Extending a Formal Specification & Requirements Language:

A Case Study

by

Danny Cho-Liang Lai

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Computer Science and Engineering

at the Massachusetts Institute of Technology

May 23, 2001

Copyright 2001 Danny Cho-Liang Lai. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author

Department of Electrical Engineering and Computer Science

May 23, 2001

Certified by

Nancy G. Leveson

Thesis Supervisor

Accepted by

Arthur C. Smith

Chairman, Department Committee on Graduate Theses
Abstract

SpecTRM, a software system modeling methodology based on specifications, was developed in earlier work by Nancy Leveson. SpecTRM hierarchically organizes design decisions based on intent, and provides for tracing through these intents in various levels of complexity. It is especially powerful for complex systems that require many levels of communication in their development. A proof of concept of SpecTRM was demonstrated on the Dallas/Ft Worth TRACON system, and since then has been used to model various other air traffic control system software. This project focused on a large-scale air traffic control system, Raytheon’s Standard Terminal Automation Replacement System (STARS). The SpecTRM methodology was used to model the STARS Conflict Alert/Mode-C Intruder (CA/MCI) component, to serve as a proof of concept for a larger system, and to identify any deficiencies in the language syntax that may be encountered during the modeling process. The current SpecTRM syntax performed remarkably well, but need to be extended to describe sequential procedures and multiple display modes. These additions would facilitate a complete specification of the STARS CA/MCI component.
Acknowledgements

I would like to thank the members of the Aero/Astro Software Engineering Research Laboratory for their helpful advice while I learned the SpecTRM methodology. William Melendez and Sean Sutherland also worked on the Raytheon STARS project with me, and were great resources for sanity checks and for insight on modeling difficult components and logic. I am also grateful for the timely and enthusiastic support from the sponsors at Raytheon Corporation, particularly Jeffrey Levine, Brian Donnelly, and George Weigler. Jeffrey, Brian, and George arranged workshop sessions to see and learn the STARS software, and were always prompt with answers to my questions and requests for more information. In addition, my apartment-mates have been very supportive and encouraging. I am especially grateful to Emily Chang, whose worth ethic and discipline I admired and gave me inspiration to see this project through to completion. Finally, I would like to acknowledge my thesis advisor, Nancy Leveson. Nancy trained me on the principles of system and software safety, and was always inspiring and encouraging. The invigorating discussions with Nancy were the most enjoyable parts of this project.
Table Of Contents

Introduction .................................................................................................................. 6

1.1 Safety Engineering vs. Software Engineering ....................................................... 7

1.2 Example: Air Traffic Control .............................................................................. 7

1.3 SpecTRM ............................................................................................................. 9

1.4 Project Purpose ................................................................................................. 10

2 Related Work ......................................................................................................... 11

3 Methodology ......................................................................................................... 12

3.1 Software Hazard Analysis ................................................................................ 12

3.2 Intent Specification ........................................................................................... 12

3.2.1 System Purpose and Properties .................................................................. 12

3.2.2 System Design Principles ........................................................................... 13

3.2.3 Blackbox Behavior ...................................................................................... 14

3.2.4 Physical Representation .............................................................................. 14

3.2.5 Implementation ........................................................................................... 15

4 Raytheon Case Study ............................................................................................ 16

4.1 STARS Overview ............................................................................................ 16

4.2 STARS Function Selection .............................................................................. 17

4.2.1 Conflict Alert ................................................................................................... 17

4.2.2 Minimum Safe Altitude Warning .................................................................. 18

4.2.3 Sector Handoff .............................................................................................. 18

4.3 Modeling Procedure & Results ......................................................................... 18
INTRODUCTION

Software automation is pervasive in society. In the interests of efficiency, precision, and increased productivity, computers are being relied upon to perform tasks that are simple and repetitive. As demands on human cognition have increased, computers are taking more active roles in productivity. Instead of assembly line processing, they are now knowledge databases, giving advice to humans in fields varying from medical diagnosis to air traffic control. With increased software automation and reliance on software, software complexity has grown tremendously.

One side effect of this rapid growth is an impact on system safety. Today, computers control most large-scale systems. In these systems, computers are present in safety-critical loops, in varying degrees. They may serve passive functions, such as providing information or advice to a controller, or interpreting data first before displaying it to the controller, deferring to the controller to make the final decisions. They may also issue commands directly, with or without the presence of a human operator to monitor actions. Where the computer issues commands directly, the impact of design flaws is obvious [1]. In systems where the role of the computer is only to generate data and issue alerts and defer to a human operator to issue direct commands, the impact of software design inadequacies may be more subtle. For example, information from a computer or technological device is more readily believed than conflicting information that is directly observed, especially when the human operator has no independent way to verify the information [1].
1.1 Safety Engineering vs. Software Engineering

The sudden widespread use of computers in safety-critical systems such as air traffic control has created problems for software and system engineers. Proven system safety engineering techniques do not include software, and because of the unique characteristics of system software, may not be easily adapted to software [1].

There is a communication barrier between the different types of engineers working on a project. System and software engineers have few common models or tools. They have different vocabularies, and use similar terminology to describe different concepts. System engineers may treat the computer as a black box. Software engineers, on the other hand, often treat the computer as a stimulus-response system; a stimulus is received and a response is provided [1]. Their field of focus does not extend beyond the computer, and they may construct the software with little consideration of hazards at the system level.

1.2 Example: Air Traffic Control

Air Traffic Control (ATC) is an excellent example of a large-scale system whose technology has grown so quickly that safety standards have lagged behind. According to the U.S. Department of Transportation’s Bureau of Transportation Statistics, commercial flights increased nearly seven percent in 2000, and the number of passengers carried on those flights increased by over four percent [2]. In the U.S., there are over 50,000 aircraft operations per day [3]. Increasing demand for air travel has cluttered up the national airspace, and placed greater workload on human air traffic controllers. This increased human workload is not sustainable, and as a result, flight delays increased 36 percent in 1999, and fewer than 75% of America’s flights arrived on time in 2000 [4]. The demand
for moving people and freight are increasing, and as a result, more automation is being introduced into the existing ATC systems to control workload for the human air traffic controllers.

