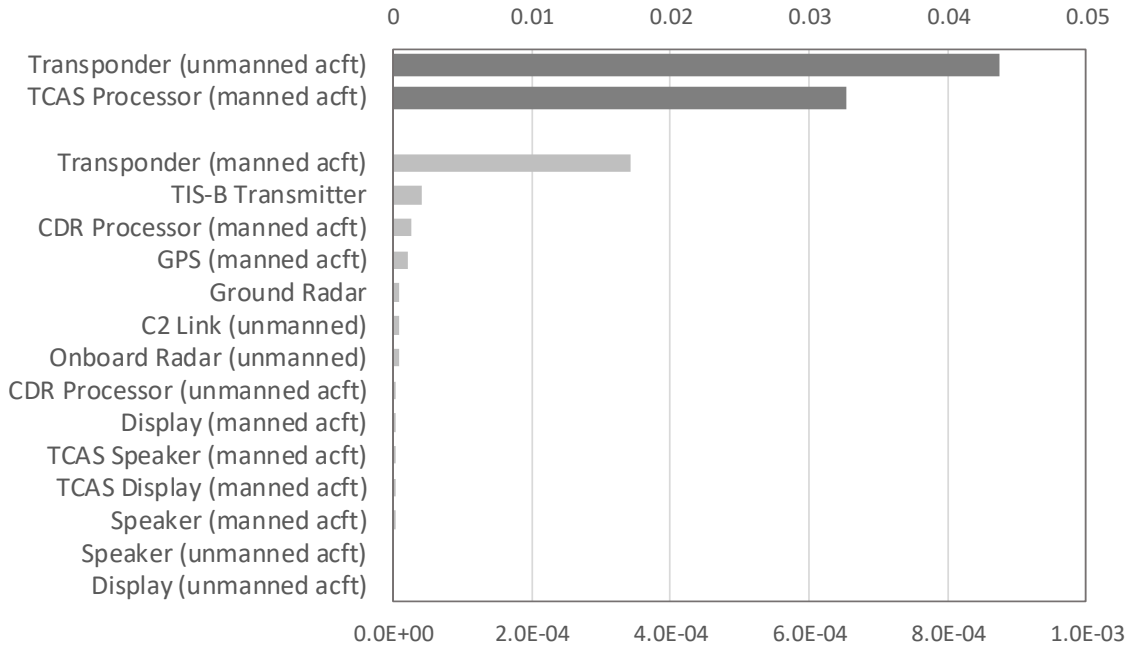


Figure 28 Sensitivity analysis result for component failure rate

Figure 29 presents a sensitivity analysis of the onboard radar detection range. Obviously, a longer detection range provides a better (i.e., reduced) collision risk. The values of sensitivity are the relative decrease in collision risk given a 10% increase of the onboard radar detection range on the unmanned aircraft. A sensitivity value of 0.09, for example, means that the collision risk is decreased by 9% in response to a 10% increase of the detection range. The improvement in collision risk varies with the path-crossing angle. The improvement gets larger as the path-crossing angle increases to 90°, then it becomes less with larger path-crossing angles. The figure also shows sensitivities with a 10% *decrease* of the radar detection range.

It is interesting to observe that the impact of an increased detection range for a 30° path-crossing angle is smaller than that for a 90° path-crossing angle. Intuitively,

with a slower closure rate (i.e., at smaller path-crossing angles), an increased range provides more time to avoid a conflict. Conversely, in a head-on case, increasing the detection range provides only a little more time. However, the risk reduction also depends on the conflict detection rate itself, which varies depending on the path-crossing angle. As an example, suppose that 10 seconds and 8 seconds of additional time are available to avoid a conflict for the 30° and 90° cases, respectively. Conflict detection probabilities per second are assumed about 0.01 and 0.02 for the two cases, respectively. Then, the total relative reduction in collision risk for the 30° case is about 9.6% ($\approx 1 - (1 - 0.01)^{10}$), while the relative reduction for the 90° case is about 14.9% (($\approx 1 - (1 - 0.02)^{8}$)). Even though fewer seconds are added in the 90° case, those seconds make more of a difference. (Note that the example is made for illustrative purposes.)

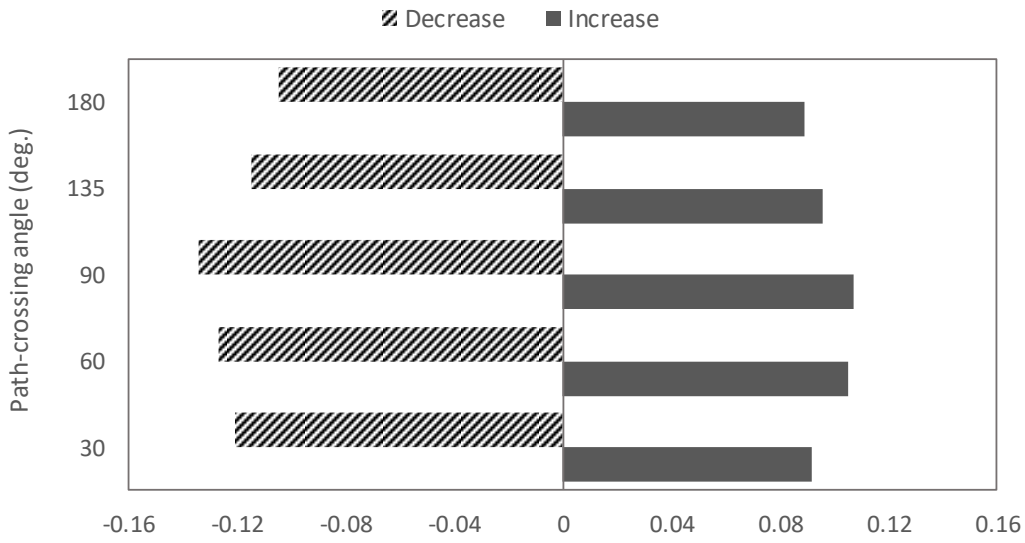


Figure 29 Sensitivity analysis result for onboard radar detection range

Next sensitivity analysis is conducted on the performance of the CD&R algorithms, specifically the trajectory prediction errors assumed in the algorithms (Figure 30). In this analysis, trajectory prediction errors for the unmanned aircraft are adjusted, while the uncertainty for the manned aircraft remains fixed. Similar to the previous sensitivity results, the value of the sensitivity is a relative change in collision risk given a change in trajectory prediction errors (e.g., errors on both along-track and cross-track dimensions change by 10%). A sensitivity value of 1, for example, means that the collision risk increases by 100% (twice as many collisions), while a value of -0.4 indicates a 40% reduction in collision risk. The impact of the trajectory prediction uncertainty is larger when two aircraft fly with a small path-crossing angle (e.g., less than 30°) or a large path-crossing angle (e.g., greater than 130°). That is, the conflict detection algorithm is more vulnerable to the uncertainty near the two extremes (i.e., 0° and 180°). Increasing the uncertainty on trajectory prediction affects the collision risk slightly more than decreasing the uncertainty.

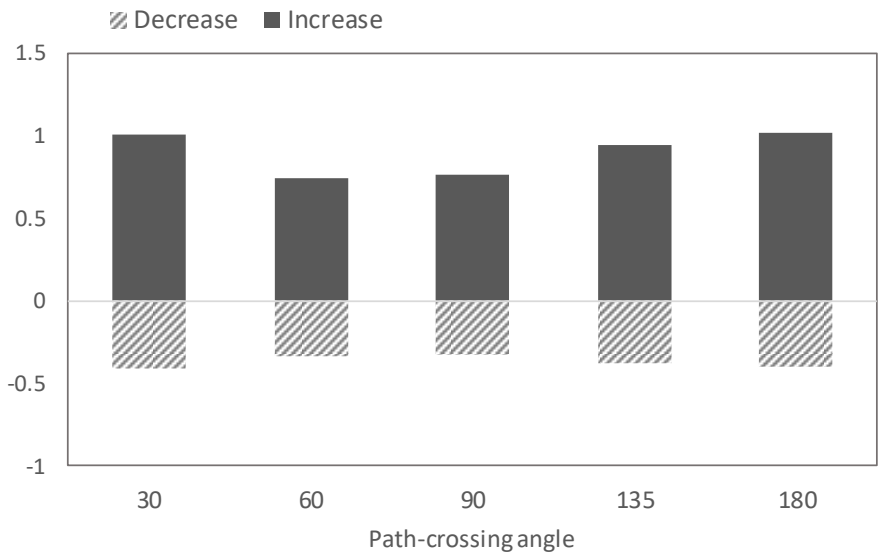


Figure 30 Sensitivity analysis result for trajectory prediction errors of CD&R algorithms

4.1.6 Discussion on Dependency between CD&R Systems

In the case study, the manned aircraft and the unmanned aircraft are independent in terms of physical components supporting the CD&R systems, thus an independent framework to each aircraft is applied. In reality, there can be dependencies between the two aircraft, since there may be common elements in the fault trees of CD&R systems on both aircraft. As an example, suppose that the UAS also has a TCAS-like system with a Mode S transponder (instead of a Mode A/C transponder) in addition to the onboard radar. The TCAS-like system on the unmanned aircraft performs the same function of the current TCAS system on the manned aircraft (i.e., direct interrogation of the transponder on the other aircraft). Similar to the current TCAS system, the assumed TCAS-like system for the unmanned aircraft requires working transponders on both aircraft, while the onboard radar is available as a backup surveillance (Figure 31). The other

components that support the CD&R system of the unmanned aircraft are the same as illustrated in Figure 24. Dependency between the two aircraft must be considered in this example, since the transponders on both aircraft appear in the fault trees of both aircraft (Figures 21-23).

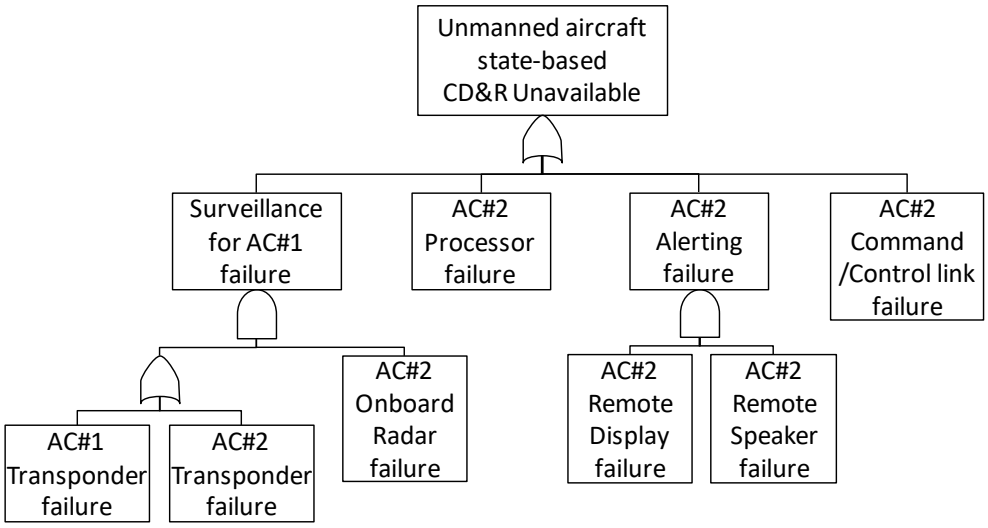


Figure 31 Supporting fault tree for tactical state-based CD&R system (unmanned TCAS-like)

In order to consider dependencies between CD&R systems on both aircraft, it is necessary to combine the two DET frameworks from each aircraft's perspective. Figure 32 illustrates the combination of two DET frameworks into one DET framework in terms of phase-time durations. As shown in the figure, the manned aircraft has three CD&R phases, each of which operates in $[T_1, T_2)$, $[T_2, T_3)$, $[T_3, 0]$ respectively, and the unmanned aircraft has one CD&R phase that starts at time T_4 prior to the predicted conflict. If T_4 is between T_1 and T_2 – i.e., the CD&R system of the unmanned aircraft is activated during

the first phase for the manned aircraft – then this first phase is divided into two phases for the joint DET framework, [T_1, T_4) and [T_4, T_2). The combined framework has four phases in total. In the first phase, only the strategic intent-based system of the manned aircraft is operating. In the remaining three phases, both aircraft have CD&R systems operating in some combination. In the example, T_4 is assumed to be between T_1 and T_2. But this is not always the case. The number of phases, the time horizons of the phases, and the CD&R systems that are operating in each phase depend on the activation times, the detection range of sensors, aircraft speeds, and collision geometries. Once the two DET frameworks are integrated, the evaluation steps of the combined DET framework are the same as explained in Chapter 3.

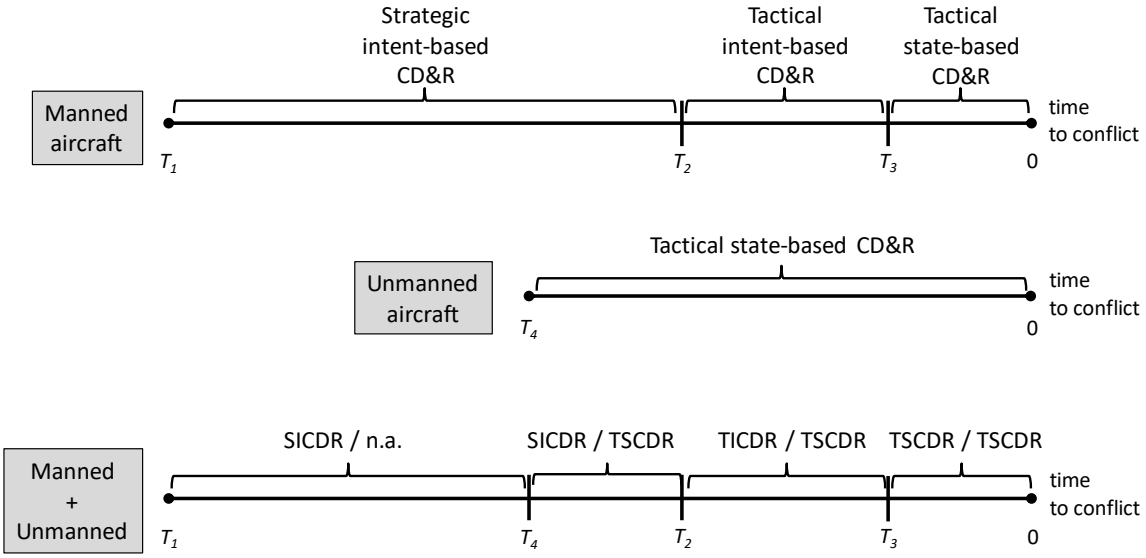


Figure 32 Combining two DET frameworks

An example analysis of dependent CD&R systems is conducted for an unmanned aircraft equipped with an onboard radar and a TCAS-like system with a Mode S transponder (as shown in Figure 31). The TCAS-like system on the unmanned aircraft is assumed to perform conflict detection with various levels of accuracy. Successful conflict detection probabilities of the system are varied ranging from 30% to 80% that of the manned aircraft. The performance level of 30% is the same level considered in the original case study. The detection range is assumed to be 35 km, as before.

Figure 33 illustrates the relative change in collision risk for the different combinations of sensors and conflict detection performance levels, compared to the original case study. For example, for the case of 'TCAS-like + Onboard rada (50%)' at a 180° path-crossing angle, the value of 0.4 means that the collision risk is improved by 40% compared to the case study. Obviously, better conflict detection performance yields reduced collision risk. In terms of path-crossing angle, the collision risk improves with smaller path-crossing angles since more time is available to avoid a collision. With the same algorithm performance level (30% scenario), the TCAS-like system can change the collision risk by 15%. The effect is small because the components additionally required for the TCAS-like system on the unmanned aircraft (i.e., transponders) are common elements that already support the CD&R systems on the manned aircraft. Thus, the improvement is not as high as might be expected, even though the unmanned aircraft has two different sources for surveillance information in parallel.

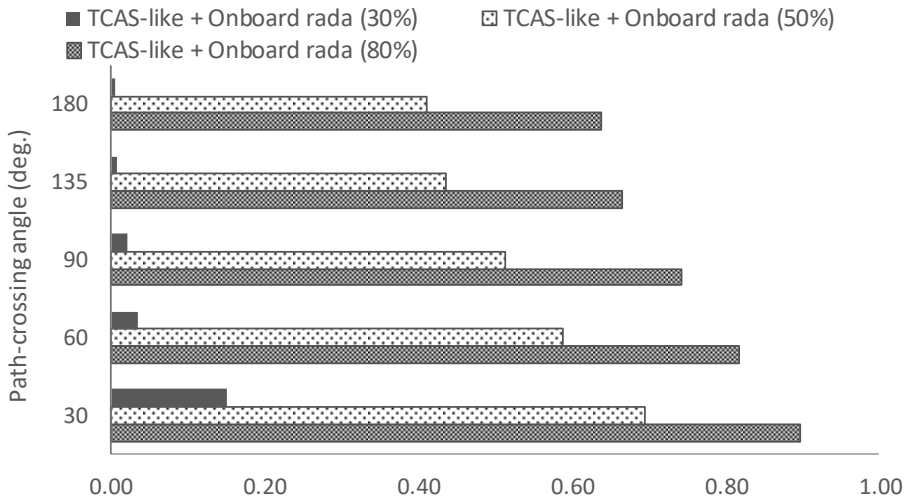


Figure 33 Relative collision risk of various CD&R systems on unmanned aircraft to case study

4.2 Case Study-2: Advanced Airspace Concept (AAC)

4.2.1 Concept of AAC Operations

The Advanced Airspace Concept (AAC) was firstly proposed by Erzberger [2001]. The AAC has evolved and has been evaluated through several researches in terms of system architecture, algorithm, and safety [Erzberger 2004; Andrews 2005; Blum 2010; Erzberger 2012]. The AAC is a ground-based centralized concept, in which an automated system detects potential conflicts between aircraft and provides resolutions to the aircraft via air-ground data link. The automated separation assurance system replaces many of the roles of air traffic controllers in the current NAS, while controllers remain in the system as a backup and monitoring purpose.

According to Erzberger [2001, 2004, 2012], the ground-based separation assurance system consists of two independent sub-systems operating conflict detection and resolution autonomously, the Autoresolver (AR) and the Tactical Separation Assured

Flight Environment (TSAFE). The AR is designed to detect and resolve a conflict predicted to occur approximately 20 to 2 minutes prior to a potential conflict, while the TSAFE provides tactical conflict detection and resolution within 3 minutes to an expected conflict. In addition, similar to AFM operations, the TCAS is assumed to be the last CD&R system that provides separation assurance within 1 minute to a predicted conflict. The central system of AAC on the ground collects surveillance information (position and speed) of all aircraft in a particular region of airspace. The system automatically detects conflicts, generate conflict free trajectories, and upload the trajectories directly into onboard systems of properly equipped aircraft. The voice communication link is still used for air traffic controllers to provide separation to unequipped aircraft or to equipped aircraft as a backup.

Note: Time horizons of each CD&R system for the AAC are overlapped slightly, so an assumption that the latest CD&R system has a priority to the previous one is made, e.g., the TSAFE would take over the AR at 3 min prior to a conflict if the AR fails to resolve the conflict until then. In addition, the AR is assumed to activate at 8 min prior to a conflict since 8 min provides an acceptable trade-off between the advantage of early alerting and the disadvantage of false alarms [Erzberger 2012].

4.2.2 CD&R Systems

Unmanned aircraft systems (UAS). UAS assume to be introduced into the future NAS. As discussed in Section 4.1, two different types of conflict avoidance capabilities, Mode A/C transponder and onboard radar and TCAS-like system and onboard radar, are considered for the CD&R system of the unmanned aircraft. All the assumed concept of

operations and failure relationships between supporting components and the system are hold as described in Section 4.1.2. One additional assumption made is that UAS is not controlled by AAC nor human air traffic controller, thus it flies as like the one under AFM operation. In other words, UAS is assumed to maintain separation to other aircraft by itself.

Fault trees for the AAC. Fault trees for the two CD&R systems of AAC concept, AR and TSAFE, are constructed to illustrate logical relationships between supporting components and the system function. These fault trees are based on a series of literature ([Erzberger 2004; Andrews 2005; Blum 2010]), which studied the AAC in terms of concept of operation, design/architecture, and safety, as well as the assumed CD&R system on the unmanned aircraft. (Note that the fault trees may be different for a pair of aircraft with different equipage.)

Figure 34 shows the failure logic of the AR. The AR can fail either due to the failure of components supporting the system (on the left side of the figure) or due to a surveillance failure (on the right side of the figure). The AR is supported by a separate processor that receives positional data for all aircraft in a given region of airspace, predicts trajectory of aircraft, detects a predicted loss of separation (LOS), and generates resolution of the LOS. Then, the AR sends the resolution directly to a display on aircraft via a data link. Voice communication link between air traffic controller and pilot is available as a backup.