However, automation does not necessarily increase air traffic safety. It is difficult to ensure that software will not do something undesired or dangerous. In addition, a balance must be struck between minimal automation and total dependence on the ATC software. ATC is a cognitively demanding job, requiring controllers to be 100% aware of the airspace they monitor, and alert enough to make quick logical decisions when an anomaly is detected. It is clear that some ATC functions need to be automated, so that controller workload per aircraft can be reduced to allow greater system capacity. However, overzealous automation may reduce the workload of the controller so much that he would essentially become a passive monitor and is unable to effectively monitor or intervene in case of a problem. This situation would result in an over-dependence on a complex software system, perhaps leading to reduced safety. Therefore, humans will continue to be in the control loop in some capacity. Control will be shared between human and computer in the immediate future. This interaction is where most safety-related problems in ATC arise.

As a result, software systems for air traffic control must be tested vigorously for safety. Safety criteria and tests must also be closely documented and easily accessed. Air traffic control is a unique field in that systems cannot be upgraded synchronously; it is not possible for all air traffic control centers to instantaneously reset and restart with new software. Instead, version upgrades are accomplished in phases, which places huge emphasis on backward compatibility and requirements traceability. Behavior changes as
the software system evolves, and these changes, whether based on requirements changes, design changes, or implementation changes, must be tracked and readily accessible to explain discrepancies.

1.3 SpecTRM

The Specification Toolkit and Requirements Methodology (SpecTRM) was developed at the MIT Software Engineering Research lab specifically to model complex software systems to identify potential hazards and to trace them to design decisions or intent [5]. SpecTRM emphasizes tracing not only requirements but also design rationale throughout the system design and documentation, building required system properties into the design from the beginning and facilitating finding errors early in development so that they can be fixed with the lowest cost and impact on system design.

SpecTRM is based on the concept of intent specifications, designed to support human problem solving by presenting consistent, relevant information that can be easily processed by developers and users at all levels. Its goal is to provide bridges between diverse groups of system designers and engineers, and to ease communication for coordinated design. It is a workbench to allow teams of engineers to use common models and analysis tools [5].

As a methodology, SpecTRM is an information presentation tool, designed to present several views of the software system, each linked through intent. A high-level software requirement is linked to a design decision that it motivates, which in turn is linked to a specific implementation scheme. These components all reside on different views. The idea is that certain views are more appropriate for certain people. As an example, consider ATC system software. Customers and regulatory officials would be more
interested in a high-level requirements view of the software (since they presumably had input). On the other hand, a project manager would be more interested in viewing the design principles or blackbox design, and a programmer would only be interested in the documentation and code. Each of these levels is a completely different view; their focus scopes are different, and they use different terminology. However, components of each view are linked to those of other views, as some explain the intent of other levels.

As software, SpecTRM is a modeling tool used to generate a SpecTRM model. The model is a single document, with several levels, each representing a different view of the modeled software system. The same document can be viewed by personnel at all levels of involvement with the system, from the developer all the way to the user. Changes made to the document will be seen by everyone, so synchronicity of updates is preserved.

1.4 Project Purpose

The purpose of this project is to evaluate the SpecTRM modeling language by using it in a case study. SpecTRM will be used to model components in Raytheon’s Standard Terminal Automation Replacement System (STARS), a complex air traffic control software system developed as part of a general air traffic control software overhaul recommended by the Federal Aviation Administration (FAA). As mentioned previously, intent specifications and traceability is especially important in the development and refinement of generations of air traffic control software, making STARS an ideal case study subject. Because STARS is large and complex, this project focuses on a single component, the conflict detection module. This project will demonstrate SpecTRM as a proof of concept for the STARS conflict detection module, and will identify strengths and weaknesses in the SpecTRM language during the modeling process.
2 RELATED WORK

The use of SpecTRM to model complex ATC software systems is not new. The methodology was created and is being improved by applying it to industrial projects. The first formal modeling language, the Requirements State Machine Language (RSML), was developed while writing the official FAA specification for TCAS II [6]. Lessons learned resulted in both extending the methodology to other parts of the specification and requirements analysis process through the use of intent specifications, and to improved formal specification languages [7].

Additional experimentation was performed on flight management systems [8], a research version of the center TRACON Automation System (CTAS) [9], a re-specification of TCAS II using intent specifications and a new formal specification language SpecTRM-RL [10], and extensions to assist in human-centered design using a conflict detection tool being developed by Eurocontrol as the test bed. Throughout, the research approach has involved evaluating parts of the proposed methodology on real systems, generation of hypotheses about how it could be improved, and the evaluation of these hypotheses on additional industrial-scale systems.
3 METHODOLOGY

3.1 Software Hazard Analysis

This methodology used in this project is based on the theoretical foundation created by Prof. Nancy Leveson described in her book *Safeware*. The *Safeware* methodology is based on system safety techniques that are extended to deal with software and human error. Identification, classification, and evaluation of hazards are done using modeling and analysis of these software systems [1].

3.2 Intent Specification

An intent specification differs from a standard specification primarily in its structure. The hierarchical abstraction is based on intent rather than description. The focus is to explain “why?” rather than “how?”. Because each level is mapped to the appropriate parts of the intent levels both above and below it, traceability of design rationale and design decisions is provided from high-level system requirements and constraints down to code, and vice versa. The intent information represents the design rationale upon which the specification is based; thus, design rationale is integrated directly into the specification. Each level also contains information about underlying assumptions important in operational safety analyses.

3.2.1 System Purpose and Properties

The highest level of an intent specification assists system engineers in their reasoning about system-level properties such as goals, design constraints, assumptions, limitations, and design evaluation criteria [5]. In the early stages of a project, goals are more
visionary than functional, stated in very general terms. At the system purpose level, goals are refined into testable and achievable high-level requirements.

Design constraints are restrictions on how the system can achieve its purpose [5]. These constraints may stem from resource limitations, the environment in which the system will be used, safety and other quality goals, government standards, etc. Because of the need to evaluate and clarify tradeoffs among alternative design decisions during the system’s development, design constraints are separated from system functional goals and requirements to frame the main issues to consider when making these decisions.

Assumptions are specified to explain a decision or to record information on which the decision is based [5]. These assumptions are often used in making lower level design decisions. Linking the assumptions to all levels of specification will assist in performing system maintenance activities.

System limitations and boundaries describe fundamental limitations in the system design, including hazards or hazard causal factors that cannot be completely eliminated or controlled [5]. They may be related to problems encountered or tradeoffs made during the system design process (see next section).

Evaluation criteria and priorities are used to measure whether or not the goals have been reached. They are used to resolve conflicts among goals and design constraints and to guide design choices at lower levels.