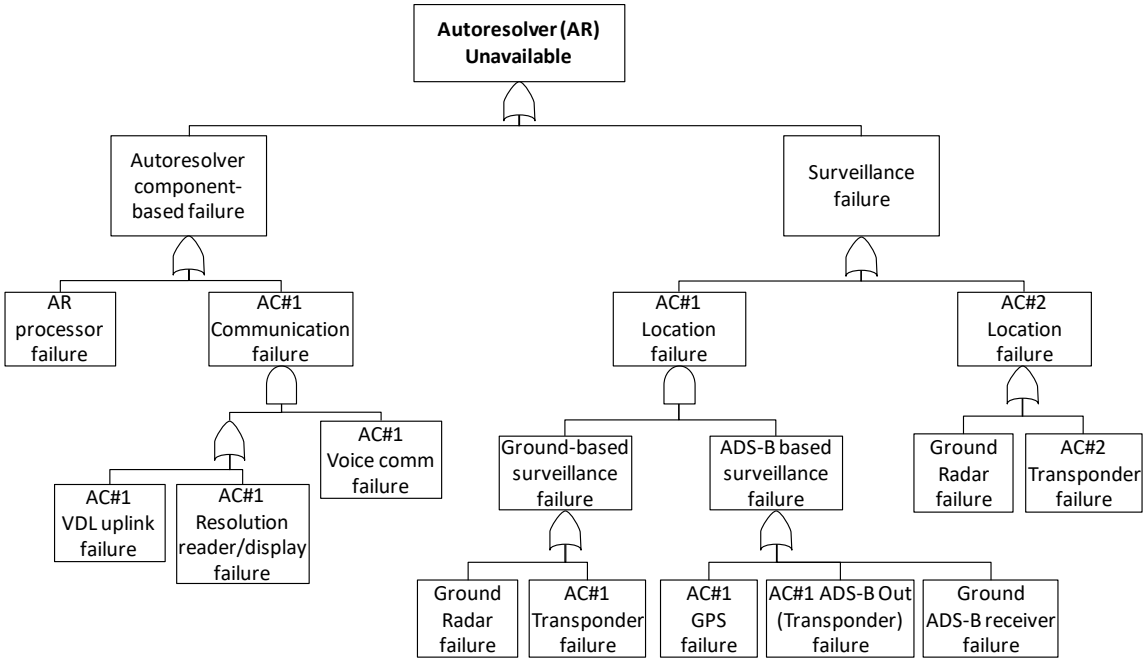


Figure 34 Supporting fault tree for Autoresolver (AR)

A surveillance failure occurs when the AR on the ground cannot locate either aircraft flying on a potential collision course. Location information of manned aircraft can be obtained primarily by Automatic Dependent Surveillance-Broadcast (ADS-B), which is equipped on most manned aircraft, as well as by a radar on the ground. GPS on aircraft is assumed to collect its own state information, i.e., position, velocity, heading, and altitude, and it passes this information to ADS-B Out, which is implemented on Mode S transponder. ADS-B Out creates ADS-B messages and broadcasts, then a receiver on the ground collects the ADS-B messages. As a backup surveillance system, a ground radar is used to locate aircraft by interrogating the transponder onboard as it is in the current NAS. The ground radar is the only source of locating unmanned aircraft since UAS assumes not to be equipped with ADS-B.

The TSAFE, which is also located on the ground, is to detect and resolve potential conflicts in tactical time horizon, i.e., within 3 min prior to a predicted conflict. Relationship between supporting components failure and failure of the TSAFE is shown in Figure 35. Similar to the AR, failures of components supporting the TSAFE or failures of tracking location of either aircraft on a collision course cause the TSAFE to fail. The TSAFE relies on the same sources for location information as the AR does, which are ADS-B and/or ground radar. The TSAFE has its own processor, which is independent to the one used for the AR, that runs conflict detection and resolution algorithm. A main difference from the AR is that a resolution found by TSAFE is communicated to the aircraft by the Mode S data link using the transponder onboard.

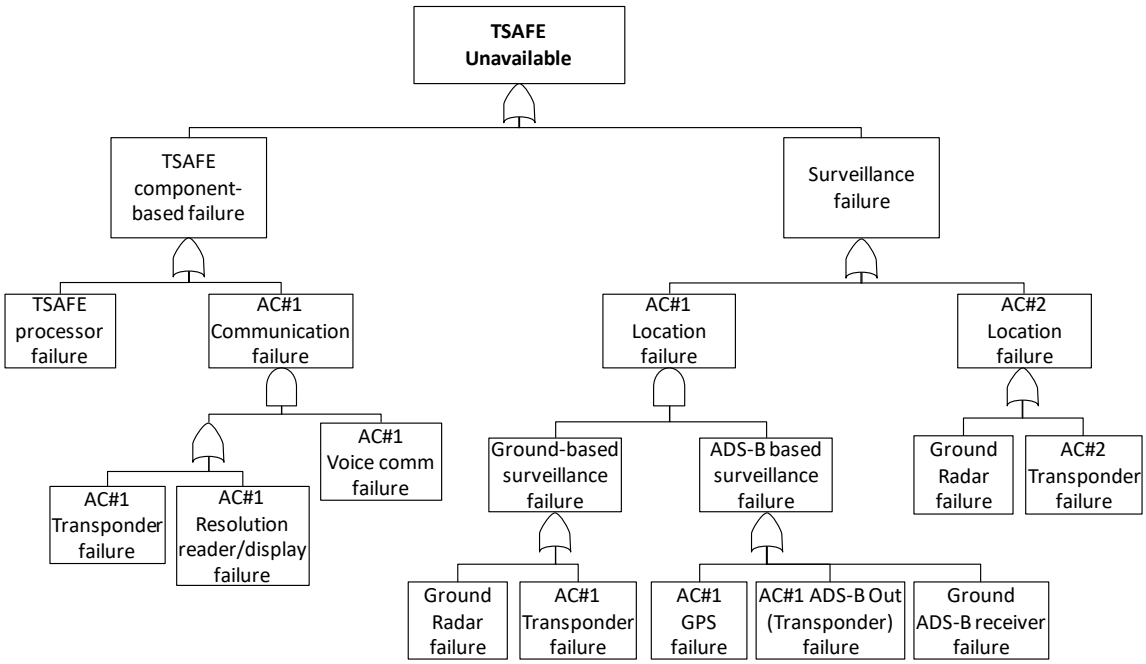


Figure 35 Supporting fault tree for TSAFE (manned aircraft)

The last CD&R system that assumes to activate at the last minute prior to a potential collision is Traffic Alert and Collision Avoidance System (TCAS). Detailed explanation for TCAS is available in Section 4.1, where operational concept, requirements, and failure logic are discussed.

4.2.3 Result & Comparison with Case Study-1

This section presents numerical results of collision risk between a manned and unmanned aircraft, which are equipped with a different level of collision avoidance capabilities, in the AAC operation environment. Three CD&R systems, two (AR and TSAFE) on the ground and the other (TCAS) onboard, are available for the manned aircraft, while a tactical state-based CD&R system utilizing an onboard radar performs conflict detection and resolution functions on the unmanned aircraft. The activation times for the ground-based CD&R systems and TCAS are pre-set at 8 min, 3 min and 1 min prior to a potential conflict respectively, while the activation time for the CD&R system on unmanned aircraft varies (4.25 min ~ 1.98 min to a conflict as reported in Table 13) depending on the speed of the aircraft and path-crossing angles between the aircraft.

Figure 36 shows the resulting collision probabilities of the two case studies as a function of the path-crossing angle. As stated in the result section of Case Study-1, these probabilities are *conditional* collision probabilities given that two aircraft are on a collision course. The same pattern of the collision probabilities for Case Study-1 (dashed line) is observed in the result of Case Study-2 (solid line) throughout the path-crossing angles. This is because of the assumption that the CD&R systems under both AFM and AAC operations use the same algorithm to detect a potential conflict. Overall, the

collision probabilities increase as the path-crossing angle grows since the closing speed of aircraft increases, thus time available to avoid a collision decrease. In addition, the resulting collision probabilities of AAC operation are higher than the probabilities of AFM operation through all the path-crossing angles (i.e., AFM operation is better than AAC in terms of system architecture given assumption of the same parameter values and CD&R algorithm performance). The CD&R systems of the AAC more likely fail than the systems of AFM operation because the AR, TSAFE and TCAS heavily rely on the transponders on both aircraft whereas the CD&R systems under AFM operation depend on the transponder on the unmanned aircraft only.

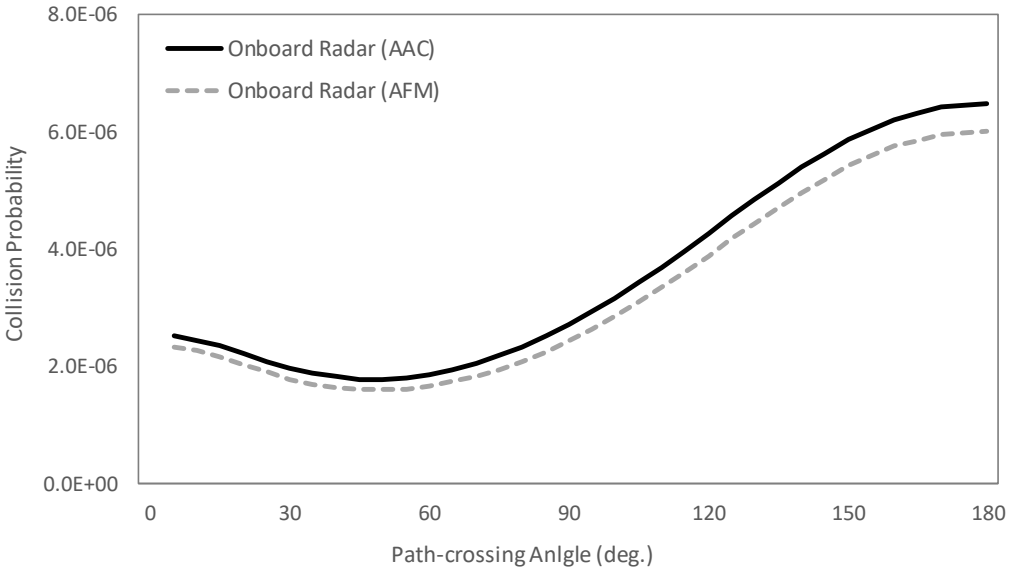


Figure 36 Collision probabilities of Case Study-1 & 2

A result of sensitivity analysis on components supporting the CD&R system under AAC operation is shown in Figure 37, where the top two bars correspond to the top

axis, while the others are read on the bottom axis. The sensitivity is measured as relative change (improvement) in the resulting collision probability when the failure rate of each component is reduced by 10%. The transponders on both manned and unmanned aircraft are the most significant components for the collision risk in AAC environment. The sensitivity value of 0.039 for the transponder on manned aircraft (or on unmanned aircraft) means that the collision risk can be reduced by 3.9% if the reliability of the transponder is increased by 10%. This is because all the CD&R systems operating under the AAC concept including TCAS rely on the transponders on both aircraft to locate the aircraft on a potential collision course.

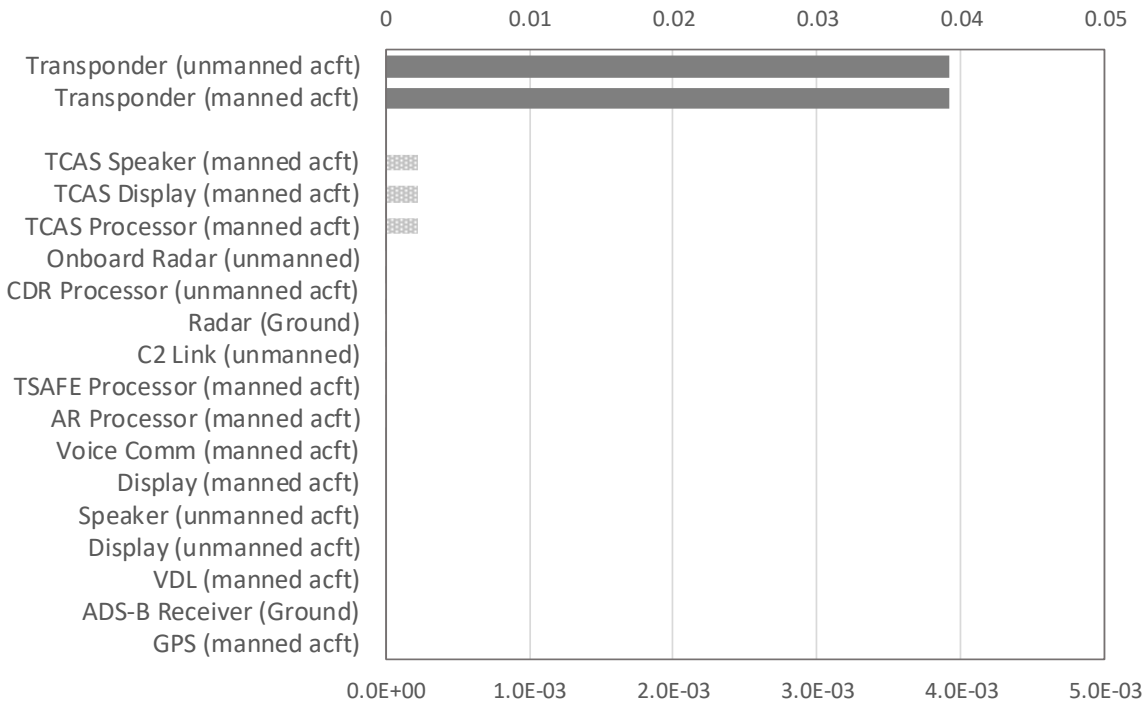


Figure 37 Sensitivity analysis of components (AAC)

Sensitivity analyses on the other model parameters (e.g., sensor detection range, trajectory prediction errors used for conflict detection probabilities) are also conducted, however, the results are not presented since the results are very similar to the ones presented for AFM operation in Section 4.1.5.

4.3 Case Study-3: Collision Risk Between Manned Aircraft

4.3.1 Concepts of NAS Operations

Current NAS (Air Traffic Controller, ATC). Currently unmanned aircraft is not allowed to fly with manned aircraft at the same time in the NAS, where air traffic services, e.g., air traffic control, flight information, and alerting services, are provided primarily by human air traffic controllers. In order to support the controllers to ensure the safety and efficient operations of air traffic in the NAS, a number of new procedures and technologies have been developed and implemented for decades, especially through the Next Generation Air Transportation System (NextGen) project. Even though the NAS has been improved a lot compared to that several decades ago, the principal and/or concept of the NAS operations are still not changed much, where human pilot operate aircraft, and human air traffic controllers mostly provide air traffic service for pilot to safely complete flights.

One of the most important services that the controllers provide is to maintain a safe separation between aircraft in the air, to which this research is directly related. Figure 38 shows failure logic of providing the separation assurance service in terms of physical components. In order for air traffic controllers to maintain a safe separation between aircraft, surveillance information of both aircraft that may have potential risk of

mid-air collision is firstly required (the right branch of Figure 38). Location of each aircraft can be available either ground-based systems (i.e., surveillance radar on ground and transponder on aircraft) or ADS-B, of which aircraft must equip ADS-B Out in most NAS by 2020. As explained in Section 4.1, the ADS-B Out function is currently implemented on the Mode S transponder of most commercial aircraft. The trajectory generator and voice communication between controllers and pilots should be both available to detect a conflict and inform pilots to execute a resolution maneuver to avoid the conflict.

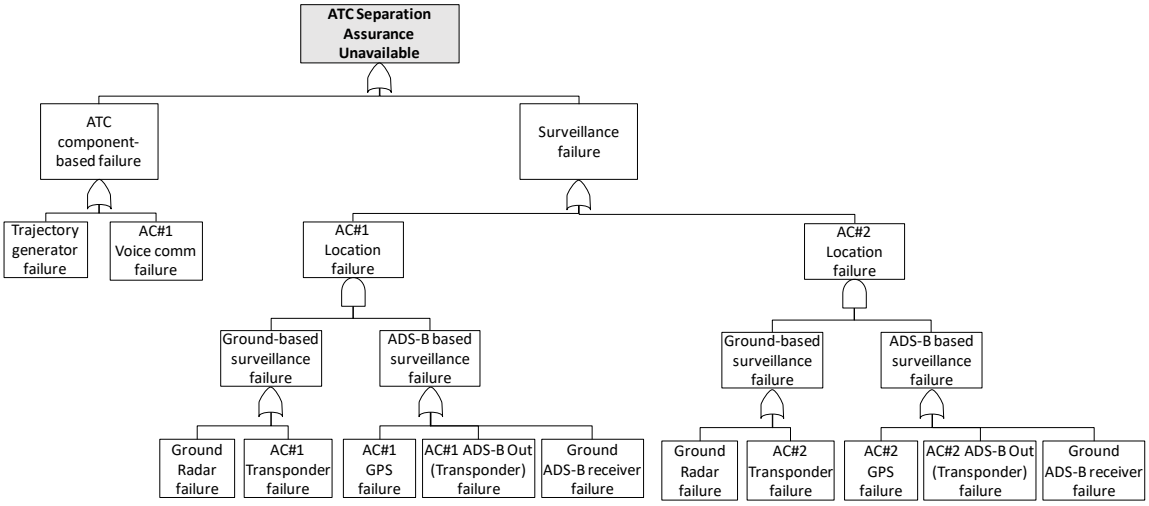


Figure 38 Supporting fault tree for air traffic control (ATC) separation assurance

Additional assumptions made to model collision risk between a pair of manned aircraft in the current NAS with the proposed DET framework are as follows:

- Similar to other concept of the NAS operation, ATC separation assurance service is provided in a time horizon between 8 min and 1 min prior to a potential conflict.
- Traffic Alert and Collision Avoidance System (TCAS) performs a separation assurance function within 1 min to a predicted conflict.
- Probabilities that air traffic controllers detect a conflict and provide a resolution are a function of time to a predicted conflict. Probability values are assumed as 30% of that for the hypothetical CD&R algorithm described in Section 4.1.4 to detect a conflict.

Future NAS. Two concepts of the future NAS operations, Autonomous Flight Management (AFM) and Advanced Airspace Concept (AAC), were discussed in detail in the previous Case Studies, where model collision risk between manned and unmanned aircraft. The same concepts of operations in the future NAS are used to model collision risk between a pair of manned aircraft. Figure 39 illustrates supporting fault trees for strategic intent-based CD&R system (top) and tactical intent-based CD&R system (bottom) under AFM operation, where relationship between component failures and system failure is modeled. A key difference between manned-unmanned pair (Figure 21 ~ 22) and manned-manned pair (Figure 39) is that there is another way for a manned aircraft to locate the other aircraft, which is Air-to-Air Surveillance via ADS-B. All other parameters such as operation time horizon of each CD&R system and time-varying detection probabilities are assumed the same with the Case Study-1 including TCAS equipped on both manned aircraft.

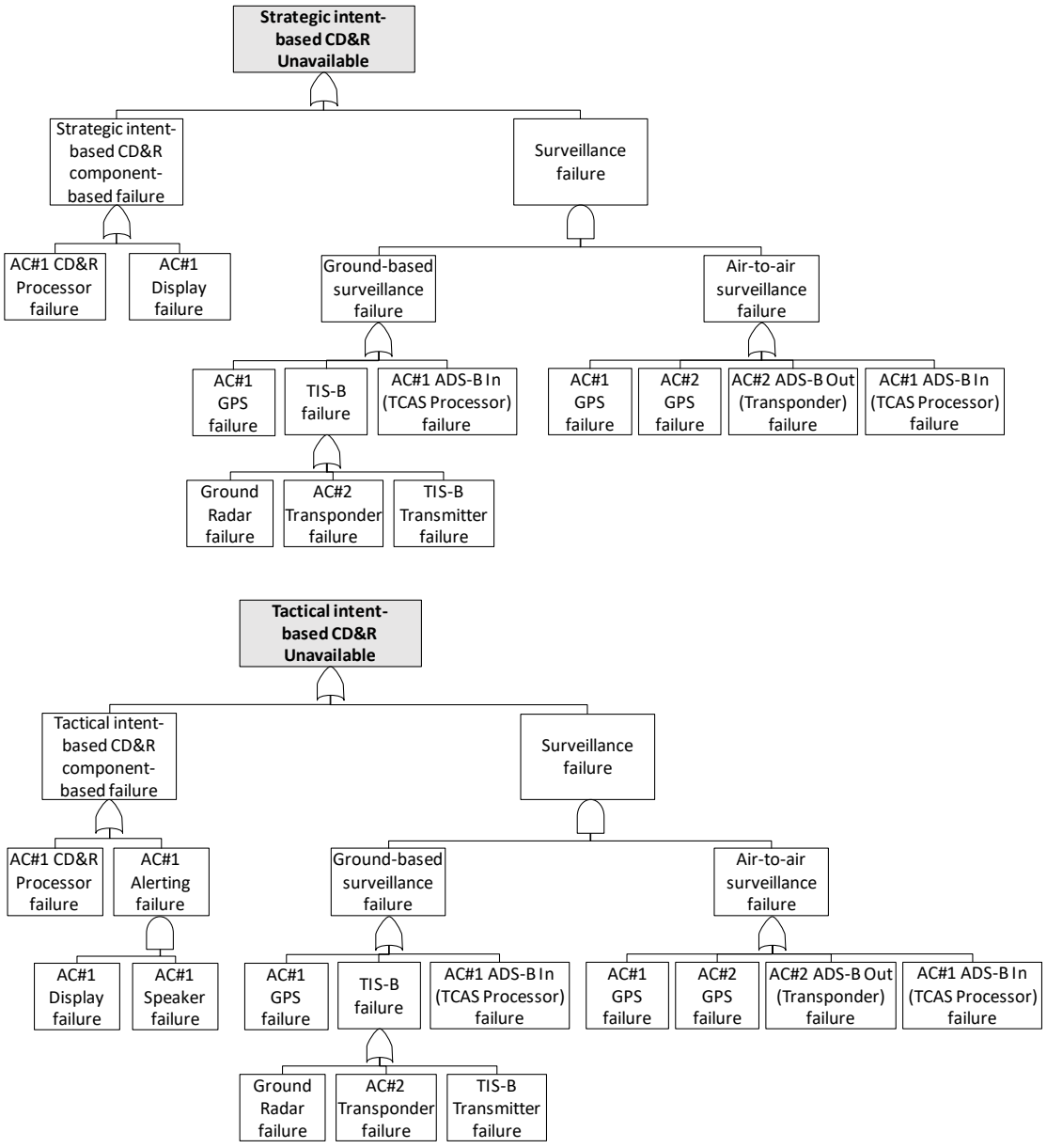


Figure 39 Supporting fault trees for CD&R systems of manned aircraft (manned-manned pair, AFM)

Similarly, supporting fault trees of the AAC CD&R systems (i.e., Autoresolver (AR) and TSAFE) for a pair of manned aircraft are modeled in Figure 40. Compared to

Figure 34 and 35, where fault trees of AR and TSAFE for manned-unmanned aircraft pair are illustrated, location information of the second (manned) aircraft is available to the CD&R systems through ADS-B Out on manned aircraft. (Note that the fault trees shown in Figure 38 ~ 40 are from one aircraft (AC#1) perspective, thus the fault trees with the same structures from the other aircraft (AC#2) perspective are also needed to completely model the case studies.)

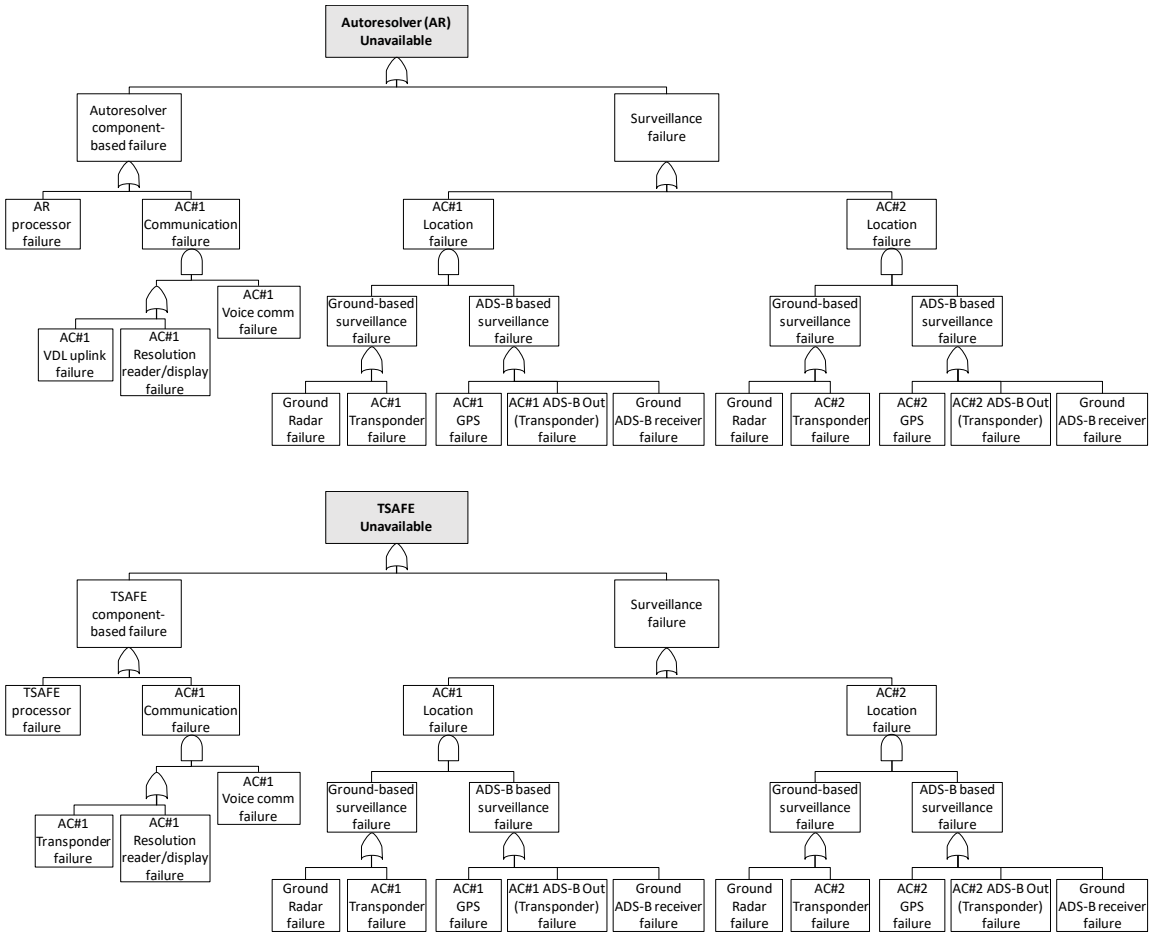


Figure 40 Supporting fault trees for CD&R systems of manned aircraft (manned-manned pair, AAC)

4.3.2 Result and Comparisons

This section provides numerical results and sensitivity analyses of the case studies for collision risk between manned and manned aircraft under an assumption of the current NAS (i.e., ATC), AFM, or AAC environment. Figure 41 shows the resulting collision probabilities between two manned aircraft in each NAS operation environment as a function of the path-crossing angle. The collision risk between manned and manned aircraft under ATC and AAC operations are similar with a high probability (in the order of 10^{-5}) for all path-crossing angles. The reason of this result is that the collision risk of ATC and AAC operations heavily rely on components failures, specifically failures of transponders on either aircraft. Based on failure logic of the CD&R systems (Figures 38 and 40) including TCAS (Figure 23) for both ATC and AAC operations, failure of the transponder on either aircraft leads to surveillance failure of all CD&R systems on the ground and the TCAS on both aircraft, thus no opportunity for either aircraft to avoid a collision. (Note again that the collision risk/probability in this dissertation is a conditional collision risk/probability given that two aircraft are on a collision course.)

In order for the result collision probability of the AAC operation in this research to be directly compared with the current mid-air-collision accident rate, it must be multiplied with the rate that two aircraft are on a potential collision course. Belle [2012] provided the rate that two aircraft are on a converging path as a function of the number of aircraft in an airspace for both structured air route (i.e., the current NAS) and great circle

route (GCR, i.e., free flights). The result rates range from in an order of 10^{-4} to 10^{-3} per flight hour for structured air route and from in an order of 10^{-5} to 10^{-4} per flight hour for GCR depending on the number of aircraft in a region of airspace. With the rates for structure air route (i.e., $10^{-4} \sim 10^{-3}$ per flight hour) and the probability that the ATC fails to resolve a conflict in this research (i.e., 10^{-5}), the total collision risk of the current NAS would be at least in an order of 10^{-8} per flight hour, which is similar to literature (e.g., Lin [2009] and ISAM).

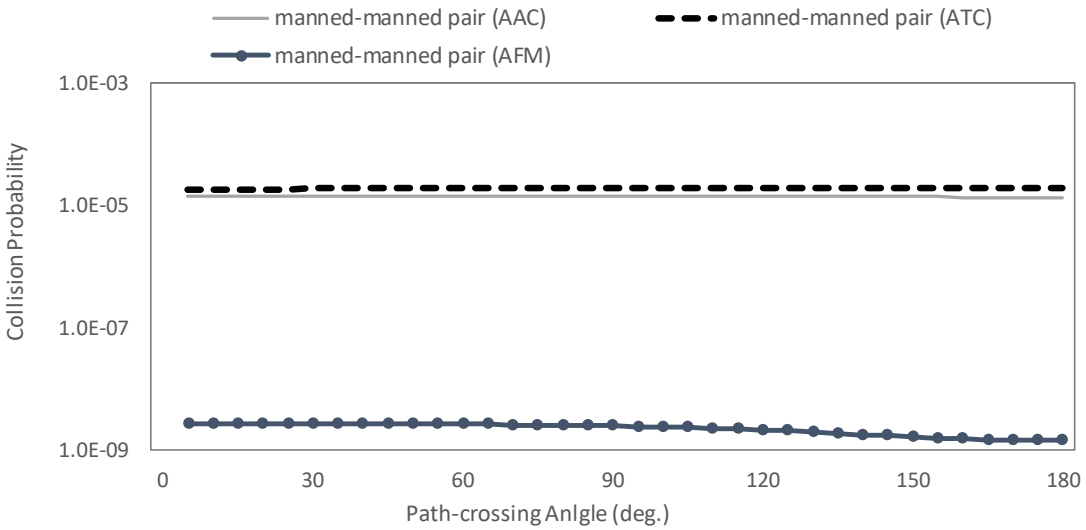


Figure 41 Collision probabilities for manned-manned aircraft pair (ATC vs. AFM vs. AAC)

The risk under AFM environment is mostly maintained at a probability in the order of 10^{-9} when path-crossing angle is less than 90°, then it decreases, but still at the same order of magnitude. The collision risk of AFM operations is much less than the risk under ATC and AAC since failure of a single component does not cause all CD&R systems on both aircraft down in AFM operation environment. The reason why the

collision risk of AFM operations decreases is a combination of two factors: 1) the most contributing scenarios to the collision risk are the case where two early CD&R systems are available but TCAS is not for one of the two aircraft, while all CD&R systems fail for the other aircraft, and 2) for manned-manned aircraft pair, the speed and trajectory prediction errors for both aircraft are assumed the same, and the conflict detection for that pair with a larger path-crossing angle is estimated as very successful, especially at early CD&R phase.

Sensitivity analyses are conducted on the components supporting the CD&R systems of the three concepts of the NAS operation, and the results are summarized in Figure 42. As same with the previous sensitivity analyses, the value associated with each component is the relative change (improvement) in collision risk given a 10% decrease in the failure probability of the component. The components shown in the Figure 42 are selected based on a criterion that the sensitivity result of a component is greater than 0.001. The transponders on both aircraft on a potential collision course are most significant for all three NAS operation environments. In addition to the transponders, the TCAS processors and the processors of CD&R systems also have recognizable impacts on the collision risk under AFM environment, while all other components have negligible impact on the collision risk of ATC and AAC operations.

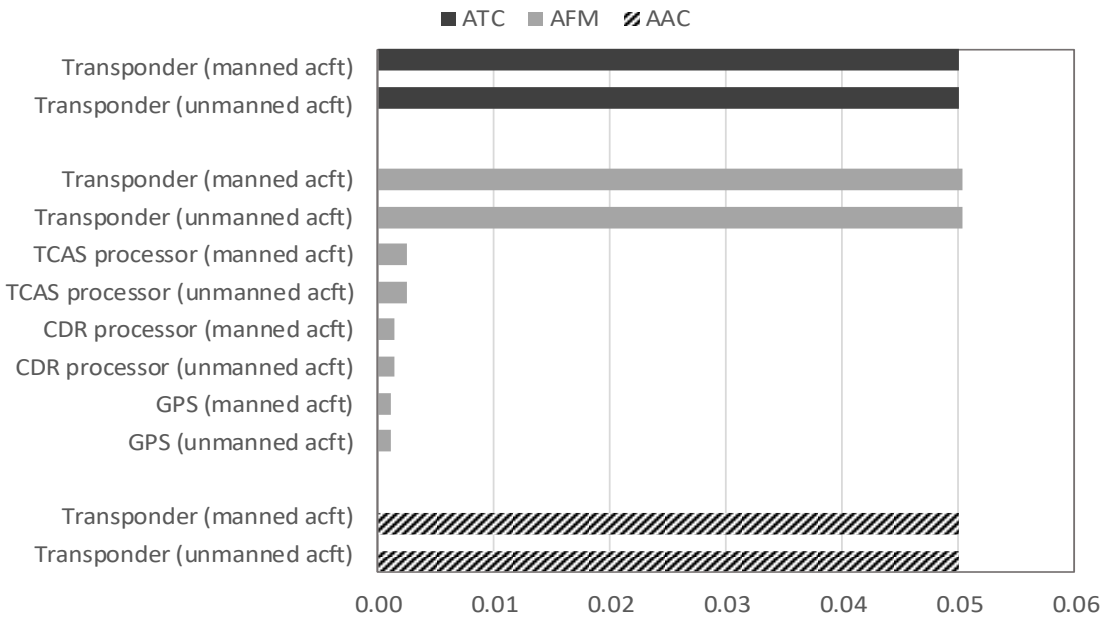


Figure 42 Critical components from each concept of operations based on sensitivity analysis

A safety-critical observation shown in Figure 43 is that the collision risk between manned and manned aircraft would be worse than the risk between manned and unmanned aircraft under AAC environment. This happens, as mentioned earlier, because the CD&R systems on manned aircraft, AR, TSAFE and TCAS, all depend on a single component (a transponder on either aircraft), while the assumed CD&R system on unmanned aircraft has an independent source (onboard radar) of sensing the other aircraft, thus does not completely rely on the transponder. This dependency, which is very significant to the collision risk, would expect to be reduced by having a redundant transponder onboard or equipping a separate ADS-B Out system, which is currently implemented on the transponder.

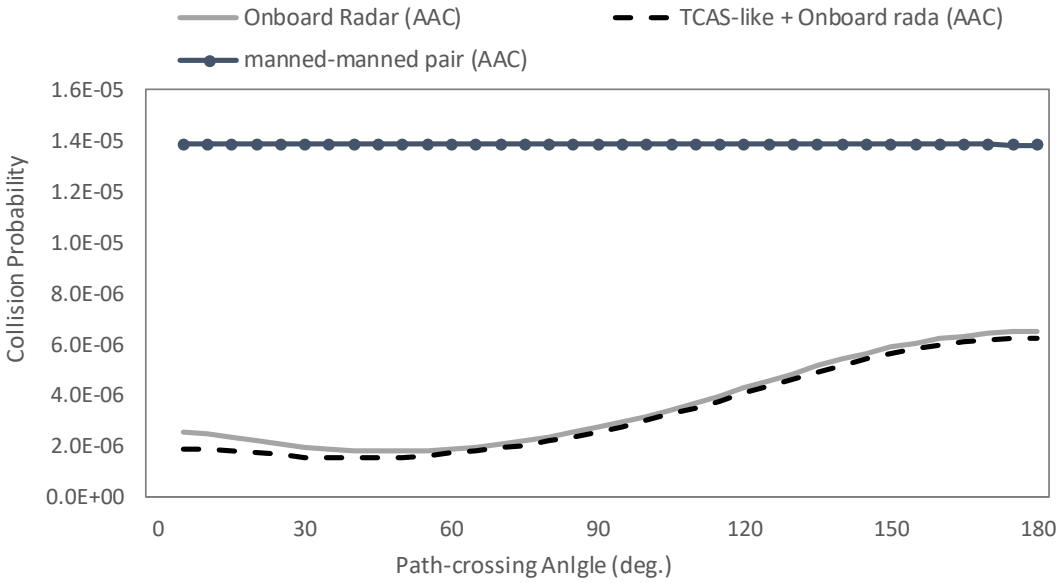


Figure 43 Comparison of collision probabilities between manned-manned and manned-unmanned pairs (AAC)

4.4 Summary

This chapter presented an application of a dynamic event tree framework to evaluate collision risk between aircraft equipped with different collision avoidance capabilities. Firstly, a case study in detail was developed for collision risk between a manned and a remotely-piloted unmanned aircraft, both flying under Autonomous Flight Management (AFM). For the manned aircraft, parameters of the conflict detection and resolution (CD&R) systems were studied. Fault trees were constructed to model failure relationships between physical components of each CD&R system. Time varying conflict-detection probabilities were estimated based on an algorithm from Paielli [1997]. For unmanned aircraft, various types of sensor technologies were surveyed in terms of type, information acquired, and detection range. A combination of a Mode A/C transponder and an onboard radar with an assumed CD&R concept of operation for

unmanned aircraft was firstly considered. A way to apply the DET framework considering dependency between aircraft on a collision course was also discussed with another CD&R system (i.e., TCAS-like system with a Mode S transponder and an onboard radar).

With Advanced Airspace Concept (AAC) for the future NAS operations, collision risk between manned and unmanned aircraft with two different CD&R systems are modeled. Lastly, case studies for collision risk between manned and manned aircraft under various NAS environments, AFM, AAC, and the current NAS (ATC), are developed. Table 14 summarizes the case studies discussed in this chapter.

Table 14 Summary of case studies

Case No.	Conflict Case	NAS Operation	Collision Avoidance Equipage (Unmanned)
1	Manned aircraft – Unmanned aircraft	AFM	Onboard radar, transponder
2	Manned aircraft – Unmanned aircraft	AFM	Onboard radar, TCAS-like system
3	Manned aircraft – Unmanned aircraft	AAC	Onboard radar, transponder
4	Manned aircraft – Unmanned aircraft	AAC	Onboard radar, TCAS-like system
5	Manned aircraft – Manned aircraft	ATC	-
6	Manned aircraft – Manned aircraft	AFM	-
7	Manned aircraft – Manned aircraft	AAC	-

Under AFM and AAC operation environment, collision risk between a manned and an unmanned aircraft increases with greater path-crossing angles, since the closing

speed between aircraft increases reducing available time to avoid a collision. Sensitivity analysis indicated that the transponder on the unmanned aircraft is the most significant component in given AFM operation, while the transponders on both manned and unmanned aircraft are most important in AAC environment. The maximum detection range of the onboard radar also affects collision risk, especially when two aircraft are approaching with an acute path-crossing angle. The effect of the new CD&R system (i.e., TCAS-like system) for the unmanned aircraft to collision risk is very small unless the detection performance of the new system is better than the previous system, because the components additionally required for the new system on the unmanned aircraft (i.e., transponders) are the common elements in the CD&R systems on manned aircraft.

The collision risk between manned and manned aircraft under ATC and AAC operations are similar with a high probability for all path-crossing angles due to dependency on transponders of either aircraft. The collision risk of AFM operations, however, is much less than the risk of ATC and AAC environments since failure of a single component does not cause all CD&R systems on both aircraft fail. The transponders on both manned aircraft flying on a collision course are the most significant components to the collision risk for all three NAS operation environments.

Throughout the case studies and sensitivity analyses, it is observed that the CD&R systems/concepts considered in this dissertation depend significantly on the aircraft transponders. Based on that, one of the means that can improve the CD&R systems thus decrease the risk is to equip an additional redundant transponder as a backup on aircraft. The graph on left of Figure 44 shows resulting collision probabilities for the

case #1 (solid line) and the case where a redundant transponder assumes to be equipped on both aircraft of the case #1 (dashed line). The additional transponder can decrease the collision risk by ranging from 30% to 46% depending on path-crossing angles. Similarly, the graph on right of the figure shows collision probabilities for the case #7 assuming that both manned aircraft have a separate ADS-B Out component instead of the current implementation on Mode S transponder. Having a separate ADS-B Out component reduces dramatically surveillance failure of aircraft, thus decreases the collision risk by several orders of magnitude. (Note that the primary radar system is assumed not in use.)