3.2.2 System Design Principles

The second level allows engineers to reason about the system in terms of the physical principles and laws upon which the design is based [5]. These are the basic system design and engineering principles needed to achieve the behavior specified in the top
level. Each principle should be linked to one or more higher level requirements, constraints, assumptions, or limitations. They may also reflect tradeoffs between higher-level goals and constraints.

3.2.3 Blackbox Behavior

The blackbox behavior level enhances reasoning about the logical design of the system as a whole and the interactions between the components [5]. This level most closely resembles software and interface requirements documents written by program managers and software engineers. It specifies the system components (including human components) and their interfaces. Behavioral descriptions at this level are purely blackbox. They describe the externally visible behavior using only the inputs and outputs of each component and their relationships. That is, internal design and implementation details that do not affect externally visible behavior is omitted. The goal is to specify the important system functions assigned to the components without overly constraining the implementers (e.g. software designers).

3.2.4 Physical Representation

The physical representation level is the first in which information about the physical implementation of the components is included. Most of the information contained in this level is linked to the Design Representation level. It typically contains pseudocode design that is unoptimized for readability, human-computer interface design specifications, and verification requirements for the design specified on this level [5]. In software development cycles, this level most closely resembles the prototype stage. However, instead of being discarded (which is what happens in most software
development cycles), the prototype is maintained as a conceptual link between the optimized code (described below) and the software design stage. The case study will not focus on the physical representation level.

3.2.5 Implementation

The Implementation Level is the lowest specification level, and usually includes a description of the physical implementation of the levels above, including software, hardware assembly instructions, etc. In most instances, this is the documented software code. The case study will not focus on the Physical representation or Implementation levels.
4 RAYTHEON CASE STUDY

4.1 STARS Overview

Raytheon has developed the Standard Terminal Automation Replacement System (STARS) to provide automation to support air traffic control within the Federal Aviation Administration (FAA) Terminal Radar Approach Control (TRACON) and Department of Defense (DoD) Terminal Control regions. STARS will eventually replace the existing Automated Radar Terminal System, which are nearly 30 years old and have been patched repeatedly over the years. STARS is composed of products developed in-house (such as AutoTrac), along with modified versions of commercial-off-the-shelf hardware products and commercially available software [11].

STARS will be used by human air traffic controllers to perform terminal radar air traffic control functions as defined in FAA Handbook 7110.65, “Air Traffic Control.” These roles and responsibilities will be unchanged with the deployment of STARS. STARS will be used to support both search and beacon radar target identification, maintenance of target identity, radar separation, traffic and weather advisory services, and navigational assistance to participating aircraft [11]. STARS will be used to assist both interfacility and intrafacility radar handoffs, and provide safety functions such as conflict alert (CA)/Mode-C intruder (MCI) and minimum safe altitude warning (MSAW) [11].

STARS is capable of tracking up to 1350 airborne aircraft simultaneously within a terminal area. The system interfaces with multiple radars (up to 16 short and long range), 128 controller positions, 20 remote towers, and a 400 by 400 mile area of coverage.
Initial Operational Capability (IOC) of STARS Early Display Configuration (EDC) was achieved at El Paso, Texas in December, 1999 and Syracuse, New York, in January, 2000. As a joint procurement of the FAA and DoD, STARS will be deployed at 570 FAA and DoD terminal and tower facilities [12].

4.2 STARS Function Selection

STARS is a substantial software system, with a complicated protocol to maintain data integrity, detect and diagnose failures in client facilities. There are also separate network protocol associated with software updates and synchronization. Thus, it is beyond the scope of this project to model the entire STARS system. Instead, specific STARS modules were chosen as candidates for SpecTRM models. These modules either contained complex logic whose rationale spanned several levels of design, or are human-machine interface intense and required extensive usability documentation.

4.2.1 Conflict Alert

The STARS Conflict Alert/Mode-C Intruder (CA/MCI) function detects conflicts between pairs of aircraft, based on location, altitude, and velocity. It is intended to provide 30 seconds of warning before predicted impact and to notify the air traffic controller so that he/she may issue warnings to both planes. The CA/MCI module is especially rich in FAA standards and predictive logic, which makes it an excellent candidate for SpecTRM. This case study will focus on the System Purpose, System Design Principles, and Blackbox Behavior levels of the CA/MCI module.
4.2.2 Minimum Safe Altitude Warning

The STARS Minimum Safe Altitude Warning (MSAW) function detects when a plane is flying to low or too close to the terrain, and provides safeguards against flight into the terrain. MSAW generates alerts whenever a tracked plane falls below a minimum safe altitude or is predicted to fall below it within an adapted amount of time. MSAW also issues warnings to the air traffic controller so that he/she may notify the pilot of the plane. MSAW is also rich in FAA separation standards and predictive logic, making it a good candidate for SpecTRM. The System Purpose, System Design Principles, and Blackbox Behavior levels of MSAW are presented in a separate paper.

4.2.3 Sector Handoff

Sector handoff refers to situations when responsibility for a plane is transferred from one air traffic controller to another. This situation usually occurs when a plane crosses the boundary between one controlling sector and another. The Sector Handoff function in STARS ensures that the transition in control is smooth and information about the transition is synchronously communicated to all involved parties. The Sector Handoff involves substantial human-machine interaction, making it another good candidate for SpecTRM. The System Purpose, System Design Principles, and Blackbox Behavior levels of MSAW are presented in a separate paper.

4.3 Modeling Procedure & Results

Modeling CA/MCI in SpecTRM was an iterative process. To gain an overall understanding of STARS and the motivation behind it, documents such as articles from Aviation Week, FAA’s RFP (request for proposal), and Raytheon proposals were
reviewed. An initial meeting with members of the STARS design and development team determined project focus and timeline. It was during this meeting that CA/MCI was chosen for this project.

Before modeling, a thorough understanding of CA/MCI was needed. Raytheon provided documentation used at all stages of development, including design descriptions, software specifications, and user’s manuals. In addition, a workshop session was scheduled with STARS system engineers Brian Donnelly and George Weigler to clarify subtle details in the documentation [13].

Modeling began with an overview phase. In the overview phase, an outline of how CA/MCI, as it was developed by STARS, functioned. This outline included descriptions of key functions within CA/MCI, their inputs and outputs, and the flow of information between these functions. Model boundaries were set during this process. This model focuses only on the conflict detection logic; actual generation of the alert and display to the controller are outside the model boundary.

Level I modeling focused on the high-level system requirements for CA/MCI. Most of these were obtained directly from Raytheon requirements documents. Others were added as justifications for some of the decisions encountered during the creation of the outline. For CA/MCI, these requirements included conditions for checks, frequency of checks, and rules for interaction with its environment. The Level I model for CA/MCI can be found in Appendix A.

Level II modeling focused on the design principles that were motivated by the high-level requirements. This model was divided into two parts. The first contained FAA specifications that served as a theoretical foundation for conflict checking methodology.
The *ARTS Computer Program Functional Specification (CPFS): Conflict Alert (NAS-MD-632)* document describes a set of conflict checks that are exhaustive across all current air traffic situations, and these were included in the Level II model [14]. The second part is STARS-specific, and describes how each of the high-level requirements are fulfilled by specific CA/MCI procedures. Much of the information from the overview phase was used in this description. It is noted that many of the procedure names and variable names used in STARS CA/MCI were identical or analogous to those referred to in *NAS-MD-632*. The Level II model for CA/MCI can be found in Appendix B.

Level III modeling created a black box representation of CA/MCI. This level was the most complicated to model. An internal state diagram was constructed to represent the supervisory mode, control mode, and inferred states for CA/MCI. In addition, all possible values for each of these modes are specified in the diagram. Input sources and output sinks were added to this diagram. For each of the sources and sinks, all relevant input and outputs are enumerated. To model the CA/MCI logic, the output variables were used as a reference point. For each output variable, it was determined which input variables and/or internal state variables contributed to the value. These variables formed the basis of an AND/OR table that specifies how the output value is determined. Together, these logic tables and the overall internal state diagram specify the behavior of CA/MCI at the black box level. The Level III model for CA/MCI can be found in Appendix C.
5 Evaluation

This project is a proof-of-concept demonstration of the SpecTRM modeling methodology. In the STARS case study, the SpecTRM model was generated based on available information from a system that was already developed. In other words, the model was developed AFTER the system, which would lead one to disregard its practicality. However, the purpose of this project is not to design a system from scratch using STARS. Rather, it is to demonstrate how different requirements from different levels of design can be woven together into a model that is readable and useful to personnel at all levels of development and use, from the programmers to the system engineers to the air traffic controllers themselves.

5.1 Existing STARS Documentation

ARTS Computer Program Functional Specification (CPFS): Conflict Alert (NAS-MD-632) was created by the FAA and last updated in May, 1998. CPFS defined all of the aircraft collision logic that was used in CA/MCI. The logic was explained with a theoretic foundation, highly mathematical in nature, and contains built-in FAA biases. For instance, many of the conflict detection tests applied solely to certain air traffic conditions, and don’t seem to make much sense by themselves. Only when justified by high-level requirements from the FAA does the reasoning tie in with the motivation. The document also carefully left out physical values for parameters, so specifications can be easily updated.

The Software Requirements Specification (SRS) for the Radar Data Processing System was created by Raytheon and last updated February, 2001 [15]. SRS described the
high-level system purpose for CA/MCI in a series of “shall” statements. STARS-specific conflict detection logic were also described in detail, although they conform very closely to the description in CPFS. Information in the SRS was used in the development of the System Purpose and Blackbox Behavior levels.

The existing documentation of various STARS components (including CA/MCI) is thorough and extensive. Several volumes of specifications and descriptions exist, ranging from high-level FAA requirements descriptions to System/Subsystem Specifications to the software user’s manual [16]. Each document provides a good overview of the overall STARS design (although they are essentially identical across different documents) before delving into subsystem specifics. The Raytheon documents provide good description of how CA/MCI functions, carefully naming the various subfunctions and specifying input/output parameters as well as internal variables. In addition, they reliably linked each design decision back to a high level FAA-mandated “shall” statement.

Where the documents fall short lies in the incremental traceability of the FAA “shall” statements through the system design principles to the blackbox behavior level. In the Raytheon documentation, the descriptions of various subfunctions were justified by pointing directly back to the high-level “shall” statements, with no explanation of the design principles behind how the requirement is fulfilled. In Raytheon’s documentation of blackbox behavior and information flow, some input/output variable names were not well explained, or their sources were undisclosed. The SpecTRM model introduced links between the “shall” statements to the system design principles (provided by the FAA, and referenced only indirectly by the Raytheon documents), and between the principles to the
blackbox behavior level. In addition, the SpecTRM model introduced additional state variables and modified the input/output logic to facilitate tracing each output to its affecting inputs.

5.2 SpecTRM Strengths

The primary strength of SpecTRM lies in its flexibility. For the CA/MCI modeling exercise, copious amounts of information were available, at all levels of design, development, and use. Because the format for SpecTRM requirements is primarily document-based, the information in these reference documents is easily transferable to SpecTRM. The power of SpecTRM lies in the backbone structure in which to categorize and link this information. Information related to decisions and their consequences are easily accessed when the document-based information is organized in a SpecTRM document-based model.

The state-machine based syntax of SpecTRM’s Level III methodology is particularly effective in capturing the system’s response to all possible inputs and events. With the language, it is possible to completely specify all of the possible states (and associated characteristics) the system can be in, and to enumerate through all possible stimuli to show state transitions. In fact, this syntax is what makes the model executable.

5.3 SpecTRM Weaknesses

Throughout the modeling process, some weakness areas in the SpecTRM language were also encountered. These areas are specific in that they directly applied to the CA/MCI case study, but can be generalized for most SpecTRM models.

It was difficult to model a display mode for CA/MCI. In the Raytheon design documents, conflict alerts can be specified in different ways. Conflicts can be current,
predicted, imminent, etc. Each conflict requires a different display and computer-human interaction. This variation is exacerbated by the fact that the controller can manually suppress certain types of conflict alert functions, and thus prevent some of them from being displayed at all. In past work, modeling of display modes were “pushed aside” into a HMI module that is developed separate from the actual conflict alert generator. However, the display logic should be integrated with the conflict alert logic, so that the human-computer interaction loop within CA/MCI can be closed (from detection to display to acknowledgement).

The Level III Blackbox Behavior model is perhaps the most difficult to create, because it must reconcile system design principles with actual procedural software code. There is rarely a trivial diagram to show this translation. It is not surprising, then, for the CA/MCI modeling exercise to suggest some improvements for the Level III syntax. The AND-OR tables used to simplify output logic is a very powerful tool, translating complex conditional statements into a basic table. However, these tables are derived from computer science, and may not be the most effective method of presentation for system engineers. It may be instructive to pull out the logic from these tables and order them in a flow chart. Although this increases the size and complexity of the diagram, it does two important things. First, it enumerates all of the possible combinations of inputs (which may bring insight into test cases that were overlooked). Second, it orders the sequence in which the input/internal variables are tested/evaluated, which reveals which variables are the most critical (and thus should be most closely monitored).