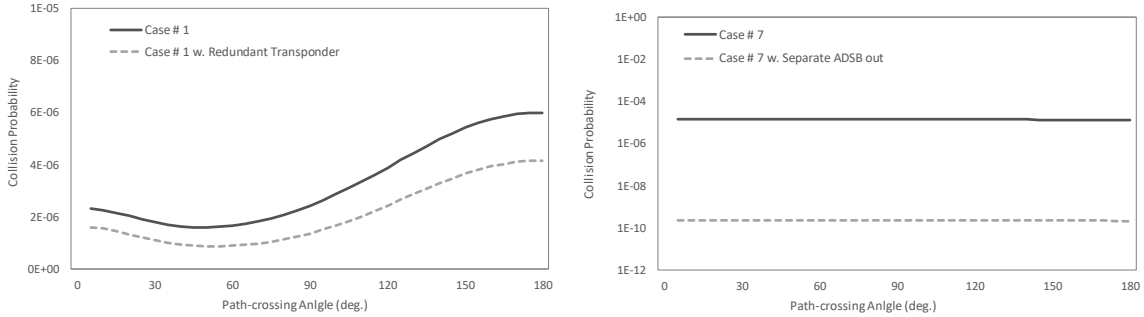


Figure 44 Collision probabilities with redundant transponder (Case #1, left) and separate ADS-B Out (Case #7, right)

CHAPTER 5: CONCLUSIONS

5.1 Summary and Results

This dissertation proposed a general form of a dynamic event tree (DET) to model the mid-air collision risk between aircraft. The DET framework consists of three levels, a high-level dynamic event tree, a generic sub-tree, and fault trees. A methodology for evaluating the DET framework was proposed (cDET-PMS), combining a conditional-based approach for evaluating event trees with a phased-mission-system approach from the reliability literature. Several variants of the approach (cDET and PMS-BDD) were considered and applied to evaluate a test DET problem. Each method has different assumptions on the timing of component-based system failures and the modeling of event sequences in the sub-trees. The PMS-BDD approach has significant limitations in modeling possible sequences of events, which causes under-estimation of the true risk. The cDET and cDET-PMS method perform differently depending on the numerical parameters of the test problem. They estimate collision risk very well in case that algorithmic failures of the first two CD&R systems occur very likely, while they may over-estimate the risk by a factor of more than 10 in other cases.

The proposed DET framework and evaluation methods have several benefits:

1) The approach captures several different behaviors influencing collision risk, which are CD&R systems with time-varying detection rates, pilot delays, component failures of the CD&R systems, and conflict geometry.

2) Computation of the evaluation methods are relatively quick. As an example, one of the most complicated case studies, which modeled collision risk between manned and manned aircraft under AFM operations, is analyzed in 20 seconds on a personal laptop.

3) The approach takes benefit of creating or modifying a model and of evaluation of the new model. In a typical system design phase, for example, adding a redundant component, replacing a better component, or even a new architecture needs to be evaluated to choose the best design of a system that meets requirement of system reliability/safety. The DET approach proposed in this dissertation can be used to evaluate design alternatives of a system readily with a reasonable fidelity.

This dissertation also presented a way to apply the dynamic event tree framework to evaluate collision risk between aircraft equipped with different levels of collision avoidance capability. Several case studies were developed for collision risk between various types of aircraft flying under one of the concepts of current or future NAS operation, such as current air traffic control (ATC), Autonomous Flight Management (AFM), and Advanced Airspace Concept (AAC). Manned-unmanned aircraft pairs and manned-manned aircraft pairs were considered in the case studies. Unmanned aircraft were assumed to be equipped either with an onboard radar and a transponder or with an onboard radar and TCAS-like system.

Parameters of the conflict detection and resolution (CD&R) systems for each NAS operation, which may have a different architecture in terms of hardware and software, were studied. Fault trees were constructed to model failure relationships between physical components of each CD&R system. Time varying probabilities in which each CD&R system performs its function successfully were estimated based on Paielli [1997]. A way to apply the DET framework considering dependency between aircraft on a collision course was also discussed. In addition, sensitivity analyses on the model parameters of the case studies were conducted such as supporting components, detection range of sensor, and trajectory prediction errors of the algorithm used to estimate conflict detection probabilities.

Collision risk of the case studies for manned and unmanned aircraft pairs increase with angles between flight paths since closing speed between aircraft increases, thus the time available to avoid a collision decrease. The transponder on the unmanned aircraft is the most significant component under AFM and AAC operation, while the transponder on the manned aircraft is also important to the collision risk in AAC environment. The maximum detection range of onboard radar also affects collision risk significantly, especially when two aircraft are approaching with a path-crossing angle of about 90°.

Through the case study for a manned aircraft pair, the overall collision risk in the current NAS is approximately estimated in an order of 10^{-8} per flight hour or less, which is a product of the rate that two aircraft are on a collision course ($10^{-3} \sim 10^{-4}$ per flight hour estimated in Belle [2012]) and the (conditional) collision probability of this research (about 10^{-5}). For ATC and AAC operations, the collision risk between manned and

manned aircraft is high because of dependency on the transponders of either aircraft, while the collision risk of AFM operations is much less than the risk of the other two since failure of a single component does not cause all CD&R systems on both aircraft fail.

Case studies indicate that the reliability of aircraft transponders significantly drives collision risk since the CD&R systems and concepts considered in this dissertation highly rely on the transponders for surveillance. Due to the dependency on the transponders, the AAC concept of operations may not provide sufficient improvement in collision risk for manned-manned aircraft pairs compared to the current NAS operation, whereas the AFM operation shows much improved (i.e., reduced) collision risk because of less dependency on the transponders. Collision risk between manned and unmanned aircraft, compared to the risk between manned aircraft pairs, increases significantly under the AFM operations, but is not so changed in the AAC environment because the collision risk is already high for manned aircraft pair.

5.2 Future Work

This research in this dissertation can be extended in several ways. First, the methodology can be used in an applied setting to evaluate safety requirements for new types of aircraft including collision avoidance systems and separation assurance systems. Better system designs can be suggested by answering several "What-if" questions such as "what if there is a redundant component" or "what if we replace a component with a more reliable one". Second, more specific algorithms to detect and resolve conflicts can be used for better estimations of the model parameters and/or incorporating real data for

failure rates of components. While many of the model parameters come from the literature, some are simply assumed values in the case studies. Next, aircraft dynamics can be considered in the approach to add more fidelity. To do so, aircraft response delay (distribution function) to avoid a collision given a conflict geometry and aircraft type needs to be researched, then it can be included within a sub-tree in the framework, similar to pilot delay. Lastly, the methodology can be improved to relax the assumption that component failures occur at the start of each CD&R phase. An improved evaluation method could allow components to fail at each time step, hopefully without losing the advantage of fast computation.

APPENDIX A: DET FRAMEWORK USER GUIDE

This appendix describes steps that users should follow to set up a dynamic event tree (DET) for a mid-air-collision proposed in this dissertation. Inputs to define a DET framework are presented, and the high-level algorithm flow and corresponding function for each step are given.

A.1 Inputs for DET Framework

Three types of inputs coded in the main algorithm or as a delimited text file are explained with a graphical example. Then, an additional input that is required to apply the framework for a pair of aircraft on a collision course is described.

A.1.1 High-level tree structure

In order to define the high-level dynamic event tree, the following parameters are required:

- Time horizon for each phase, T_1, T_2, …, T_r (with $T_1 > T_2 > \dots T_r > 0$)
- Time step Δt

Figure A-1 shows an example input statement for high-level tree structure in the main algorithm. Variable ‘time-horizon’ defines time (T_r) at which each CD&R phase begins prior to a collision, and ‘deltaT’ specifies a computation time step (Δt) used

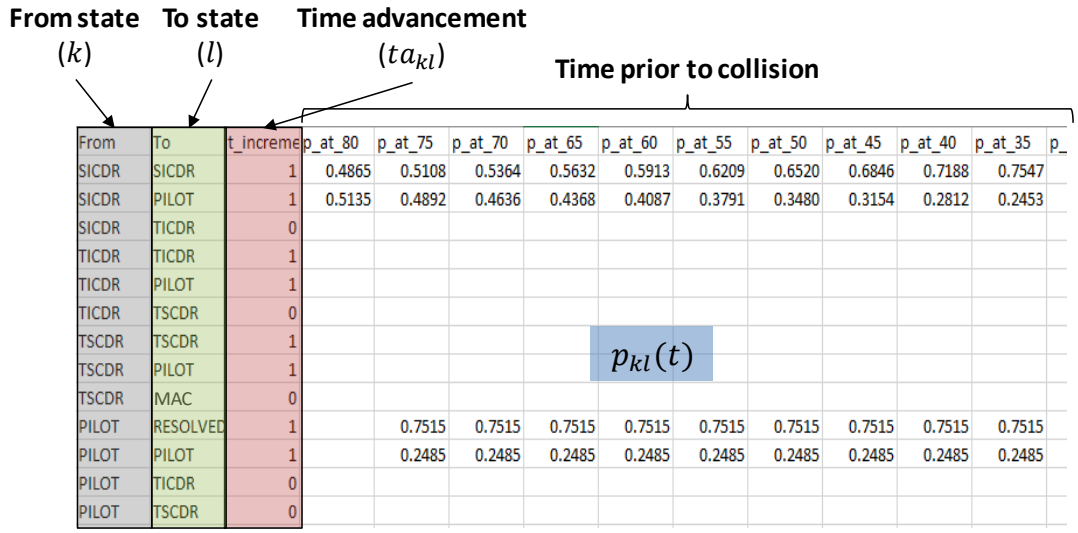
throughout the analysis. (Note that time horizon (T) and number of phases (r) can be derived from the above parameters.)

```
% High-level tree structure
time_horizon = [480; 180; 60];
deltaT = [1];
```

Figure A-1 Example input statement of high-level tree structure

A.1.2 Sub-tree structure

The sub-tree shows a template for evaluating a sequence of events within each CD&R phase. In order to specify the sub-tree, a set of state transitions (k, l) and associated time advancement for each transition (TA_{kl}, where $TA_{kl} = 0$ if transition from state k to l occurs instantly; $TA_{kl} = 1$ if transition from state k to l takes a time step) are firstly set. Figure A-2 is an example input file for the sub-tree structure. Each row specifies a transition between two states that can occur during conflict detection and resolution procedures. States include each CD&R system state (SICDR, TICDR, and TSCDR in the figure), a state where a resolution is provided to pilot (PILOT in the figure), a mid-air collision state (MAC), and a state in which a conflict is resolved (RESOLVED). State transition probabilities at different time ($p_{kl}(t)$) are generated with either pre-determined numerical values (i.e., inputs like the figure) or probabilistic distribution functions that can be defined in the main algorithm, e.g., time-dependent conflict-detection rate $\mu(t)$ and pilot execution rate γ. For the latter case (i.e., defined in the algorithm), the input file has empty cells after the third column, i.e., it needs to specify only a set of state transitions and associated time advancement.



From	To	t_increme	p_at_80	p_at_75	p_at_70	p_at_65	p_at_60	p_at_55	p_at_50	p_at_45	p_at_40	p_at_35	p_
SICDR	SICDR	1	0.4865	0.5108	0.5364	0.5632	0.5913	0.6209	0.6520	0.6846	0.7188	0.7547	
SICDR	PILOT	1	0.5135	0.4892	0.4636	0.4368	0.4087	0.3791	0.3480	0.3154	0.2812	0.2453	
SICDR	TICDR	0											
TICDR	TICDR	1											
TICDR	PILOT	1											
TICDR	TSCDR	0											
TSCDR	TSCDR	1											
TSCDR	PILOT	1											
TSCDR	MAC	0											
PILOT	RESOLVED	1		0.7515	0.7515	0.7515	0.7515	0.7515	0.7515	0.7515	0.7515	0.7515	
PILOT	PILOT	1		0.2485	0.2485	0.2485	0.2485	0.2485	0.2485	0.2485	0.2485	0.2485	
PILOT	TICDR	0											
PILOT	TSCDR	0											

Figure A-2 Example input file of sub-tree structure

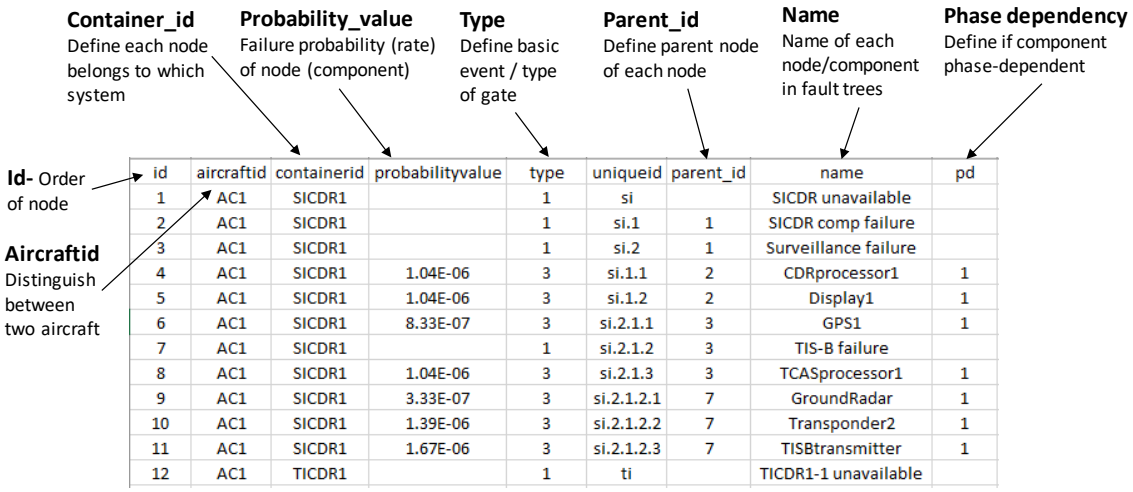
A.1.3 Fault tree structure

Fault trees model logical relationships between physical components and the functional failures of the CD&R systems. It is required to create an input file for the fault trees supporting the CD&R systems. Figure A-3 shows an example input file for the fault trees. Each row defines a node (i.e., basic event node or gate event node) in fault trees with several elements as follows:

- ID: Order of node in file
- Aircraft_id: Identify which CD&R system equipped on which aircraft
- Container_id: Identify which node belongs which CD&R system
- Probability_value: Failure probability/rate for component. Required only for basic event nodes.
- Type: Specify a type of a node with ‘0’ for AND-gate event, ‘1’ for OR-gate event, and ‘3’ for basic event.

- Parent_id: Specify a parent node of a node. The column for the parent node is empty if the node is a top event of a fault tree.
- Name: Provide name of component or gate event.
- Phase_dependency (pd): Specify if a component is phase-dependent or not (1- phase-dependent, 0- phase-independent). Required only for basic event nodes.

(Note: There is another element called 'uniqueid', which is not required for the analysis, but it is helpful to build the input file for fault trees because it provides information of a parent-child pairs that is easy to understand.)



id	aircraftid	containerid	probabilityvalue	type	uniqueid	parent_id	name	pd
1	AC1	SICDR1		1	si		SICDR unavailable	
2	AC1	SICDR1		1	si.1	1	SICDR comp failure	
3	AC1	SICDR1		1	si.2	1	Surveillance failure	
4	AC1	SICDR1	1.04E-06	3	si.1.1	2	CDRprocessor1	1
5	AC1	SICDR1	1.04E-06	3	si.1.2	2	Display1	1
6	AC1	SICDR1	8.33E-07	3	si.2.1.1	3	GPS1	1
7	AC1	SICDR1		1	si.2.1.2	3	TIS-B failure	
8	AC1	SICDR1	1.04E-06	3	si.2.1.3	3	TCASprocessor1	1
9	AC1	SICDR1	3.33E-07	3	si.2.1.2.1	7	GroundRadar	1
10	AC1	SICDR1	1.39E-06	3	si.2.1.2.2	7	Transponder2	1
11	AC1	SICDR1	1.67E-06	3	si.2.1.2.3	7	TISBtransmitter	1
12	AC1	TICDR1		1	ti		TICDR1-1 unavailable	

Figure A-3 Example input file of fault tree structure

A.1.4 Information for conflict

Once a DET framework for each aircraft is defined, conflict information between two aircraft such as speed, conflict geometry, and onboard sensor range needs to be specified (Figure A-4).

```
%% Conflict information (speed, conflict geometry, sensor range, etc.)
AC(1).spd = 400;  % knots
AC(2).spd = 170;
AC(1).heading = 90;  % degrees
AC(2).heading = 0;
AC(1).range = 240;    % onboard sensor range (km)
AC(2).range = 35;
```

Figure A-4 Example input of conflict information

A.2 High-level Algorithm Flow

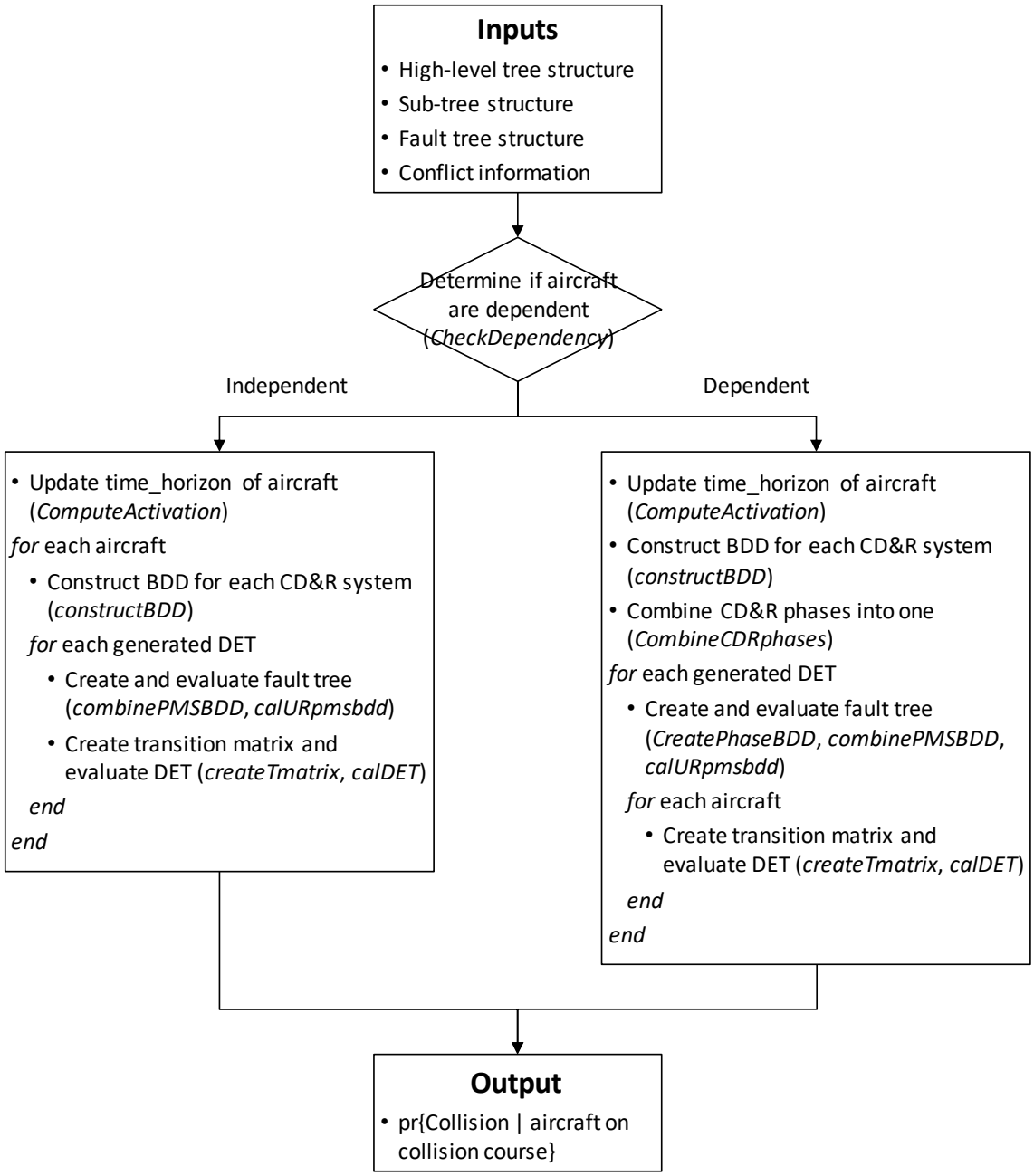


Figure A-5 High-level algorithm flow and associated functions

A.3 Example Input Files

Sub-tree structure (StateTransition-AC1.csv)

```
From,To,t_advancement
SICDR1,SICDR1,1
SICDR1,PILOT1,1
SICDR1,TICDR1,0
TICDR1,TICDR1,1
TICDR1,PILOT1,1
TICDR1,TSCDR1,0
TSCDR1,TSCDR1,1
TSCDR1,PILOT1,1
TSCDR1,NMAC,0
PILOT1,RESOLVED,1
PILOT1,PILOT1,1
PILOT1,TICDR1,0
PILOT1,TSCDR1,0
```