The SpecTRM Level III syntax is based on a state machine modeling language. It clearly describes outputs and state changes based on atomic events. However, in many
instances (and CA/MCI is one), a number of different procedures run sequentially for a given event. The logic tables are not well suited for capturing the sequential nature of operations. For instance, in CA/MCI, an event will trigger CA/MCI to check for conflicts every five seconds. During each run, CA/MCI will first generate a list of track pairs, then determine the volume type of each pair, then perform each of the conflict alert checks on each pair. This order is not clearly reflected in the output logic on the Level III model. There should be a structure to aggregate a list of operations that run in sequence.

The most difficult task in modeling the CA/MCI Blackbox Behavior was keeping track of the various input/internal/output variables. Most variables are either inputs or outputs to the system, but a few are internal. The internal variables were challenging to model because they could represent either temporary or permanent data. Most data is temporary: a list of aircraft is passed to CA/MCI each time it runs, so at the end of its execution, this list is erased, and reinitialized the next time CA/MCI runs. However, some of the data is permanent: the list of adapted RCB’s for the system resides unchanged in the CA/MCI until it is updated by a central STARS facility. Currently, there is no syntax in SpecTRM to distinguish between these two types of internal variables.

5.4 Project Limitations

STARS is a vastly complicated software system, composed of new and old software developed at a variety of companies. As such, it is beyond the scope of this project to model the entire STARS system. Instead, the project focused on modeling the complex logic of a critical STARS module, CA/MCI. There are drawbacks for limiting the scope. Some levels of correctness evaluations are lost. For CA/MCI, the SpecTRM model
specifies how the module will or will not work correctly given appropriate inputs, but cannot appropriately specify how to tell if the module is working (for instance, it could be powered down, and we would have no indication of its status). This diagnosis must be performed by modules external to CA/MCI, and are lost once the boundaries have been drawn around CA/MCI.

Difficulties were also encountered during the creation of the model using SpecTRM software. The software is relatively new, and is still in the Alpha test stage. As a result, many features are unreliable and unstable, which lengthened the model development process. As an alternative, the models were created using other software. The System Purpose and System Design Principles levels were created using Microsoft Word (with references to links that would be placed once it was ported to SpecTRM), and the System Blackbox Behavior level was created using object modeling software such as Visio.
6 Conclusion

The SpecTRM modeling exercise yielded encouraging results. At the very least, it demonstrated the ability of this new language to integrate and present currently available intent and requirements information in a new structure. The SpecTRM modeling methodology and software consolidated FAA requirements specifications, Raytheon design descriptions, and Raytheon user manuals, and presented them in a hierarchical format that allows anyone to incrementally trace development from an external requirement, through internal design principles and design decisions, to the finalized documented software.

During the process, there were many lessons learned regarding use of the SpecTRM modeling methodology. There were certain characteristics encountered during the Blackbox Behavior phase that could not easily be modeled using current SpecTRM syntax and structure. This project suggests changes to enrich the language, by adding display modes, providing alternatives to AND-OR tables, and incorporating structures that aggregate sequential operations and distinguish between temporary and permanent data. The power of the SpecTRM modeling language was clearly shown through the CA/MCI case study. With a few changes/additions to the language syntax, it will reach full maturity.
7 Future Work

This project was a part of a proof-of-concept demonstration of SpecTRM as a modeling tool for Raytheon’s Standard Terminal Automation Replacement System. It focused on the CA/MCI module of STARS. Other projects are focusing on other modules in STARS. Together, they will comprise a complete SpecTRM model of Raytheon’s STARS.

This thesis applies the current methodology to parts of STARS in order to evaluate its good and bad features and applicability, and to suggest further improvements that might be made. Unfortunately the automated simulation and analysis tools were not available by the time the thesis had to be completed and so the thesis concentrates on the specification and modeling components of SpecTRM. Follow up work will extend the empirical evaluation to the analysis tools.
8 APPENDIX A

Level I: SYSTEM PURPOSE

Overview
The Conflict Alert/Mode-C Intruder (CA/MCI) function generates alerts to the air traffic controller’s data monitor (Situation Data Display, or SDD), whenever a pair of aircraft are determined, based on location, velocity, or both, to be in violation of minimum safe separation standards. This violation can be based on the current situation, or can be based on prediction. Aircraft track pairs are generated from the list of aircraft in the given sector. Based on their location and altitude, separation standards are determined, and each pair is passed through a series of conflict detectors. The conflict detectors test for safe separation violation, and if one or more violations occur, the pair will be flagged, and appropriate information will be provided to other STARS function from which alerts will be generated.

High Level System Goals
CA/MCI shall provide accurate and timely detection of hazardous conditions created when a pair of aircraft is in a conflict situation, or if they are predicted to be in a conflict situation, and alert the controller of the situation.

High Level Functional Requirements
CA/MCI’s specifications encompass the processing of flight information for potential conflicts, and generating alarms for those determined to be in conflict. CA/MCI will produce a list of pairs of tracks (aircraft) that are determined to be in current or predicted conflict, and generate alarms for these pairs through the Conflict Alert module.

[FR.1] Provide collision avoidance protection for any two aircraft that will be in conflict with a look ahead time of at most 120 seconds
Rationale: FAA requires 40 second look ahead time, which allows for a 30 second warning time

[FR.2] CA/MCI shall detect conflicts between aircraft pairs on parallel approach in airport areas adapted for parallel runways

[FR.3] CA/MCI shall detect all conflicts between associated/unassociated Mode C altitude reporting aircraft track pairs

[FR.4] CA/MCI shall detect current conflicts using current track position and data
[↓PR.3]

[FR.5] CA/MCI shall detect predicted conflicts using track position, altitude, and altitude change rate data

[FR.6] CA/MCI shall perform conflict detection checks every 5 seconds [→EI.3]
Intent: currently, the fastest radar update is every 5 seconds

29
CA/MCI shall have the ability to detect 100 simultaneous CA situations [PR.2]

**Intent: Performance Specifications in STARS SSS**

CA/MCI shall, for every aircraft pair that is determined to be in conflict alert, send a Conflict Alert Continuous Data Recording message to the Data Recording Facility (DRF) for off-line analysis [PR.35]

**Intent: provides a way to roll back in case of synchronization issues between data facilities**

### Environment Description

The CA/MCI function is part of the Standard Terminal Automation Replacement System (STARS), and interacts with specific STARS components.

CA/MCI takes as input the state of the airspace sector that it is responsible for monitoring, represented as a list of tracks (aircraft). For each of these tracks, it also needs aircraft information and flight plans. Most of this information is provided by other STARS components, such as the Flight Data Processing System.

Output from the Conflict Alert module will be sent to the Situation Data Display (SDD), which interfaces with the human controller, and to the DRF.

CA/MCI also has direct input/output interaction with the human controller. The controller may wish to enable/disable the CA/MCI function for an airspace sector, individual aircraft, or pairs of aircraft. These will be inputs to CA/MCI, and CA/MCI would output acknowledgement messages of success or failure.

### Environment Assumptions

Environment assumptions are assumptions about other system components (human and non-human) that affect the design of CA/MCI. The validity of these assumptions is beyond the control of CA/MCI, but the correctness of CA/MCI depends upon the validity of these assumptions.

#### STARS Input Sources

- **[EI.1]** High-integrity communications exist between aircraft and ground station
- **[EI.2]** There will not be more than 1,350 tracks in the system at any single point in time
- **[EI.3]** The CA/MCI will be reliably provided with a list of tracks in the monitored sector every 5 seconds [FR.6]

**Rationale:** CA/MCI will run every 5 seconds, so input data to CA/MCI must be provided at least as frequently.

#### Situation Data Display (Human/Machine Interface)

The display facilitates bi-directional information transfer between CA/MCI and the controller. It translates information generated by CA/MCI to a format that’s easy for controllers to understand, and allows controllers to respond to information by entering acknowledgements and other commands. The SDD software allows the controller to create, delete, and modify flight plans and to manually associate/disassociate flights. The controller can also manually transfer control of flights or flight plans to another controller.

**[ED.1]** The SDD will display information (including alarms) generated by CA/MCI

[PR.35]
Rationale: Alerts generated by CA/MCI need to be visible to the controller through the SDD

[ED.2] The HMI shall provide a way for controllers to enable/disable CA/MCI

Intent: CA/MCI module may be malfunctioning, which would distract the controller if not disabled. In some instances, disabling the CA/MCI function for some circumstances would reduce the number of false alerts

[ED.3] Alarms shall be noticeable enough for the controller to notice them within the alarm duration time

Intent: There will be information about all aircraft in the sector displayed on the screen, thus lots of data for a controller to process at any given time. Conflict Alerts must take highest priority to resolve, and thus should be clearly distinguished from other information

Operator

[EO.1] The controller will rely on data generated by CA/MCI, presented through the SDD, in order to respond to alerts and coordinate aircraft that are in a conflict situation

[EO.2] The controller shall be properly trained to handle emergencies due to conflict alerts generated by CA/MCI by giving appropriate instructions to pilots of the aircraft in potential conflict

[EO.3] The controller shall take an appropriate response to a CA/MCI alert, which involves communication with the human-machine interface and communication to pilot or other controllers, in 10 seconds [$\rightarrow$ C.5]

Rationale: Alarms have a fixed duration

[EO.4] After a CA/MCI alert is resolved, the controller shall return to his/her normal duties

System Limitations

[L.1] CA/MCI does not perform conflict checks for frozen tracks
[L.2] CA/MCI does not perform conflict checks for tracks without associated valid Mode-C data

Rationale: Unreliable altitude data will result in inaccurate predictions

[L.3] CA/MCI does not perform conflict checks for tracks in which neither track has an associated flight plan

Rationale: If a track is not flight plan associated, then no air traffic controller is aware of it. Thus the safety of the track pair is not the responsibility of the controller

[L.4] CA/MCI will not perform conflict checks for aircraft that are considered to be in the landing/take-off phase
[L.5] CA/MCI is dependent on the accuracy of input data
[L.6] CA/MCI may be turned off or disabled by the human controller, which will inhibit ability to report detected conflicts to the controller
[L.7] Aircraft performance or flight crew cognitive limitations may constrain the magnitude of the escape maneuver that the flight crew can safely execute in response to a resolution advisory.
System Constraints

There are many possible implementations of CA/MCI. System constraints are restrictions on how the system can achieve its purpose. They limit the possible solutions.

[C.1] CA/MCI must comply with all applicable FAA and FCC policies, rules, and philosophies

*Intent: FAA request*

[C.2] CA/MCI conflict detection logic shall be the same as previous generation (ARTS)

*Rationale: Customer request*

[C.3] CA/MCI must work with a low level of unwanted nuisance alarms

[C.4] Alerts generated by CA/MCI must not interfere with the controller’s performance of his/her other duties

[C.5] Alerts generated by CA/MCI must have a fixed duration time of 10 seconds

*Intent: Avoid causing indifference by controllers*

[H.1] Two aircraft are, based on their current altitude and lateral positions, in violation of minimum safe separation standards with each other

[H.2] Two aircraft are, based on their current positions and velocities, predicted to be in violation of minimum safe separation standards within a predefined amount of time

[H.3] Two aircraft approaching parallel runways are, based on their current positions and landing trajectory, in violation of minimum safe separation standards with each other
9 APPENDIX B

CA/MCI System Components

The following is a list of components that together perform the CA/MCI function. Their description is taken from Raytheon’s *Software Requirements Specification for the Radar Data Processing System* (Feb. 5, 2001). Each description is followed by the design principles on which they are based. The design rationale was specified by FAA, and taken from the document *Arts Computer Program Functional Specification (CPFS): Conflict Alert (NAS-MD-632)*, last revised May 1998. In this description, the following formatting conventions are used:

*Italics* are used to denote input parameters to CA/MCI

*Bold Italics* are used to denote variables internal to CA/MCI

CA/MCI System Overview

The CA/MCI function generates alerts for pairs of aircraft (also called tracks) that are in current or predicted conflict. A conflict situation between two aircraft occurs when they are not far enough apart. Each run of CA/MCI involves several components. A list of track pairs is first generated. A volume type is then assigned to each track pair, in order to determine minimum separation standards. Then, each pair is subjected to a series of four conflict alert tests. Alerts will be displayed for track pairs that are determined to be in conflict alert.