Fault tree structure (FaultTree-AC1.csv)

```
id,aircraftid,containerid,probabilityvalue,type,uniqueid,parent_id,name
,pd
1,AC1,SICDR1,,1,si,,SICDR unavailable,
2,AC1,SICDR1,,1,si.1,1,SICDR comp failure,
3,AC1,SICDR1,,1,si.2,1,Surveillance failure,
4,AC1,SICDR1,1.04E-06,3,si.1.1,2,CDRprocessor1,1
5,AC1,SICDR1,1.04E-06,3,si.1.2,2,Display1,1
8,AC1,SICDR1,8.33E-07,3,si.2.1.1,3,GPS1,1
9,AC1,SICDR1,,1,si.2.1.2,3,TIS-B failure,
10,AC1,SICDR1,1.04E-06,3,si.2.1.3,3,TCASprocessor1,1
11,AC1,SICDR1,3.33E-07,3,si.2.1.2.1,9,GroundRadar,1
12,AC1,SICDR1,1.39E-06,3,si.2.1.2.2,9,Transponder2,1
13,AC1,SICDR1,1.67E-06,3,si.2.1.2.3,9,TISBtransmitter,1
18,AC1,TICDR1,,1,ti,,TICDR1-1 unavailable,
19,AC1,TICDR1,,1,ti.1,18,TICDR1-1 comp failure,
20,AC1,TICDR1,,1,ti.2,18,Surveillance failure,
22,AC1,TICDR1,1.04E-06,3,ti.1.1,19,CDRprocessor1,1
23,AC1,TICDR1,,0,ti.1.2,19,Alerting failure,
24,AC1,TICDR1,1.04E-06,3,ti.1.2.1,23,Display1,1
25,AC1,TICDR1,1.04E-06,3,ti.1.2.2,23,SPK1,1
28,AC1,TICDR1,8.33E-07,3,ti.2.1.1,20,GPS1,1
29,AC1,TICDR1,,1,ti.2.1.2,20,TIS-B failure,
30,AC1,TICDR1,1.04E-06,3,ti.2.1.3,20,TCASprocessor1,1
31,AC1,TICDR1,3.33E-07,3,ti.2.1.2.1,29,GroundRadar,1
32,AC1,TICDR1,1.39E-06,3,ti.2.1.2.2,29,Transponder2,1
33,AC1,TICDR1,1.67E-06,3,ti.2.1.2.3,29,TISBtransmitter,1
45,AC1,TSCDR1,,1,ts1,,TSCDR unavailable,
46,AC1,TSCDR1,1.04E-06,3,ts1.1,45,TCASprocessor1,1
47,AC1,TSCDR1,,0,ts1.2,45,TCAS Alerting failure,
48,AC1,TSCDR1,1.39E-06,3,ts1.3,45,Transponder2,1
49,AC1,TSCDR1,1.04E-06,3,ts1.2.1,47,TCASdisplay1,1
50,AC1,TSCDR1,1.04E-06,3,ts1.2.2,47,TCASspk1,1
51,AC1,TSCDR1,1.39E-06,3,ts1.0,45,Transponder1,1
```

A.4 MATLAB Code

DET main.m

```
clear;
% Define DET structure for each aircraft
% create aircraft structure
AC = struct('time_horizon','');

% high-level tree
AC(1).time_horizon = [480 ; 180 ; 60];
AC(2).time_horizon = [480];   % calculated and updated depending on
speed, conflict geometry
deltaT = 1;

% sub-tree
AC(1).StateTransition =
table2cell(readtable('StateTransition_AC1_base.csv'));
AC(2).StateTransition =
table2cell(readtable('StateTransition_AC2_base.csv'));
% some of information for transition prob.
AC(1).pilotRate = [1.84501845; 2.909796314; 12];  % base pilot
execution rates (/min)
AC(2).pilotRate = [1.84501845];
AC(1).rate = 1.0;   % performance of detection algorithm
AC(2).rate = 0.3;   % 30% of performance of detection algorithm

% fault tree
AC(1).faultTree = loadTable('FaultTree_AC1_base.csv', 'allString', ',',
1);
AC(2).faultTree = loadTable('FaultTree_AC2_base_dep.csv', 'allString',
',', 1);
for i = 1 : length(AC)
    for j = 1 : length(AC(i).faultTree)
        if ~isempty(AC(i).faultTree(j).probabilityvalue)
            AC(i).faultTree(j).probabilityvalue =
str2double(AC(i).faultTree(j).probabilityvalue);
        end
    end
end

%% Conflict information (speed, conflict geometry, sensor range, etc.)
AC(1).spd = 400;  % knots
AC(2).spd = 170;
AC(1).heading = 90;  % degrees
AC(2).heading = 0;
AC(1).err = [15; 2];  % error level for conflict detection (along-track,
cross-track)
AC(2).err = [30; 4];
AC(1).range = 240;    % onboard sensor range (km)
AC(2).range = 35;

%% Compute collision prob. given two aircraft on a collision course
```

```
% check if aircraft are dependent
if length(AC) > 1
    [Dep] = CheckDependency(AC(1).faultTree, AC(2).faultTree);  % 0-
independent, 1-dependent
else
    Dep = 0;
end

% Call main algorithm
if Dep < 1
    [CollisionProb] = Independent(AC, deltaT);    % if aircraft are
independent
else
    [CollisionProb] = Dependent(AC, deltaT);      % if aircraft are
dependent
end
```

loadTable.m

```
function table = loadTable(fileName, dataTypes, delimiter,
headerRowNumber)
% loadTable loads a CSV or other text file into a data structure
if ~exist('delimiter','var')
    delimiter = ',';
end

if ~exist('headerRowNumber','var')
    headerRowNumber = 1;
end

if strcmp(dataTypes,'allString')
    % Count how many columns.
    fid = fopen(fileName);
    skipLines(fid, headerRowNumber-1);
    headerLine = fgetl(fid);
    fclose(fid);
    numCols = numel(strfind(headerLine, delimiter))+1;
    dataTypes = repmat('%s ', 1, numCols);
end

% Read the data.
fid = fopen(fileName);
skipLines(fid, headerRowNumber-1);
numCols = numel(strfind(dataTypes, '%'));
headerFormatString = repmat('%s ', 1, numCols);
headerData = textscan(fid, headerFormatString, 1, 'Delimiter',
delimiter);
data = textscan(fid, dataTypes, 'Delimiter', delimiter);
fclose(fid);

% Put it into the output struct format.
table = struct;
for col_idx = 1:size(data,2)
```

```
    colName = headerData{col_idx}{1};
    colName = replaceIllegalFieldNameCharacters(colName);
    if isempty(colName)
        % Ignore empty column names (and allow for comma at end of each
row).
        continue;
    end
    colData = data{col_idx};
    if iscell(colData)
        for row_idx = 1:numel(colData)
            table(row_idx).(colName) = colData{row_idx};
        end
    else
        for row_idx = 1:numel(colData)
            table(row_idx).(colName) = colData(row_idx);
        end
    end
end

end

function skipLines(fid, numLines)
% Skip lines above the header
for extraLineNum = 1:numLines
    fgetl(fid);
end
end

function s = replaceIllegalFieldNameCharacters(s)
s = strrep(s, ' ', '_');
s = strrep(s, '(', '');
s = strrep(s, ')', '');
s = strrep(s, '/', '');
s = strrep(s, '-', '');
s = strrep(s, '#', 'num_');
end
```

CheckDependency.m

```
function [Dep] = CheckDependency(FT_A, FT_B)
Dep = 0;
% Find event names in common in both FT_A and FT_B
indexA = find(strcmp({FT_A.type},'3'));
indexB = find(strcmp({FT_B.type},'3'));
basicA = FT_A(indexA);
basicB = FT_B(indexB);
for i = 1 : length(basicA)
    index = find(strcmp({basicB.name},basicA(i).name));
    if ~isempty(index)
        Dep = 1;
        break;
    end
end
```

Independent.m

```
function [TotalFailureProb] = Independent(AC, deltaT);

    TotalFailureProb = 1.0;
    % adjust activation time
    if length(AC) > 1
        [AC] = ComputeActivation(AC, deltaT);
    end

    %% Compute CD&R failure prob. for each aircraft using cDET-PMS
    for i = 1 : length(AC)
        % construct BDD and DBDD for each system
        BE = struct([]);
        [SystemBDD, BE] = constructBDD(AC(i).faultTree, BE);
        for j = 1 : size(SystemBDD,2)
            [SystemBDD(j).dbdd] = DBDD(SystemBDD(j).bdd);
        end
        nSystem = size(SystemBDD, 2);

        % phase start time in second
        PST = [AC(i).time_horizon; 0];

        for j = 1 : nSystem
            PTD(j) = (PST(j)-PST(j+1))/60;   % phase duraion in min
        end

        % crete initial phasebdd with phase names
        phasebdd = struct('name',[],'bdd',[]);
        for j = 1 : nSystem
            phasebdd(j).name = SystemBDD(j).name;
            PhaseConf(2*j-1) = {phasebdd(j).name};
            if j < nSystem
                PhaseConf(2*j) = {'0'};
            end
        end

        % each combination of available phases
        rowN = 1;
        for j = 0 : (pow2(nSystem)-1)
            b = dec2bin(j, nSystem); % [100], [010], ..., [110],[111]

            % create 'phasebdd'
            for k = 1 : nSystem
                a = str2num(b(k));
                CDRavailability(k) = a;
                if a < 1
                    phasebdd(k).bdd = SystemBDD(k).bdd;
                else
                    phasebdd(k).bdd = SystemBDD(k).dbdd;
                end
            end
```

```
            % weight prob. of each DET being used
            [cPMSBDD] = combinePMSBDD(PhaseConf, phasebdd);
            [UR] = calURpmsbdd(cPMSBDD, BE, PTD, PhaseConf);
            % save weight probabilities
            result(rowN,1) = UR;

            % create transition prob. and compute conditional prob. of
CD&R failures
            AC(1).rate = 1.0;
            AC(2).rate = 0.3;
            [sTransProb] = createTmatrix(AC(i).StateTransition, deltaT,
AC(i).system, PST, CDRavailability, AC(i).pilotRate, AC, AC(i).rate);
            [DET, EndState, states] = calDET(sTransProb);
            result(rowN,2) = EndState{1,3};
            rowN = rowN + 1;
        end

        % CD&R failure prob.
        FailureAC(1,i) = sum(result(:,1).*result(:,2));
        TotalFailureProb = TotalFailureProb * FailureAC(1,i);
        clear PhaseConf result PTD;
    end
end
```

ComputeActivation.m

```
function [AC] = ComputeActivation(AC, deltaT)

    % Extract unique fault trees
    for i = 1 : length(AC)
        FTCell = struct2cell(AC(i).faultTree);
        AC(i).system = unique(FTCell(3,:,:), 'stable');
    end

    % change units and calculate relative speed
    C = 1.852;  % km / nautical mile
    for i = 1 : length(AC)
        HD = AC(i).heading * pi() / 180;  % radian
        SPD = AC(i).spd * C / 60; % km / min
        V(i,1) = SPD*sin(HD);
        V(i,2) = SPD*cos(HD);
    end

    for j = 1 : size(V,2)
        rV(j) = V(2,j)-V(1,j);
    end
    rSPD = sqrt(rV*rV');    % relative speed (km/min)

    % compute activation times
    for i = 1 : length(AC)
        TRange(i) = AC(i).range / rSPD * 60;  % sec
        tempT = fix(TRange(i));   % in secs
        if tempT < AC(i).time_horizon(1)
```

```
            m = mod(tempT, deltaT);
            AC(i).time_horizon(1) = tempT - m;
        end
    end
end
```

constructBDD.m

```
function [SystemBDD, BE] = constructBDD(faultTreeStruct, BE)
    % Extract unique fault trees
    FTCell = struct2cell(faultTreeStruct);
    FTs = unique(FTCell(3,:,:), 'stable');

    q = CQueue();
    s = CStack();
    order = size(BE,2) + 1;
    BE(order).name = '';
    %% Convert a fault tree to a BDD
    for i = 1 : length(FTs)
        CurrFT =
faultTreeStruct(strcmp({faultTreeStruct.containerid},{FTs{i}}));
        SystemBDD(i).name = FTs{i};
        % find top event
        [TOP,Topindex] = findTOP(CurrFT);
        % assign each basic event an order in a manner of top-down and
left to right,then create ite array
        [ite, BE, CurrFT, order] = iteArray(CurrFT, TOP, BE, order);
        % Convert to fault tree that has only binary gates
        [CurrFT] = modify(CurrFT);
        % set an order for ite operation
        [s] = iteOrder(CurrFT, TOP);
        % ite operation in a manner of bottom-up
        while ~isempty(s)
            Cgate = s.pop;
            Gindex = find(strcmp({CurrFT.id},Cgate.id));
            op = Cgate.type;
            index = find(strcmp({CurrFT.parent_id},Cgate.id));
            F = CurrFT(index(1)).value; % row number in ite array
            G = CurrFT(index(2)).value;
            [ite, CurrFT] = convertBDD(CurrFT, Gindex, ite, op, F, G);
        end
        % Extract lines representing BDD from ite array
        s.empty();
        push(s, CurrFT(Topindex).value);    % row number for top node
of BDD
        [BDD] = extract(ite, s);
        % reduce rows of BDD table
        [BDD] = reduceTable(BDD);
        SystemBDD(i).bdd = BDD;
    end
    % Add phase and prob. to BE
    for j = 1 : size(BE,2)
        tprob = zeros(3,1);
        for i = 1 : length(FTs)
```

```
            CurrFT =
faultTreeStruct(strcmp({faultTreeStruct.containerid},{FTs{i}}));
            index = find(strcmp({CurrFT.name},{BE(j).name}));
            if ~isempty(index)
                tprob(i) = CurrFT(index(1)).probabilityvalue;
            else
                tprob(i) = 0;
            end
        end
        idx = find(~tprob); % find index for zero element
        idx0 = find(tprob); % find index for non-zero element
        for i = 1 : length(idx)
            tprob(idx(i)) = tprob(idx0(1));
        end
        BE(j).phase = FTs;
        BE(j).prob = tprob;
    end
end
```

findTOP.m

```
function [top,Topindex] = findTOP(CurrFT)
    n = 1;
    while ~isempty(CurrFT(n).parent_id)
        n = n + 1;
    end
    Topindex = n;
    top = CurrFT(Topindex);
end
```

iteArray.m

```
function [ite, BE, CurrFT, order] = iteArray(CurrFT, top, BE, order)

    q = CQueue();
    push(q,top);
    nRow = 1;
    ite = zeros(nRow,4);
    while ~q.isempty
        Cnode = q.pop;
        index = find(strcmp({CurrFT.parent_id},Cnode.id));
        for i = 1 : length(index)
            if CurrFT(index(i)).type == '3'
                check = find(strcmp({BE.name},CurrFT(index(i)).name));
                if isempty(check)
                    BE(order).name = CurrFT(index(i)).name;
                    if ~isnumeric(CurrFT(index(i)).pd)
                        CurrFT(index(i)).pd =
str2double(CurrFT(index(i)).pd);
                    end
                    BE(order).pd = CurrFT(index(i)).pd;
                    CurrFT(index(i)).value = order;
                    ite(order,1) = order;
                    ite(order,2) = order;
                    ite(order,3) = -1;
```

```
                    ite(order,4) = 0;
                    order = order + 1;
                    nRow = nRow + 1;
                else
                    CurrFT(index(i)).value = check(1);
                    if ite(:, 2) ~= check(1)
                        ite(check(1),1) = check(1);
                        ite(check(1),2) = check(1);
                        ite(check(1),3) = -1;
                        ite(check(1),4) = 0;
                        nRow = nRow + 1;
                    end
                end
            else
                push(q, CurrFT(index(i)));
            end
        end
        clear index;
    end
end
```

modify.m

```
function [CurrFT] = modify(CurrFT)
% convert a binary fault tree
q = CQueue();
n = 1;
while ~isempty(CurrFT(n).parent_id)
    n = n + 1;
end
top = CurrFT(n);
push(q,top);
l = length(CurrFT) + 1;
ng = 10001;
idn = 20001;

while ~isempty(q)
    Cnode = q.pop;
    index = find(strcmp({CurrFT.parent_id},Cnode.id));
    for i = 1 : length(index)
        if CurrFT(index(i)).type ~= '3'
            push(q,CurrFT(index(i)));
        end
    end
    while length(index) > 2
        CurrFT(l).id = num2str(idn);
        CurrFT(l).containerid = Cnode.containerid;
        CurrFT(l).type = Cnode.type;
        CurrFT(l).parent_id = Cnode.id;
        CurrFT(l).name = num2str(ng);
        for i = 2 : length(index)
            CurrFT(index(i)).parent_id = CurrFT(l).id;
        end
        clear index;
```

```
        index = find(strcmp({CurrFT.parent_id},CurrFT(l).id));
        Cnode = CurrFT(l);
        l = l + 1;
        ng = ng + 1;
        idn = idn + 1;
    end
end
```

iteOrder.m

```
function [s] = iteOrder(CurrFT, top)
    q = CQueue();
    s = CStack();
    push(q,top);
    push(s,top);
    while ~q.isempty
        Cnode = q.pop;
        index = find(strcmp({CurrFT.parent_id},Cnode.id));
        for i = 1 : length(index)
            if CurrFT(index(i)).type ~= '3'
                push(s, CurrFT(index(i)));
                push(q, CurrFT(index(i)));
            end
        end
        clear index;
    end
end
```

convertBDD.m

```
function [ite, CurrFT] = convertBDD(CurrFT, Gindex, ite, op, F, G)
l = length(ite) + 1;
% Determine which one has priority
% F and G are row numbers, a and b are variable numbers meaning order
a = ite(F, 2);
b = ite(G, 2);
if a > b    % b has a priority
    temp = a;
    a = b;
    b = temp;
    temp = F;
    F = G;
    G = temp;
end
if Gindex(1) ~= 0
    CurrFT(Gindex(1)).value = l;
end

ite(l, 1) = l;
ite(l, 2) = ite(F, 2);
if op == '0'    % 'AND' gate
    if a ~= b
        % compute 'then' value
        if ite(F, 3) == -1
            ite(l, 3) = ite(G, 1);
```

```
    elseif ite(F, 3) == 0
        ite(l, 3) = 0;
    else
        F1 = ite(F, 3);
        G1 = G;
        Gindex(1) = 0;
        ite(l, 3) = length(ite) + 1;
        [ite, CurrFT] = convertBDD(CurrFT,Gindex,ite,op,F1,G1);
    end
    % compute 'else' value
    if ite(F, 4) == -1
       ite(l, 4) = ite(G, 1);
    elseif ite(F, 4) == 0
        ite(l, 4) = 0;
    else
        F2 = ite(F, 4);
        G2 = G;
        Gindex(1) = 0;
        ite(l, 4) = length(ite) + 1;
        [ite, CurrFT] = convertBDD(CurrFT,Gindex, ite, op, F2, G2);
    end
elseif a == b
    % compute 'then' value
    if ite(F, 3) == -1
        ite(l, 3) = ite(G, 3);
    elseif ite(F, 3) == 0
        ite(l, 3) = 0;
    elseif ite(G, 3) == -1
        ite(l, 3) = ite(F, 3);
    elseif ite(G, 3) == 0
        ite(l, 3) = 0;
    else
        F1 = ite(F, 3);
        G1 = ite(G, 3);
        Gindex(1) = 0;
        ite(l, 3) = length(ite) + 1;
        [ite, CurrFT] = convertBDD(CurrFT,Gindex, ite, op, F1, G1);
    end
    % compute 'else' value
    if ite(F, 4) == -1
        ite(l, 4) = ite(G, 4);
    elseif ite(F, 4) == 0
        ite(l, 4) = 0;
    elseif ite(G, 4) == -1
        ite(l, 4) = ite(F, 4);
    elseif ite(G, 4) == 0
        ite(l, 4) = 0;
    else
        F2 = ite(F, 4);
        G2 = ite(G, 4);
        Gindex(1) = 0;
        ite(l, 4) = length(ite) + 1;
        [ite, CurrFT] = convertBDD(CurrFT,Gindex, ite, op, F2, G2);
    end
```

```
    end
elseif op == '1'    % 'OR' gate
    if a ~= b
        % compute 'then' value
        if ite(F, 3) == -1
            ite(l, 3) = -1;
        elseif ite(F, 3) == 0
            ite(l, 3) = ite(G, 1);
        else
            F1 = ite(F, 3);
            G1 = G;
            Gindex(1) = 0;
            ite(l, 3) = length(ite) + 1;
            [ite, CurrFT] = convertBDD(CurrFT,Gindex, ite, op, F1, G1);
        end
        % compute 'else' value
        if ite(F, 4) == -1
           ite(l, 4) = -1;
        elseif ite(F, 4) == 0
            ite(l, 4) = ite(G, 1);
        else
            F2 = ite(F, 4);
            G2 = G;
            Gindex(1) = 0;
            ite(l, 4) = length(ite) + 1;
            [ite, CurrFT] = convertBDD(CurrFT,Gindex, ite, op, F2, G2);
        end
    elseif a == b
        % compute 'then' value
        if ite(F, 3) == -1
            ite(l, 3) = -1;
        elseif ite(F, 3) == 0
            ite(l, 3) = ite(G, 3);
        elseif ite(G, 3) == -1
            ite(l, 3) = -1;
        elseif ite(G, 3) == 0
            ite(l, 3) = ite(F, 3);
        else
            F1 = ite(F, 3);
            G1 = ite(G, 3);
            Gindex(1) = 0;
            ite(l, 3) = length(ite) + 1;
            [ite, CurrFT] = convertBDD(CurrFT,Gindex, ite, op, F1, G1);
        end
        % compute 'else' value
        if ite(F, 4) == -1
            ite(l, 4) = -1;
        elseif ite(F, 4) == 0
            ite(l, 4) = ite(G, 4);
        elseif ite(G, 4) == -1
            ite(l, 4) = -1;
        elseif ite(G, 4) == 0
            ite(l, 4) = ite(F, 4);
        else
```

```
            F2 = ite(F, 4);
            G2 = ite(G, 4);
            Gindex(1) = 0;
            ite(l, 4) = length(ite) + 1;
            [ite, CurrFT] = convertBDD(CurrFT,Gindex, ite, op, F2, G2);
        end
    end
end
```