Generate & Filter Candidate Track Pairs

[PR.1] The STARS subfunction **Coarse_CA_MCI_Filter** generates candidate track pairs. All non-frozen tracks transmitting valid Mode C altitude data are eligible for conflict alerts, and will be paired up with potential conflict alert partners. Tracks that are frozen, tracks that do not have valid Mode C data, and any generated track pairs in which neither track is flight plan associated or in which the tracks belong to different exercises will be exempt from all conflict alert processing until the next iteration of **Coarse_CA_MCI_Filter**.

[PR.2] The **Coarse_CA_MCI_Filter** exists to reduce the computation load for the minimum separation violation tests. **Coarse_CA_MCI_Filter** filters out track pairs that are so far apart that they will never collide before the next iteration of the CA/MCI tests. This filtering reduces the number of track pairs that need to be processed by the minimum violation tests, and increases performance and scalability. [↑FR.7]

[PR.3] All tracks in the STARS track file are stored in *track_data* and are checked to determine which ones are eligible for conflict alert processing. Those that are eligible are paired up with potential conflict alert “partners.” Each eligible track can be paired up with any other eligible track in its conflict search volume. The conflict alert volume is defined in each tile of the *radar_mosaic*, and extends from the radar tile floor...
to an adapted eligibility altitude. The search volume for a track is calculated by using the
adapted look ahead time, the maximum possible lateral velocity of an aircraft, and the
largest adapted lateral separation area. Any tracks that are in the track’s search volume
will be paired up with the given track. [PR.4]

[PR.4] Once a pair is determined, it is then passed through a geographic filter. A
horizontal check is applied first. For each track in the pair, a box (horizontal plane) is
drawn based on the same parameters used to calculate the search volume (using the
adapted look ahead time, the maximum possible lateral velocity of an aircraft, and the
largest adapted lateral separation area). If there is any overlap between the two boxes,
then the vertical check calculates the difference in the vertical positions of the track pair.
If the difference is less than or equal to the largest adapted vertical separation, or if the
tracks are converging vertically, then the pair is a valid candidate track pair.

DESIGN PRINCIPLE
Source: NAS-MD-632

Primary Filter
[PR.5] Each track in the system is processed by CA/MCI once per scan as new
coordinate data is received. The track, called a primary track, must theoretically be
paired with every other track in the system to determine if it is in conflict. A track that is
paired with the primary track is called a secondary track. Thus, if there are N tracks in
the system, the number of track pairs is O(N^2). Primary Filter should reduce the number
of primary/secondary track pairs that undergo the conflict detection tests in CA/MCI.
The Primary Filter is designed based on the Primary Filter theorem:

Primary Filter Theorem
Let C_1 be the current position of the primary track, and V_1 be the speed of the primary
track.
Let P_1 be the time T projected position of the primary track. That is, P_1 is the position
where the primary track will be at time T assuming its speed and heading do not change.
T is expressed relative to current time.
Let C_2, V_2, and P_2 be the analogous quantities for the secondary track.

If these tracks will approach each other closer than distance Q within time T, assuming
their speed and heading do not change, then at least one of the following two conditions
must be satisfied:

\[ |C_1 - P_2| \leq V_1 \times T + Q \] (1)
\[ |C_2 - P_1| \leq V_2 \times T + Q \] (2)
[end of Primary Filter Theorem]

All secondary tracks with satisfy inequality (1) are passed to the CA/MCI detection
algorithms; the others (which satisfy inequality (2)) need not be considered at this time,
because at some time during the scan, the secondary track will become the primary track
and the second inequality will in effect become the first inequality.
The quantity $V_1 \cdot T + Q$ is called the primary filter search distance, and is approximated by the quantity $D$: 

$$D = (|X_1| + |Y_1|)(T_a + T_s) + Q + A$$

| $X_1$ | = X component of the velocity  
| $Y_1$ | = Y component of the velocity  
| $T_a$ = Lookahead time CA_T3LkaQ (40 seconds)  
| $T_s$ = One scan time  
| $Q$ = LINCON Type 3 lateral separation threshold CA_T3LATQ (SP = 1.2 NM)  
| $A$ = Augmented Search Increment = 3 nautical miles, a value determined to be large enough to detect all MCI conflicts

The projected positions are updated for each aircraft at most once during the scan. Thus, when they are used, they can be as much as one scan old. For this reason, the time $T$ is set to $T_a$, which is the LINCOLN Lookahead time CA_T3LKAQ (SP = 40 seconds) plus one scan time $T_s$.

[PR.6] With the addition of the Mode C Intruder (MCI) function, unassociated tracks must also be processed, thus increasing the number of Conflict Alert tracks. To reduce processing time per scan to a manageable level, unassociated tracks are processed as primary tracks only after they have first been detected to be in conflict for one scan (they exist in the Conflict Table). This modification violates the assumptions of the Primary Filter theorem, resulting in some MCI conflict pairs going undetected. To prevent this, an additional distance, the Augmented Search Increment, is added to the search distance of a primary track if the speed of the track is less than 200 knots. [†C.3]

The inequality 1 check is done in two steps. Let xpos and ypos denote the X and Y coordinates of a track. Let subscripts be defined as follows:

1 = Primary track  
2 = Secondary track  
c = Current position  
p = CA_T3LKAQ Second (SP) projected position

Then, a track pair is passed on to the detection algorithms if the following 2 conditions hold:

$$|x_{pos_{1c}} - x_{pos_{2p}}| \leq D \quad (3)$$  
$$|y_{pos_{1c}} - y_{pos_{2p}}| \leq D \quad (4)$$

It can be seen that if a track pair satisfies inequalities (3) and (4), it will also satisfy inequality (1).

To reduce the possibility of excessive false alerts, intersensor MCI pairs are not passed to the detection algorithms for processing. Most track pairs passed on by the Primary Filter will satisfy both inequalities (1) and (2). Consequently, all such non-MCI pairs would normally be processed twice per scan (once when each of the tracks is processed as the primary track). To avoid this duplicate processing, the following duplicate pair logic is implemented:
When a track pair is to be passed on to the detection algorithms for processing, it is determined if it has already been passed on this scan. If the secondary track has been processed as a primary track during this scan and if inequality (2) is satisfied, then the pair has already been passed on, and the current pair is rejected.

**Minimum Separation Violation Tests**

The minimum separation violation tests are performed by the `Fine_CA_MCI_Filt`er subfunction. For each track pair identified as a candidate track pair, the volume type is first determined in order to assign the appropriate minimum separation standards. Then, the pair is tested for minimum separation violation. Alerts will be generated for pairs that fail one or more of the conflict tests.