extract.m

```
function [BDD] = extract(ite, s)
while ~isempty(s)
    Cindex = s.pop;
    BDD(Cindex,:) = ite(Cindex,:);
    if ite(Cindex, 3) > 0
        push(s, ite(Cindex, 3));
    end
    if ite(Cindex, 4) > 0
        push(s, ite(Cindex, 4));
    end
end
```

reduceTable.m

```
function [BDD] = reduceTable(BDD)
pBDD = BDD;
clear BDD;
l = length(pBDD);
r = 1;
for i = 1 : l
    if pBDD(i, 1) > 0
        cNode = pBDD(i, 1);
        BDD(r, 1) = r;
        BDD(r, 2) = pBDD(i, 2);
        BDD(r, 3) = pBDD(i, 3);
        BDD(r, 4) = pBDD(i, 4);
        for j = 1 : l
            if pBDD(j, 3) == cNode
                pBDD(j, 3) = r;
            end
            if pBDD(j, 4) == cNode
                pBDD(j, 4) = r;
            end
        end
        sz = size(BDD);
        for j = 1 : sz(1)
            if BDD(j, 3) == cNode
                BDD(j, 3) = r;
            end
            if BDD(j, 4) == cNode
                BDD(j, 4) = r;
            end
        end
        r = r + 1;
```

```
    end
end

% delete the duplicated row if existing
for k = size(BDD, 1) : -1 : 1
    if BDD(k,2) ~= 0
        index = find(BDD(:, 2)==BDD(k, 2));
        if ~isempty(index)
            for j = 1 : length(index)
                if index(j) ~= k
                    if BDD(index(j), 3) == BDD(k, 3)
                        if BDD(index(j), 4) == BDD(k, 4)
                            BDD(find(BDD(:, 3)==index(j)),3) = k;
                            BDD(find(BDD(:, 4)==index(j)),4) = k;
                            BDD(index(j),:) = [0,0,0,0];
                        end
                    end
                end
            end
        end
        if BDD(k,3) > 0
            if BDD(k,3) == BDD(k,4)
                BDD(find(BDD(1:size(BDD, 1), 3)==k),3) = BDD(k,3);
                BDD(find(BDD(1:size(BDD, 1), 4)==k),4) = BDD(k,3);
                BDD(k,:) = [0,0,0,0];
            end
        end
    end
end
index0 = find(BDD(1:size(BDD, 1), 1)==0);
if ~isempty(index0)
    [BDD] = reduceTable(BDD);
end
```

DBDD.m

```
function [DBDD] = DBDD(BDD)
    szBDD = size(BDD);
    for b = 1 : szBDD(1)
        if BDD(b, 1) ~= 0
            DBDD(b, :) = BDD(b, :);
            if BDD(b, 3) == -1
                DBDD(b, 3) = 0;
            end
            if BDD(b, 4) == 0
                DBDD(b, 4) = -1;
            end
        end
    end
end
```

combinePMSBDD.m

```
function [cBDD] = combinePMSBDD(PhaseConf, Phase)
s = CStack();
```

```
len = size(PhaseConf);
cBDD = zeros(1,4);
for p = 1 : (len(2)+1)/2
    CurrPhase.name = PhaseConf{2*p-1};
    tempBDD = Phase(p).bdd;
    % assign phase order
    topnode(p) = 1000;
    for i = 1 : size(tempBDD,1)
        tempBDD(i, 2) = tempBDD(i, 2) + 100*p;
        if topnode(p) >= tempBDD(i,2)
            topnode(p) = tempBDD(i,2);
        end
    end
    % create cBDD to save combined BDD
    if cBDD(1,1) == 0
        cBDD = tempBDD;
    else
        sz = size(cBDD);
        rowN = sz(1);
        for j = 1 : size(tempBDD,1)
            cBDD(rowN + j, 1) = rowN + j;
            cBDD(rowN + j, 2) = tempBDD(j, 2);
            if tempBDD(j, 3) > 0
                cBDD(rowN + j, 3) = tempBDD(j, 3) + rowN;
            else
                cBDD(rowN + j, 3) = tempBDD(j, 3);
            end
            if tempBDD(j, 4) > 0
                cBDD(rowN + j, 4) = tempBDD(j, 4) + rowN;
            else
                cBDD(rowN + j, 4) = tempBDD(j, 4);
            end
        end
        F = find(cBDD(1:rowN,2)==topnode(p-1));
        G = find(cBDD(rowN+1:rowN+j,2)==topnode(p))+rowN;
        op = PhaseConf{(p-1)*2};
        [cBDD] = operatePDO(cBDD, op, F, G);

        % Extract lines representing combined BDD from cBDD array
        s.empty();
        a = topnode(p-1) - fix(topnode(p-1) / 100)*100;
        b = topnode(p) - fix(topnode(p) / 100)*100;
        if a == b
            if topnode(p-1) > topnode(p)
                topnode(p) = topnode(p-1);
            end
        elseif a < b
            topnode(p) = topnode(p-1);
        end

        rowNtop = size(cBDD,1);
        while cBDD(rowNtop, 2)~=topnode(p)
            rowNtop = rowNtop - 1;
```

```
        end
        push(s, rowNtop);    % row number for top node of BDD
        [cBDD] = extract(cBDD, s);

        % reduce rows of BDD table
        [cBDD] = reduceTable(cBDD);
    end
end
```

operatePDO.m

```
function [BDD] = operatePDO(BDD, op, F, G)
l = size(BDD,1) + 1;
% Determine which one has priority
% F and G are row numbers, a and b are variable numbers meaning order
a.p = fix(BDD(F, 2) / 100); % phase
a.v = BDD(F, 2) - a.p * 100;    % variable
b.p = fix(BDD(G, 2) / 100);
b.v = BDD(G, 2) - b.p * 100;

BDD(l, 1) = l;
BDD(l, 2) = BDD(F, 2);
if a.v == b.v
    if a.p > b.p
        temp1 = a;
        a = b;
        b = temp1;
        temp2 = F;
        F = G;
        G = temp2;
    end
    BDD(l, 2) = BDD(G, 2);
elseif a.v > b.v
    temp1 = a;
    a = b;
    b = temp1;
    temp2 = F;
    F = G;
    G = temp2;
    BDD(l, 2) = BDD(F, 2);
end

if (a.v == b.v) && (a.p ~= b.p)     % phase dependent operation
    if op == '0'    % 'AND' gate
        % compute 'then' value
        if BDD(G, 3) == -1
            BDD(l, 3) = BDD(F, 1);
        elseif BDD(G, 3) == 0
            BDD(l, 3) = 0;
        else
            F1 = BDD(F, 1);
            G1 = BDD(G, 3);
            BDD(l, 3) = size(BDD,1) + 1;
            [BDD] = operatePDO(BDD, op, F1, G1);
```

```
        end
        % compute 'else' value
        if BDD(F, 4) == -1
            BDD(l, 4) = BDD(G, 4);
        elseif BDD(F, 4) == 0
            BDD(l, 4) = 0;
        elseif BDD(G, 4) == -1
            BDD(l, 4) = BDD(F, 4);
        elseif BDD(G, 4) == 0
            BDD(l, 4) = 0;
        else
            F2 = BDD(F, 4);
            G2 = BDD(G, 4);
            BDD(l, 4) = size(BDD,1) + 1;
            [BDD] = operatePDO(BDD, op, F2, G2);
        end
    else            % 'OR' gate
        % compute 'then' value
        if BDD(G, 3) == -1
            BDD(l, 3) = -1;
        elseif BDD(G, 3) == 0
            BDD(l, 3) = BDD(F, 1);
        else
            F1 = BDD(F, 1);
            G1 = BDD(G, 3);
            BDD(l, 3) = size(BDD,1) + 1;
            [BDD] = operatePDO(BDD, op, F1, G1);
        end
        % compute 'else' value
        if BDD(F, 4) == -1
            BDD(l, 4) = -1;
        elseif BDD(F, 4) == 0
            BDD(l, 4) = BDD(G, 4);
        elseif BDD(G, 4) == -1
            BDD(l, 4) = -1;
        elseif BDD(G, 4) == 0
            BDD(l, 4) = BDD(F, 4);
        else
            F2 = BDD(F, 4);
            G2 = BDD(G, 4);
            BDD(l, 4) = size(BDD,1) + 1;
            [BDD] = operatePDO(BDD, op, F2, G2);
        end
    end
else              % conventional BDD operation
    if op == '0'    % 'AND' gate
        if a.v ~= b.v
            % compute 'then' value
            if BDD(F, 3) == -1
                BDD(l, 3) = BDD(G, 1);
            elseif BDD(F, 3) == 0
                BDD(l, 3) = 0;
            else
                F1 = BDD(F, 3);
```

```
            G1 = G;
            BDD(l, 3) = size(BDD,1) + 1;
            [BDD] = operatePDO(BDD, op, F1, G1);
        end
        % compute 'else' value
        if BDD(F, 4) == -1
           BDD(l, 4) = BDD(G, 1);
        elseif BDD(F, 4) == 0
            BDD(l, 4) = 0;
        else
            F2 = BDD(F, 4);
            G2 = G;
            BDD(l, 4) = size(BDD,1) + 1;
            [BDD] = operatePDO(BDD, op, F2, G2);
        end
    elseif a.v == b.v
        % compute 'then' value
        if BDD(F, 3) == -1
            BDD(l, 3) = BDD(G, 3);
        elseif BDD(F, 3) == 0
            BDD(l, 3) = 0;
        elseif BDD(G, 3) == -1
            BDD(l, 3) = BDD(F, 3);
        elseif BDD(G, 3) == 0
            BDD(l, 3) = 0;
        else
            F1 = BDD(F, 3);
            G1 = BDD(G, 3);
            BDD(l, 3) = size(BDD,1) + 1;
            [BDD] = operatePDO(BDD, op, F1, G1);
        end
        % compute 'else' value
        if BDD(F, 4) == -1
            BDD(l, 4) = BDD(G, 4);
        elseif BDD(F, 4) == 0
            BDD(l, 4) = 0;
        elseif BDD(G, 4) == -1
            BDD(l, 4) = BDD(F, 4);
        elseif BDD(G, 4) == 0
            BDD(l, 4) = 0;
        else
            F2 = BDD(F, 4);
            G2 = BDD(G, 4);
            BDD(l, 4) = size(BDD,1) + 1;
            [BDD] = operatePDO(BDD, op, F2, G2);
        end
    end
else            % 'OR' gate
    if a.v ~= b.v
        % compute 'then' value
        if BDD(F, 3) == -1
            BDD(l, 3) = -1;
        elseif BDD(F, 3) == 0
            BDD(l, 3) = BDD(G, 1);
```

```
            else
                F1 = BDD(F, 3);
                G1 = G;
                BDD(l, 3) = size(BDD,1) + 1;
                [BDD] = operatePDO(BDD, op, F1, G1);
            end
            % compute 'else' value
            if BDD(F, 4) == -1
               BDD(l, 4) = -1;
            elseif BDD(F, 4) == 0
                BDD(l, 4) = BDD(G, 1);
            else
                F2 = BDD(F, 4);
                G2 = G;
                BDD(l, 4) = size(BDD,1) + 1;
                [BDD] = operatePDO(BDD, op, F2, G2);
            end
        elseif a.v == b.v
            % compute 'then' value
            if BDD(F, 3) == -1
                BDD(l, 3) = -1;
            elseif BDD(F, 3) == 0
                BDD(l, 3) = BDD(G, 3);
            elseif BDD(G, 3) == -1
                BDD(l, 3) = -1;
            elseif BDD(G, 3) == 0
                BDD(l, 3) = BDD(F, 3);
            else
                F1 = BDD(F, 3);
                G1 = BDD(G, 3);
                BDD(l, 3) = size(BDD,1) + 1;
                [BDD] = operatePDO(BDD, op, F1, G1);
            end
            % compute 'else' value
            if BDD(F, 4) == -1
                BDD(l, 4) = -1;
            elseif BDD(F, 4) == 0
                BDD(l, 4) = BDD(G, 4);
            elseif BDD(G, 4) == -1
                BDD(l, 4) = -1;
            elseif BDD(G, 4) == 0
                BDD(l, 4) = BDD(F, 4);
            else
                F2 = BDD(F, 4);
                G2 = BDD(G, 4);
                BDD(l, 4) = size(BDD,1) + 1;
                [BDD] = operatePDO(BDD, op, F2, G2);
            end
        end
    end
end
```

calURpmsbdd.m

```
function [UR] = calURpmsbdd(BDD, BE, PTD, PhaseConf)
    % set the order bottom-up to compute probability
    q = CQueue();
    sz = size(BDD);
    phase = fix(BDD(:,2) / 100);
    variable = BDD(:,2) - phase * 100;
    M = max(variable);
    temp = 0;
    for i = 2 : size(PTD,2)
        temp = temp + PTD(i);
    end
    PTD(1) = 8 - temp;
    while M ~= 0
        V = find(variable==M);
        m = min(phase(V));
        while m ~= 100
            rown = find(phase(V)==m);
            for i = 1 : length(rown)
                q.push(V(rown(i)));
            end
            phase(V(rown)) = 100;
            variable(V(rown)) = 0;
            m = min(phase(V));
        end
        M = max(variable);
    end

    % Compute probability of each node
    BDD(:,5) = -1;
    for i = 1 : size(BDD,1)
        if BDD(i, 3) == -1
            if BDD(i, 4) == 0
                BDD(i, 5) = 1;
            end
        end
    end
    while ~isempty(q)
        G.r = q.pop;
        G1.r = BDD(G.r,3);
        G2.r = BDD(G.r,4);
        G.p = fix(BDD(G.r, 2) / 100); % phase
        G.v = BDD(G.r, 2) - G.p * 100;    % variable
        [prob] = PDprob(G, BE, PTD, PhaseConf);
        if G1.r > 0
            G1.p = fix(BDD(G1.r, 2) / 100); % phase
            G1.v = BDD(G1.r, 2) - G1.p * 100;    % variable
            if G.v == G1.v
                H2.r = BDD(G1.r,4);
                if G2.r > 0
                    if H2.r > 0
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) *
(BDD(G2.r,5)-BDD(H2.r,5));
```

```
                    elseif H2.r < 0
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) *
(BDD(G2.r,5)-1);
                    else
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) *
(BDD(G2.r,5)-0);
                    end
                elseif G2.r < 0
                    if H2.r > 0
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) * (1-
BDD(H2.r,5));
                    elseif H2.r < 0
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) * (1-1);
                    else
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) * (1-0);
                    end
                else
                    if H2.r > 0
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) * (0-
BDD(H2.r,5));
                    elseif H2.r < 0
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) * (0-1);
                    else
                        BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) * (0-0);
                    end
                end
            else
                if G2.r > 0
                    BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) *
(BDD(G2.r,5)-BDD(G1.r,5));
                else
                    BDD(G.r,5) = BDD(G1.r,5) + (1 - prob) *
(power(G2.r,2)-BDD(G1.r,5));
                end
            end
        else
            if G2.r > 0
                BDD(G.r,5) = power(G1.r,2) + (1 - prob) * (BDD(G2.r,5)-
power(G1.r,2));
            else
                BDD(G.r,5) = power(G1.r,2) + (1 - prob) *
(power(G2.r,2)-power(G1.r,2));
            end
        end
    end
    UR = BDD(G.r,5);
end

function [prob] = PDprob(G, BE, PTD, PhaseConf)
    tPhase = BE(G.v).phase;
    tProb = BE(G.v).prob;
    variable = G.v;
    phase = G.p;
```

```
    phaseName = PhaseConf(G.p*2-1);
    phaseindex = find(strcmp(tPhase,phaseName));
    PhaseDependency = BE(variable).pd;
    if PhaseDependency < 1
        lamda = tProb(phaseindex);
        PhaseDuration = PTD(phase);
        prob = 1 - exp(-lamda*PhaseDuration);
        if PhaseDependency < 0
            prob = 1 - prob;
        end
    else
        temp = 1;
        if phase > 1
            for i = 1 : (phase - 1)
                index = find(strcmp(tPhase,PhaseConf{i*2-1}));
                lamda = tProb(index);
                PhaseDuration = PTD(i);
                temp = temp * exp(-lamda*PhaseDuration);
            end
        end
        lamda = tProb(phaseindex);
        PhaseDuration = PTD(phase);
        prob = 1 - temp + temp * (1 - exp(-lamda*PhaseDuration));
    end
end
```

createTmatrix.m

```
%% Create transition matrix
function [StateTransition] = createTmatrix(StateTransition, dt, Phases,
PST, cdrRateAC, pilotRate, AC, ratio)
    CdrRate = zeros(480,1);
    states = unique(StateTransition(:,1:2),'stable');
    % find phase = state
    for i = 1 : length(Phases)
        idx = find(strcmp(states(:,1), Phases{i}));
        if ~isempty(idx)
            states{idx, 2} = 1;
        end
    end

    % find end-states
    n = 0;
    for i = 1 : length(states)
        if isempty (find(strcmp(StateTransition(:,1), states{i})))
            n = n + 1;
            end_states{n,1} = states{i};
            states{i,2} = 3;
        else
            % intemediate state
            if isempty(states{i,2})
                states{i,2} = 2;
            end
        end
```

```
end

for i = 1 : length(states)
    if states{i,2} == 1 % CD&R state
        fState = states{i,1};
        PhaseNum = find(strcmp(Phases(:,1), fState));
        idx = find(strcmp(StateTransition(:,1), fState));
        for j = 1 : length(idx)
            tState = StateTransition{idx(j),2};
            if strcmp(fState, tState)   % failure
                fidx = idx(j);
            elseif states{strcmp(states(:,1), tState),2}== 2
                sidx = idx(j);
            else
                tidx = idx(j);  % transition between phases
            end
        end
        for j = 1 : size(PhaseNum, 1)
            cCol = (PST(1) - PST(PhaseNum(j)))/dt + 4;
            nCol = (PST(PhaseNum(j)) - PST(PhaseNum(j)+1))/dt;
            clock = PST(PhaseNum(j));
            for k = 1 : nCol
                if cdrRateAC(PhaseNum(j)) > 0
                    [cdrRate] = ConflictDetectionProb(AC, clock);
                    cdrRate = cdrRate * ratio;
                    prob = 1 - exp(-cdrRate*dt/60);
                else
                    prob = 0;
                end
                StateTransition{fidx, cCol} = 1-prob;
                StateTransition{sidx, cCol} = prob;
                cCol = cCol + 1;
                clock = clock - dt;
            end
        end
        StateTransition{tidx, cCol} = 1;
    elseif states{i,2} == 2 % intermediate state (pilot state)
        fState = states{i,1};
        idx = find(strcmp(StateTransition(:,1), fState));
        for j = 1 : length(idx)
            tState = StateTransition{idx(j),2};
            if strcmp(fState, tState)
                fidx = idx(j);
            elseif states{strcmp(states(:,1), tState),2} == 3
                sidx = idx(j);
            end
        end
        for k = 1 : size(Phases,1)
            PhaseNum = k;
            clock = PST(PhaseNum);
            cCol = (PST(1) - PST(PhaseNum))/dt + 5;
            pRate = pilotRate(k) / 60;
            prob = 1 - exp(-pRate*dt);
```

```
                while clock > PST(PhaseNum+1)
                    StateTransition{fidx, cCol} = 1-prob;
                    StateTransition{sidx, cCol} = prob;
                    cCol = cCol + 1;
                    clock = clock - dt;
                end

                if PhaseNum < size(Phases,1)
                    if  ~strcmp(Phases{PhaseNum,1},  Phases{PhaseNum+1,1})
                        cCol = (PST(1) - PST(PhaseNum+1))/dt + 4;
                        StateTransition{fidx, cCol} = [];
                        StateTransition{sidx, cCol} = [];
                        tidx = find(strcmp(StateTransition(idx,2),
Phases{PhaseNum+1,1}));
                        StateTransition{idx(tidx), cCol} = 1;
                    end
                else
                    cCol = (PST(1) - PST(PhaseNum+1))/dt + 4;
                    StateTransition{fidx, cCol} = [];
                    StateTransition{sidx, cCol} = [];
                end
                clear tidx;
            end
        end
    end
end
```

ConflictDetectionProb.m

```
function [cdrRate] = ConflictDetectionProb(AC, clock)
% Paielli (1996)
    mps0 = [0; 100/6076.12; 500/6076.12; 1.1; 5];
    theta = (AC(1).heading - AC(2).heading) * pi/180;
    mps = mps0(2)*[cos(theta);sin(theta)];
    sc = 5;   %separation standard for conflict (nm)
    Vr = AC(1).spd / 60;    % reference (manned), nm/min
    Vs = AC(2).spd / 60; % stochastic (unmanned), nm/min
    R=[cos(theta) -sin(theta);sin(theta) cos(theta)];
    time = clock / 60; % (min)
    varRx=(AC(1).err(1) / 60)^2 * time^2;
    varRy=(AC(1).err(2) / 1.96)^2;% RNP X, 95% of time stay in +/- X nm
    varSx=(AC(2).err(1) / 60)^2 * time^2;
    varSy=(AC(2).err(2) / 1.96)^2;
    Ss=[varSx 0;0 varSy];
    Sr=[varRx 0;0 varRy];
    Qr=R*Sr*R';
    M = Ss + Qr;
    L11=sqrt(M(1,1));
    L21=M(2,1)/L11;
    L22=sqrt(M(2,2)-L21^2);
    L=[L11 0;L21 L22];
    mpsT = inv(L)*mps;
    dV = [Vs*cos(theta)-Vr; Vs*sin(theta)];
    dVt = inv(L)*dV;
```

```
    R1 = 1/sqrt(power(dVt(1),2)+power(dVt(2),2))*[dVt(1), dVt(2); -
dVt(2), dVt(1)];
    T=R1/L;
    W=inv(T);
    Wc=W'*W;
    a=Wc(1,1);
    b=Wc(1,2);
    c=Wc(2,2);
    yc=sc*sqrt(a/(a*c-b^2));
    pc=normcdf(-mpsT(2)+yc,0,1)-normcdf(-mpsT(2)-yc,0,1);

    cdrRate = -log(1-pc) / (clock/ 60); % /min
end
```

calDET.m

```
%% Dynamic Event Tree
function [DET, end_states, states] = calDET(transition)
    [nRow,nCol] = size(transition);
    states = unique(transition(:,1:2),'stable');
    transition = sortrows(transition,3);

    % find end-states
    n = 1;
    for i = 1 : length(states)
        if isempty (find(strcmp(transition(:,1), states{i})))
            end_states{n,1} = states{i};
            end_states{n,2} = i;  % index of end_state in states array
            end_states{n,3} = 0;
            n = n + 1;
        end
    end
    numEndStates = n - 1;

    %% Create dynamic event tree
    DET = cell(length(states), nCol-2);
    [m,n] = size(DET);
    for i = 1 : m
        for j = 1 : n
            DET{i,j} = 0;
        end
    end
    DET{1,1} = 1;
    for i = 1 : nCol-3   % time
        Sindex = find(~cellfun('isempty', transition(:,i+3)));
        nState = unique(transition(Sindex,1:2),'stable');
        while ~isempty(nState)
            cState = nState{1,1};
            nState(1,:) = [];
            index = find(strcmp(transition(:,1), cState));
            Findex = find(strcmp(states, cState));
            Eindex = find(strcmp(end_states, cState));
            if isempty(Eindex)
                for j = 1 : length(index)
```

```
                    PbTrans = transition{index(j),i+3};
                    if PbTrans > 0
                        toState = transition(index(j),2);
                        Tindex = find(strcmp(states, toState));
                        tAdv = transition{index(j),3
                        DET{Tindex(1),i+tAdv} = DET{Tindex(1),i+tAdv} +
DET{Findex(1),i} * PbTrans;
                    end
                end
            end
        end
        DET{end_states{2,2},i+1} = DET{end_states{2,2},i+1} +
DET{end_states{2,2},i};
    end
    %store end-state probabilities at time 0
    end_states{2,3} = DET{end_states{2,2},nCol-3};  % resolved
    end_states{1,3} = 1-DET{end_states{2,2},nCol-3};
end
```

Dependent.m

```
function [TotalFailureProb] = Dependent(AC, deltaT);

    % adjust activation time
    if length(AC) > 1
        [AC] = ComputeActivation(AC, deltaT);
    end

    % construct BDD and DBDD for each system
    BE = struct([]);
    faultTreeStruct = [AC(1).faultTree, AC(2).faultTree];
    [SystemBDD, BE] = constructBDD(faultTreeStruct, BE);
    for j = 1 : size(SystemBDD,2)
        [SystemBDD(j).dbdd] = DBDD(SystemBDD(j).bdd);
    end

    % combine CD&R phases into one
    [Combined] = CombineCDRphases(AC);

    % update phase name with combined one
    for i = 1 : size(BE,2)
        BE(i).phase = Combined.Phases;
    end

    % crete initial phasebdd with phase names
    phasebdd = struct('name',[],'bdd',[]);
    for i = 1 : size(Combined.Phases,1)
        phasebdd(i).name = Combined.Phases{i};
    end

    % each combination of available phases
    rowN = 1;
    nSystem = Combined.nSystems;
```

```
    for j = 0 : (pow2(nSystem)-1)
        b = dec2bin(j, nSystem); % [100], [010], ..., [110],[111]

        % checking if the sequence is possible (1: possible, 0:
impossible)
        [Seq] = CheckSequence(b, Combined);

        if Seq > 0
            % create 'phasebdd'
            [phasebdd, Combined] = CreatePhaseBDD(phasebdd, b,
SystemBDD, Combined);

            % weight prob. of each DET being used
            [cPMSBDD] = combinePMSBDD(Combined.PhaseConf, phasebdd);
            [UR] = calURpmsbdd(cPMSBDD, BE, Combined.PTD,
Combined.PhaseConf);
            % save weight probabilities
            result(rowN,2) = UR;

            % AC#1: create transition prob. and compute conditional
prob. of CD&R failures
            CDRavailabilityAC1 = cell2mat(Combined.systemAC1(:,2));
            [sTransProb] = createTmatrix(AC(1).StateTransition, deltaT,
Combined.systemAC1(:,1), Combined.PST, CDRavailabilityAC1,
Combined.pilotAC1, AC, AC(1).rate);
            [DETac1, EndStateAC1, statesAC1] = calDET(sTransProb);

            % AC#2: create transition prob. and compute conditional
prob. of CD&R failures
            idx = 1;
            while isempty(Combined.systemAC2{idx,2})
                idx = idx + 1;
            end
            CDRavailabilityAC2 =
cell2mat(Combined.systemAC2(idx:size(Combined.systemAC2,1),2));
            SystemAC2 =
Combined.systemAC2(idx:size(Combined.systemAC2,1),1);
            PST_AC2 = Combined.PST(idx:size(Combined.PST,1),1);
            pilotRateAC2=
Combined.pilotAC2(idx:size(Combined.pilotAC2,1),1);
            [sTransProb] = createTmatrix(AC(2).StateTransition, deltaT,
SystemAC2, PST_AC2, CDRavailabilityAC2, pilotRateAC2, AC, AC(2).rate);
            [DETac2, EndStateAC2, statesAC2] = calDET(sTransProb);
            Combination(rowN,:) =
[CDRavailabilityAC1',CDRavailabilityAC2'];

            % Conditional CD&R failure prob.
            result(rowN,1) = EndStateAC1{1,3}*EndStateAC2{1,3};
            rowN = rowN + 1;
        end
        clear PST_AC2;
    end
```

```
    % exclude cases where system fails during operation
    [result, Combination1] = ExcludeCombi(Combined.systems, Combination,
result);

    % CD&R failure prob.
    TotalFailureProb(1,1) = AC(1).heading - AC(2).heading;
    TotalFailureProb(1,2) = sum(result(:,1).*result(:,2));
end
```

CombineCDRphases.m

```
function [Combined] = CombineCDRphases(pair)
    System_A = pair(1).system;
    System_B = pair(2).system;
    Activation_A = [pair(1).time_horizon; 0];
    Activation_B = [pair(2).time_horizon; 0];
    Pilot_A = pair(1).pilotRate;
    Pilot_B = pair(2).pilotRate;
    tempCombined = cell(1,3);
    p = 1;
    i = 1;
    j = 1;
    while i <= size(Activation_A,1)
        if Activation_A(i) > Activation_B(j)
            tempCombined{p,1} = Activation_A(i);
            tempCombined(p,2) = System_A(i);
            if p > 1
                tempCombined(p,3) = tempCombined(p-1,3);
                tempCombined{p,5} = tempCombined{p-1,5};
            else
                tempCombined{p,3} = 'n/a';
                tempCombined{p,5} = 0;
            end
            tempCombined{p,4} = Pilot_A(i);
            i = i + 1;
        elseif Activation_A(i) < Activation_B(j)
            tempCombined{p,1} = Activation_B(j);
            if p > 1
                tempCombined(p,2) = tempCombined(p-1,2);
                tempCombined{p,4} = tempCombined{p-1,4};
            else
                tempCombined{p,2} = 'n/a';
            end
            tempCombined(p,3) = System_B(j);
            tempCombined{p,5} = Pilot_B(j);
            j = j + 1;
        elseif Activation_A(i) == Activation_B(j)
            if Activation_A(i) > 0
                tempCombined{p,1} = Activation_A(i);
                tempCombined(p,2) = System_A(i);
                tempCombined(p,3) = System_B(j);
                tempCombined{p,4} = Pilot_A(i);
                tempCombined{p,5} = Pilot_B(j);
```

```
            end
            i = i + 1;
            j = j + 1;
        end
        p = p + 1;
    end
    Combined.PST = cell2mat([tempCombined(:,1); 0]);
    Combined.PST = fix(Combined.PST);
    for i = 2 : size(Combined.PST,1)
        Combined.PTD(i-1) = (Combined.PST(i-1)-Combined.PST(i))/60;
    end
    % create cell array of systems for combined phase structure
    Combined.systemAC1 = tempCombined(:,2);
    Combined.systemAC2 = tempCombined(:,3);
    Combined.pilotAC1 = cell2mat(tempCombined(:,4));
    Combined.pilotAC2 = cell2mat(tempCombined(:,5));
    Combined.systems=[Combined.systemAC1;Combined.systemAC2];
    for i = size(Combined.systems,1):-1:1
        if strcmp(Combined.systems{i},'n/a')
            Combined.systems(i)=[];
        end
    end
    Combined.nSystems = size(Combined.systems,1);
    n = 1;
    for i = 1 : size(tempCombined,1)
        % create cell array of phase names for combined phase structure
        if ~strcmp(tempCombined{i,3}, 'n/a')
            Str(1) = string(tempCombined(i,2));
            Str(2) = string(tempCombined(i,3));
            Combined.Phases{i,1} = convertStringsToChars(join(Str,
"/"));
        else
            Combined.Phases(i,1) = tempCombined(i,2);
        end
        if i == 1
            Combined.PhaseConf = Combined.Phases(i,1);
            n = n + 1;
        else
            Combined.PhaseConf(1,n) = {'0'};
            n = n + 1;
            Combined.PhaseConf(1,n) = Combined.Phases(i,1);
            n = n + 1;
        end
    end
end
```

CheckSequence.m

```
function [Seq] = CheckSequence(b, Combined)
    Seq = 1;    % initial value = possible
    idx = 1;    % tracking phases
    while idx < Combined.nSystems
        if strcmp(Combined.systems{idx}, Combined.systems{idx+1})
            if b(idx) < b(idx+1)
```

```
                Seq = 0;
                break;
            end
        end
        idx = idx + 1;
    end
end
```

CreatePhaseBDD.m

```
function [phasebdd, Combined] = CreatePhaseBDD(phasebdd, b, SystemBDD,
Combined)
nSys1 = size(Combined.systemAC1, 1);
nSys2 = size(Combined.systemAC2, 1);
n = 1;
for i = 1 : nSys1
    if ~strcmp(Combined.systemAC1{i,1}, 'n/a')
        Combined.systemAC1{i,2} = str2num(b(n));
        n = n + 1;
    end
end
for i = 1 : nSys2
    if ~strcmp(Combined.systemAC2{i,1}, 'n/a')
        Combined.systemAC2{i,2} = str2num(b(n));
        n = n + 1;
    end
end
if n > Combined.nSystems + 1
    disp('error');
end
for i = 1 : size(Combined.Phases, 1)
    x = Combined.systemAC1{i,1};
    y = Combined.systemAC2{i,1};
    if strcmp({y}, 'n/a')
        idx = find(strcmp({SystemBDD.name}, {x}));
        if Combined.systemAC1{i,2} < 1
            phasebdd(i).bdd = SystemBDD(idx).bdd;
        else
            phasebdd(i).bdd = SystemBDD(idx).dbdd;
        end
    else
        idx0 = find(strcmp({SystemBDD.name}, {x}));
        idx1 = find(strcmp({SystemBDD.name}, {y}));
        sysTOph = {x, '0', y};
        systembdd(1).name = x;
        systembdd(2).name = y;
        if Combined.systemAC1{i,2} < 1
            systembdd(1).bdd = SystemBDD(idx0).bdd;
        else
            systembdd(1).bdd = SystemBDD(idx0).dbdd;
        end
        if Combined.systemAC2{i,2} < 1
            systembdd(2).bdd = SystemBDD(idx1).bdd;
        else
```

```
            systembdd(2).bdd = SystemBDD(idx1).dbdd;
        end
        [phasebdd(i).bdd] = combineBDD(sysTOph, systembdd);
    end
end
```

combineBDD.m

```
function [cBDD] = combineBDD(PhaseConf, Phase)
s = CStack();
len = size(PhaseConf);
cBDD = zeros(1,4);
for p = 1 : (len(2)+1)/2
    CurrPhase.name = PhaseConf{2*p-1};
    index = find(strcmp({Phase.name},CurrPhase.name));
    tempBDD = Phase(index).bdd;
    % assign phase order
    topnode(p) = 1000;
    for i = 1 : size(tempBDD,1)
        if topnode(p) >= tempBDD(i,2)
            topnode(p) = tempBDD(i,2);
        end
    end
    % create cBDD to save combined BDD
    if cBDD(1,1) == 0
        cBDD = tempBDD;
    else
        sz = size(cBDD);
        rowN = sz(1);
        for j = 1 : size(tempBDD,1)
            cBDD(rowN + j, 1) = rowN + j;
            cBDD(rowN + j, 2) = tempBDD(j, 2);
            if tempBDD(j, 3) > 0
                cBDD(rowN + j, 3) = tempBDD(j, 3) + rowN;
            else
                cBDD(rowN + j, 3) = tempBDD(j, 3);
            end
            if tempBDD(j, 4) > 0
                cBDD(rowN + j, 4) = tempBDD(j, 4) + rowN;
            else
                cBDD(rowN + j, 4) = tempBDD(j, 4);
            end
        end
        F = find(cBDD(1:rowN,2)==topnode(p-1));
        G = find(cBDD(rowN+1:rowN+j,2)==topnode(p))+rowN;
        op = PhaseConf{(p-1)*2};
        [cBDD] = operateBDD(cBDD, op, F, G);

        % Extract lines representing combined BDD from cBDD array
        s.empty();
        a = topnode(p-1);
        b = topnode(p);
        if a == b
            if topnode(p-1) > topnode(p)
```

```
                topnode(p) = topnode(p-1);
            end
        elseif a < b
            topnode(p) = topnode(p-1);
        end

        rowNtop = size(cBDD,1);
        while cBDD(rowNtop, 2)~=topnode(p)
            rowNtop = rowNtop - 1;
        end
        push(s, rowNtop);    % row number for top node of BDD
        [cBDD] = extract(cBDD, s);

        % reduce rows of BDD table
        [cBDD] = reduceTable(cBDD);
    end
end
```

operateBDD.m

```
function [BDD] = operateBDD(BDD, op, F, G)
l = size(BDD,1) + 1;
% Determine which one has priority
% F and G are row numbers, a and b are variable numbers meaning order
a.v = BDD(F, 2);    % variable
b.v = BDD(G, 2);
BDD(l, 1) = l;
BDD(l, 2) = BDD(F, 2);
if a.v > b.v
    temp1 = a;
    a = b;
    b = temp1;
    temp2 = F;
    F = G;
    G = temp2;
    BDD(l, 2) = BDD(F, 2);
end

% conventional BDD operation
if op == '0'    % 'AND' gate
    if a.v ~= b.v
        % compute 'then' value
        if BDD(F, 3) == -1
            BDD(l, 3) = BDD(G, 1);
        elseif BDD(F, 3) == 0
            BDD(l, 3) = 0;
        else
            F1 = BDD(F, 3);
            G1 = G;
            BDD(l, 3) = size(BDD,1) + 1;
            [BDD] = operateBDD(BDD, op, F1, G1);
        end
        % compute 'else' value
        if BDD(F, 4) == -1
```

```
            BDD(l, 4) = BDD(G, 1);
        elseif BDD(F, 4) == 0
            BDD(l, 4) = 0;
        else
            F2 = BDD(F, 4);
            G2 = G;
            BDD(l, 4) = size(BDD,1) + 1;
            [BDD] = operateBDD(BDD, op, F2, G2);
        end
    elseif a.v == b.v
        % compute 'then' value
        if BDD(F, 3) == -1
            BDD(l, 3) = BDD(G, 3);
        elseif BDD(F, 3) == 0
            BDD(l, 3) = 0;
        elseif BDD(G, 3) == -1
            BDD(l, 3) = BDD(F, 3);
        elseif BDD(G, 3) == 0
            BDD(l, 3) = 0;
        else
            F1 = BDD(F, 3);
            G1 = BDD(G, 3);
            BDD(l, 3) = size(BDD,1) + 1;
            [BDD] = operateBDD(BDD, op, F1, G1);
        end
        % compute 'else' value
        if BDD(F, 4) == -1
            BDD(l, 4) = BDD(G, 4);
        elseif BDD(F, 4) == 0
            BDD(l, 4) = 0;
        elseif BDD(G, 4) == -1
            BDD(l, 4) = BDD(F, 4);
        elseif BDD(G, 4) == 0
            BDD(l, 4) = 0;
        else
            F2 = BDD(F, 4);
            G2 = BDD(G, 4);
            BDD(l, 4) = size(BDD,1) + 1;
            [BDD] = operateBDD(BDD, op, F2, G2);
        end
    end
else            % 'OR' gate
    if a.v ~= b.v
        % compute 'then' value
        if BDD(F, 3) == -1
            BDD(l, 3) = -1;
        elseif BDD(F, 3) == 0
            BDD(l, 3) = BDD(G, 1);
        else
            F1 = BDD(F, 3);
            G1 = G;
            BDD(l, 3) = size(BDD,1) + 1;
            [BDD] = operateBDD(BDD, op, F1, G1);
        end
```

```
        % compute 'else' value
        if BDD(F, 4) == -1
           BDD(l, 4) = -1;
        elseif BDD(F, 4) == 0
            BDD(l, 4) = BDD(G, 1);
        else
            F2 = BDD(F, 4);
            G2 = G;
            BDD(l, 4) = size(BDD,1) + 1;
            [BDD] = operateBDD(BDD, op, F2, G2);
        end
    elseif a.v == b.v
        % compute 'then' value
        if BDD(F, 3) == -1
            BDD(l, 3) = -1;
        elseif BDD(F, 3) == 0
            BDD(l, 3) = BDD(G, 3);
        elseif BDD(G, 3) == -1
            BDD(l, 3) = -1;
        elseif BDD(G, 3) == 0
            BDD(l, 3) = BDD(F, 3);
        else
            F1 = BDD(F, 3);
            G1 = BDD(G, 3);
            BDD(l, 3) = size(BDD,1) + 1;
            [BDD] = operateBDD(BDD, op, F1, G1);
        end
        % compute 'else' value
        if BDD(F, 4) == -1
            BDD(l, 4) = -1;
        elseif BDD(F, 4) == 0
            BDD(l, 4) = BDD(G, 4);
        elseif BDD(G, 4) == -1
            BDD(l, 4) = -1;
        elseif BDD(G, 4) == 0
            BDD(l, 4) = BDD(F, 4);
        else
            F2 = BDD(F, 4);
            G2 = BDD(G, 4);
            BDD(l, 4) = size(BDD,1) + 1;
            [BDD] = operateBDD(BDD, op, F2, G2);
        end
    end
end
```

ExcludeCombi.m

```
function [result, cases] = ExcludeCombi(systems, cases, result)
n = size(systems,1);
for i = 1 : n-1
    if strcmp(systems{i}, systems{i+1})
        m = size(cases, 1);
        j = 1;
        while j <= m
```

```
            if cases(j, i) > cases(j, i+1)
                a = cases(j, :);
                row = 0;
                l = i+2;
                k = j+1;
                while k <= m
                    if l <= n
                        for p = l : n
                            if cases(j,p) ~= cases(k,p)
                                break;
                            end
                        end
                        if p == n
                            break;
                        end
                        k = k + 1;
                    else
                        if cases(j,1:i) == cases(k,1:i)
                            break;
                        else
                            k = k + 1;
                        end
                    end
                end
                for l = 2 : size(result,2)  %2%
                    result(k,l) = result(k,l) + result(j,l);
                end
                result(j,:) = [];
                cases(j,:) = [];
                m = size(cases, 1);
            else
                j = j + 1;
            end
        end
    end
end
```

CQueue.m

```
classdef CQueue < handle
% CQueue define a queue data strcuture
% Copyright: zhang@zhiqiang.org, 2010.
% url: http://zhiqiang.org/blog/it/matlab-data-structures.html

    properties (Access = private)
        buffer      % a cell, to maintain the data
        beg         % the start position of the queue
        rear        % the end position of the queue
                    % the actually data is buffer(beg:rear-1)
    end

    properties (Access = public)
        capacity    % ???2
    end
```

```
    methods
        function obj = CQueue(c) % ?
            if nargin >= 1 && iscell(c)
                obj.buffer = [c(:); cell(numel(c), 1)];
                obj.beg = 1;
                obj.rear = numel(c) + 1;
                obj.capacity = 2*numel(c);
            elseif nargin >= 1
                obj.buffer = cell(100, 1);
                obj.buffer{1} = c;
                obj.beg = 1;
                obj.rear = 2;
                obj.capacity = 100;
            else
                obj.buffer = cell(1000, 1);
                obj.capacity = 1000;
                obj.beg = 1;
                obj.rear = 1;
            end
        end

        function s = size(obj) % ?
            if obj.rear >= obj.beg
                s = obj.rear - obj.beg;
            else
                s = obj.rear - obj.beg + obj.capacity;
            end
        end

        function b = isempty(obj) % return true when the queue is empty
            b = ~logical(obj.size());
        end

        function s = empty(obj) % clear all the data in the queue
            s = obj.size();
            obj.beg = 1;
            obj.rear = 1;
        end

        function push(obj, el) % ????
            if obj.size >= obj.capacity - 1
                sz = obj.size();
                if obj.rear >= obj.front
                    obj.buffer(1:sz) = obj.buffer(obj.beg:obj.rear-1);
                else
                    obj.buffer(1:sz) = obj.buffer([obj.beg:obj.capacity
1:obj.rear-1]);
                end
                obj.buffer(sz+1:obj.capacity*2) = cell(obj.capacity*2-
sz, 1);
                obj.capacity = numel(obj.buffer);
                obj.beg = 1;
```

```
                obj.rear = sz+1;
            end
            obj.buffer{obj.rear} = el;
            obj.rear = mod(obj.rear, obj.capacity) + 1;
        end

        function el = front(obj) % ??
            if obj.rear ~= obj.beg
                el = obj.buffer{obj.beg};
            else
                el = [];
                warning('CQueue:NO_DATA', 'try to get data from an
empty queue');
            end
        end

        function el = back(obj) % ???

           if obj.rear == obj.beg
               el = [];
               warning('CQueue:NO_DATA', 'try to get data from an empty
queue');
            else
                if obj.rear == 1
                    el = obj.buffer{obj.capacity};
                else
                    el = obj.buffer{obj.rear - 1};
                end
            end

        end

        function el = pop(obj) % ?
            if obj.rear == obj.beg
                error('CQueue:NO_Data', 'Trying to pop an empty queue');
            else
                el = obj.buffer{obj.beg};
                obj.beg = obj.beg + 1;
                if obj.beg > obj.capacity, obj.beg = 1; end
            end
        end

        function remove(obj) % ?
            obj.beg = 1;
            obj.rear = 1;
        end

        function display(obj) % ?
            if obj.size()
                if obj.beg <= obj.rear
                    for i = obj.beg : obj.rear-1
```

```
                        disp([num2str(i - obj.beg + 1) '-th element of
the stack:']);
                        disp(obj.buffer{i});
                    end
                else
                    for i = obj.beg : obj.capacity
                        disp([num2str(i - obj.beg + 1) '-th element of
the stack:']);
                        disp(obj.buffer{i});
                    end
                    for i = 1 : obj.rear-1
                        disp([num2str(i + obj.capacity - obj.beg + 1)
'-th element of the stack:']);
                        disp(obj.buffer{i});
                    end
                end
            else
                disp('The queue is empty');
            end
        end

        function c = content(obj) % ??
            if obj.rear >= obj.beg
                c = obj.buffer(obj.beg:obj.rear-1);
            else
                c = obj.buffer([obj.beg:obj.capacity 1:obj.rear-1]);
            end
        end
    end
end
```

CStack.m

```
classdef CStack < handle
% CStack define a stack data strcuture
% Copyright: zhang@zhiqiang.org, 2010.
% url: http://zhiqiang.org/blog/it/matlab-data-structures.html

    properties (Access = private)
        buffer      % ?cell??
        cur         % ???, or the length of the stack
        capacity    % ???2
    end

    methods
        function obj = CStack(c)
            if nargin >= 1 && iscell(c)
                obj.buffer = c(:);
                obj.cur = numel(c);
                obj.capacity = obj.cur;
            elseif nargin >= 1
                obj.buffer = cell(100, 1);
                obj.cur = 1;
                obj.capacity =100;
```

```
            obj.buffer{1} = c;
        else
            obj.buffer = cell(100, 1);
            obj.capacity = 100;
            obj.cur = 0;
        end
    end

    function s = size(obj)
        s = obj.cur;
    end

    function remove(obj)
        obj.cur = 0;
    end

    function b = empty(obj)
        b = obj.cur;
        obj.cur = 0;
    end

    function b = isempty(obj)
        b = ~logical(obj.cur);
    end

    function push(obj, el)
        if obj.cur >= obj.capacity
            obj.buffer(obj.capacity+1:2*obj.capacity) =
cell(obj.capacity, 1);
            obj.capacity = 2*obj.capacity;
        end
        obj.cur = obj.cur + 1;
        obj.buffer{obj.cur} = el;
    end

    function el = top(obj)
        if obj.cur == 0
            el = [];
            warning('CStack:No_Data', 'trying to get top element of
an emtpy stack');
        else
            el = obj.buffer{obj.cur};
        end
    end

    function el = pop(obj)
        if obj.cur == 0
            el = [];
            warning('CStack:No_Data', 'trying to pop element of an
emtpy stack');
        else
            el = obj.buffer{obj.cur};
```

```
                obj.cur = obj.cur - 1;
            end
        end

        function display(obj)
            if obj.cur
                for i = 1:obj.cur
                    disp([num2str(i) '-th element of the stack:']);
                    disp(obj.buffer{i});
                end
            else
                disp('The stack is empty');
            end
        end

        function c = content(obj)
            c = obj.buffer(1:obj.cur);
        end
    end
end
```

REFERENCES

Acosta, C. and N. Siu. (1993). *Dynamic event trees in accident sequence analysis: application to steam generator tube rupture*. Reliability Engineering and System Safety, 41, pp. 135-154

Aldemir, T. (2013). *A survey of dynamic methodologies for probabilistic safety assessment of nuclear power plants*. Annals of Nuclear Energy, 52, pp. 113-124

Andrews, J. and S. Dunnett. (2000). *Event-tree analysis using binary decision diagrams*. IEEE Transactions on Reliability, 49, pp. 230-238

Andrews, J., J. Welch, and H. Erzberger. (2005). *Safety analysis for advanced separation concepts*. Proceedings of 6th USA/Europe ATM R&D Seminar, Baltimore, MD.

Arino, T., K. Carpenter, S. Chabert, H. Hutchinson, T. Miquel, B. Raynaud, K. Rigotti, and E. Vallauri. (2002). *ACAS analysis programme ACASA project; Work Package 1: Final report on studies on the safety of ACAS II in Europe*. ACASA/WP-1.8/210D.

Belle, A., J. Shortle, and A. Yousefi (2012). *Estimation of potential conflict rates as a function of sector loading*. Proceedings of International Conference on Research in Air Transportation (ICRAT), Berkeley, CA.

Bilimoria, K., B. Sridhar, G. Chatterji, K. Sheth, and S. Grabbe (2000). *FACET: Future ATM concepts evaluation tool*. Proceedings of 3rd USA/Europe ATM R&D Seminar, Napoli, Italy.

Blom, H.A.P., G.J. Bakker, P.J.G. Blanker, J. Daams, M.H.C. Everdij and M.B. Klompstra (1999). *Accident risk assessment for advanced ATM*. NLR-TP-99015

Blom, H.A.P., M.B. Klompstra, and G.J. Bakker (2003). *Accident risk assessment of simultaneous converging instrument approaches*. NLR-TP-2003-557

Blom, H.A.P., G.J. Bakker, B.K. Obbink, and M.B. Klompstra (2006). *Free flight safety risk modeling and simulation*. NLR-TP-2006-290

Blum, D.M., D. Thipphavong, T.L. Rentas, Y. He, X. Wang, and E. Pate-Cornell (2010). *Safety analysis of the advanced airspace concept using Monte Carlo simulation*. AIAA Guidance, Navigation, and Control Conference, Toronto, Canada.

Borener, S., S. Trajkov, and P. Balakrishna. (2012). *Design and development of an Integrated Safety Assessment Model for NextGen*. International Annual Conference of the American Society for Engineering Management.

Cojazzi, G. (1996). *The DYLAM approach for the dynamic reliability analysis of systems*. Reliability Engineering and System Safety, 52, pp. 279-296

Devooght, J. and C. Smidts. (1996). *Probabilistic dynamics as a tool for dynamic PSA.* Reliability Engineering and System Safety, 52, pp. 185-196

Dugan, J.B., S.J. Bavuso and M.A. Boyd. (1990). *Fault trees and sequence dependencies.* Annual Proceedings on Reliability and Maintainability Symposium, Los Angeles, CA, USA.

Endoh, S. (1982). *Aircraft collision models.* Flight Transportation Laboratory, Massachusetts Institute of Technology, R82-2.

Erzberger, H. (2001). *The automated airspace concept*. Proceedings of the 4th USA/Europe ATM R&D Seminar, Santa Fe, NM.

Erzberger, H. (2004). *Transforming the NAS: The Next Generation Air Traffic Control System*. NASA Ames Research Center, NASA/TP–2004-212801.

Erzberger, H., T.A. Lauderdale, and Y-C. Chu (2012). *Automated conflict resolution, arrival management, and weather avoidance for air traffic management*. Journal of Airspace Engineering, 226, pp. 930-949.

Esary, J.D., and H. Ziehms. (1975). *Reliability analysis of phased missions.* Technical Report (NPS55Ey75021), Naval Postgraduate School.

FAA. (2011). *Introduction to TCAS II* (Version 7.1)

FAA. (2018). FAA aerospace forecasts fiscal years 2018 – 2038

FAA. (2018). https://www.faa.gov/nextgen/

FAA. (2019). *Safety management system manual*. Air

Farley, T., M. Kupfer, and H. Erzberger. (2007). *Automated conflict resolution: A simulation evaluation under high demand including merging arrivals*.

Proceedings of AIAA Aviation Technology, Integration and Operations Conference (ATIO), Belfast, Northern Ireland.

Fasano, G., D. Accado, A. Moccia, and D. Moroney. (2016). *Sense and avoid for unmanned aircraft systems*. IEEE Aerospace and Electronic Systems Magazine, 31(11), pp. 82-110.

Ferreira, R.B., D.M. Baum, E.C.P. Neto, M.R. Martins, J.R. Almeida Jr., P.S.Cugnasca, and J.B. Camargo Jr. (2018). *A risk analysis of unmanned aircraft systems (UAS) integration into non-segregate airspace.* 2018 International Conference on Unmanned Aircraft Systems (ICUAS), Dallas, TX, pp. 42-51.

Gulati, R. and J.B. Dugan. (1997). *A modular approach for analyzing static and dynamic fault trees,*" Annual Reliability and Maintainability Symposium, Philadelphia, PA, USA.

Hofer, E., M. Kloos, B. Krzykacz-Hausmann, J. Peschke, and M. Sonnenkalb. (2004). *Dynamic Event Trees for Probabilistic Safety Analysis*. GRS, Garsching, Germany.

Jenie, Y.I., E. van Kampen, J. Ellerbroek, and J.M. Hoekstra (2018), *Safety assessment of a UAV CD&R system in high density airspace using Monte Carlo simulations*. IEEE Transactions on Intelligent Transportation Systems, 19(8), pp. 2686-2695.

Kim, K. and K.S. Park. (1994). *Phased-mission system reliability under Markov environment*. IEEE Transactions on Reliability, 43(2), pp. 301-309.

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

Kuchar, J., J. Andrews, A. Drumm, T. Hall, V. Heinz, S. Thompson, and J. Welch. (2004). *A Safety analysis process for the traffic alert and collision avoidance system (TCAS) and see-and-avoid systems on remotely piloted vehicles*. AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit. Chicago, IL.

Kuchar, J.K. (2005). *Safety analysis methodology for unmanned aerial vehicles (UAV) collision avoidance systems*. Proceedings of 6th USA/Europe ATM R&D Seminar, Baltimore, MD.

Lacher, A.R., D.R. Maroney, and A.D. Zeitlin. (2007). *Unmanned aircraft collision avoidance technology assessment and evaluation methods*. Proceedings of 7th USA/Europe ATM R&D Seminar, Barcelona, Spain.

Lin X., N.L. Fulton, and M. Westcott. (2009). *Target level of safety measures in air transportation - Review, validation and recommendations*. Proceedings of the IASTED International Congress on Advances in Management Science and Risk Assessment (AMSRA 2009), Beijing, China. pp. 222–662

Muñoz, C., A. Narkawicz, G. Hagen, J. Upchurch, A. Dutle, and M. Consiglio. (2015). *DAIDALUS: Detect and avoid alerting logic for unmanned systems*. 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), Prague, Czech Republic.

Noh, S., and J. Shortle. (2015). *Sensitivity analysis of event sequence diagrams for aircraft accident scenarios*. 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), Prague, Czech Republic.

Podofillini, L. and V.N. Dang. (2012). *Conventional and dynamic safety analysis: Comparison on a chemical batch reactor.* Reliability Engineering and System Safety, 106, pp. 146-159

Rausand, M. and A. Hoyland. (2004). *System reliability theory; models and statistical methods*. Wiley, New York.

Rauzy, A. (1993). *New alorithms for fault trees analysis.* Reliability Engineering and System Safety, 40, pp. 203-211

Rauzy, A. (2008). *Binary decision diagrams for reliability studies*. In Handbook of Performability Engineering, K. Misra (ed.), Springer

Reich, P.G. (1966). *Analysis of long-range air traffic systems: Separation standards–I.* Journal of Navigation, 19(1), pp. 88-98

Shortle, J., Y. Xie, C.H. Chen, and G.L. Donohue (2004). *Simulating collision probabilities of landing airplanes at nontowered airports*. Simulation, 80(1), pp. 21-31.

Shortle, J., L. Sherry, A. Yousefi, and R. Xie. (2012). *Safety and sensitivity analysis of the advanced airspace concept for NextGen*. Proceedings of the Integrated Communication, Navigation, and Surveillance Conference (ICNS), Herndon, VA.

Shortle, J., S. Noh, and L. Sherry. (2017). *Collision risk analysis for alternate airspace architecture*. 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), St. Petersburg, FL.

Sinnamon, R. and J. Andrews. (1997). *Improved accuracy in quantitative fault tree analysis*. Quality and Reliability Engineering International, 13, pp. 285-292

Sinnamon, R. and J. Andrews. (1997). *Improved efficiency in qualitative fault tree analysis*. Quality and Reliability Engineering International, 13, pp. 293-298

Siu, N. (1994). *Risk assessment for dynamic systems: An overview*. Reliability Engineering and System Safety, 43, pp. 43-73

Sweet, D.N., V. Manikonda, J.S. Aronson, K. Roth, and M. Blake (2002). *Fast-time simulation system for analysis of advanced air transportation concepts*. Proceedings of AIAA Modeling and Simulation Technologies Conference and Exhibit, Monterey, CA

Vesely, W.E., F.F. Goldberg, N.H. Robers, and D.F. Haasl (1981). *Fault tree handbook*. U.S. Nuclear Regulatory Commision/NUREG-0492

Walden, D.D., G.J. Roedler, K. Gorsberg, R.D. Hamelin, and T.M. Shortell. (2015). *System engineering handbook: a guide for system life cycle processes and activities*. Fourth edition, John Wiley & Sons, New Jersey.

Wieland, F. (2016). *The drones are coming: Is the National Airspace System prepared?* Journal of Air Traffic Control, 58(2), pp. 24-30

Wieland, F. and Y. Ryabov (2017). *Future architectures for autonomous National Airspace System control: Concept of operation and evaluation*. 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), St. Petersburg, FL.

Wing, D.J. and W.B. Cotton (2011). *Autonomous flight rules: A concept for self-separation in U.S. domestic airspace*. NASA/TP–2011-217174

Xing, L. and J.B. Dugan. (2002). *Analysis of generalized phased-mission system reliability, performance, and sensitivity*. IEEE Transactions on Reliability, 51(2), pp. 199-211

Xing, L. and S. Amari. (2008). *Fault tree analysis*. In Handbook of Performability Engineering, K. Misra (ed.), Springer

Xing, L. and S. Amari. (2008). *Reliability of phased-mission systems*. In Handbook of Performability Engineering, K. Misra (ed.), Springer

Xing, L. and G. Levitin. (2013). *BDD-based reliability evaluation of phased-mission systems with internal/external common-cause failures*. Reliability Engineering and System Safety, 112, pp. 145-153.

Yu, X. and Y. Zhang. (2015). *Sense and avoid technologies with applications to unmanned aircraft systems: Review and prospects.* Progress in Aerospace Sciences, 74, pp. 152-166.

Zang, X., H. Sun, and K. Trivedi. (1999). *A BDD-based algorithm for reliability analysis of phased-mission systems*. IEEE Transactions on Reliability, 48(1), pp. 50-60.

Zhang, Y., J. Shortle, and L. Sherry. (2015). *Methodology for collision risk assessment of an airspace flow corridor concept*. Reliability Engineering and System Safety, 142, pp. 444-455.

Zhang, X., Y. Liu, Y. Zhang, X. Guan, D. Delahaye, and L. Tang. (2018). *Safety assessment and risk estimation for unmanned aerial vehicles operating in national airspace system*. Journal of Advanced Transportation. https://doi.org/10.1155/2018/4731585

BIOGRAPHY

Seungwon Noh received Bachelor of Science in Air Transportation from Korea Aerospace University, Korea, in 2005. He received Master of City Planning in Environmental Planning from Seoul National University, Korea, in 2007 and Master of Science in Operations Research from George Mason University, Virginia, in 2013.