**Volume Type Determination**

Each track in the pair will be assigned a volume type. Horizontal Separation Standards differ according to the plane’s current airspace. There are three types of volumes, Type I, Type II, and Type III. Type I and Type II volumes are in the Runway Capture Box (RCB). Type I represents the immediate vicinity of an airport, and encompasses the actual runway. Type II encompasses the approach and departure corridors for the runways. Airspace outside the RCB is considered to be Type III.

In assigning volume type, `Fine_CA_MCI_Filt`er first checks to see if the track is in a RCB volume. For each RCB adapted for the system, it checks three conditions:

1. the track’s x and y coordinates are inside the RCB’s area
2. the track’s altitude is below the RCB’s ceiling altitude
3. the track’s heading lines up with the RCB’s heading within an adapted threshold

\[ \text{trk\_hdng\_dev} \]

[\text{\textasciitilde FUNCTION: Aircraft Volume Type}]\n
If these conditions are satisfied, the track is considered to be in the RCB. If a track fits more than one RCB (this is possible because RCB’s can overlap), additional comparisons are needed to determine the best-fit RCB. These comparisons are prioritized below (with the first being the primary criteria):

1. Track heading – compare the track’s heading with the RCBs’ headings. The RCB with the smallest heading difference is used, unless all of the differences are less than the adaptable value \( CA\_wry\_ovrlp\_heading\_diff(MSAW\_CA\_alert\_calculation) \). In this case, this test is inconclusive.
2. Track distance – compare the track’s distance from each runway’s centerline. The runway with the smaller distance is used.

Once the RCB has been determined for a track, the track’s x and y coordinates are used to determine if it is a Type I or Type II volume. Tracks that are Type I or Type II will have airport and runway ID’s attached to them. Tracks that do not belong in any RCB are given a Type III volume label.
Each pair is assigned a volume label based on the volume labels of each track in the pair. The logic is described in the following table.

<table>
<thead>
<tr>
<th>First Track Volume Type</th>
<th>Second Track Volume Type</th>
<th>Track Pair Volume Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>III</td>
<td>I</td>
<td>I - If in same airport</td>
</tr>
<tr>
<td>III</td>
<td>II</td>
<td>III - If not in same airport</td>
</tr>
<tr>
<td>I</td>
<td>III</td>
<td>I - If in same airport</td>
</tr>
<tr>
<td>II</td>
<td>III</td>
<td>III - If not in same airport</td>
</tr>
<tr>
<td>III</td>
<td>III</td>
<td>III - If not in same airport</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>III - If not in same airport</td>
</tr>
<tr>
<td>II</td>
<td>I</td>
<td>III - If not in same airport</td>
</tr>
<tr>
<td>I</td>
<td>II</td>
<td>II - If in same airport</td>
</tr>
<tr>
<td>II</td>
<td>II</td>
<td>III - If not in same airport</td>
</tr>
</tbody>
</table>

**Airport Suppression**

[PR.8] If the track pair volume type is Type I or Type II and either of the tracks is below the airport conflict alert floor altitude (APFLOOR in MSAW_CA_airport_data), then the pair will be considered as not in a conflict alert condition and no further processing of the pair is required. *Rationale: to protect against issuing alerts between arriving aircraft and aircraft on the ground. [↑C.3]*

In this situation, management of aircraft is handled either procedurally (with pilots and air traffic controllers following a predefined protocol), or with the aid of a separate automation system.

**Conflict Detection**

Violation of minimum safe separation standards depends on the track pair volume type. There are four kinds of conflict detectors: Linear Conflict Detector (LINCON), Maneuver Conflict Detector (MANCON), Proximity Conflict Detector (PROCON), and Predicting Maneuver Conflict Detector (MANCONP).

**Linear Conflict Detector (LINCON)**

[PR.9] LINCON detects conflicts based on the projection of the track’s current speed and heading. LINCON evaluates the altitude time interval of a conflict (the interval of time that track pairs will enter and leave an altitude separation that is considered potentially dangerous) and the horizontal time interval of a conflict for a track pair. If these two intervals exist and overlap each other between the current time and the adapted conflict alert look ahead time CA_lkhd_time, then a conflict exists for the track pair.
LINCON first determines if two aircraft are flying in parallel. If two aircraft are flying in parallel (either vertically, or laterally), then their paths will not intersect anytime in the immediate future. However, they can still generate a LINCON alert, if they are flying too close to each other to begin with. Two aircraft are flying too close if both their vertical and lateral separations are less than a predefined threshold. [FUNCTION: LINCON check]

If the two aircraft are not flying in parallel, then two intervals are calculated (refer to the design logic section below for more detail): an altitude conflict interval and a lateral conflict interval. [FUNCTION: LINCON check]

An altitude conflict interval is the period of time when two aircraft are vertically separated by a distance below a predefined threshold. When two aircraft are converging vertically, the time at which their vertical separation falls below the threshold is the time of altitude violation when converging. Then, the time at which their vertical separation increases above the threshold (e.g. the aircraft have vertically “crossed paths” and are now diverging) is the time of altitude violation when diverging. Note that if aircraft are diverging vertically, they don’t need to be as far apart to be safe, so the threshold is lower for aircraft pairs that are diverging. [FUNCTION: LINCON check]

Similarly, a lateral conflict interval is the period of time when two aircraft are laterally separated by a distance below a predefined threshold. The calculation for time of lateral violation when converging and time of lateral violation when diverging is similar to that of the vertical conflict interval, except that lateral positions, velocities, and thresholds are used. [FUNCTION: LINCON check]

If there is an overlap between the altitude and lateral conflict intervals, then there will be an interval where the two aircraft will be too close both vertically and horizontally. If the beginning of this interval is less than a predefined look ahead time, then an alert will be generated for this aircraft pair.

Tracks that are in current or predicted conflict in LINCON are further processed to see if the conflict alert is imminent. [FUNCTION: LINCON check]

DESIGN PRINCIPLE
Source: NAS-MD-632

Intervals of Conflict
[FUNCTION: LINCON check]

To determine if two tracks are in conflict, the altitude and the lateral dimensions must be considered separately. For example, it may be safe for two tracks to be at the same lateral position and separated by 500 feet in altitude, but it is not safe for tracks to be at the same altitude and separated by 500 feet laterally. [FUNCTION: LINCON check]

A conflict situation is determined by first evaluating an altitude interval of conflict. The time at which the two aircraft are predicted to first come in conflict in altitude is the beginning of this time interval, called AL1. The time at which the aircraft pair is
predicted to leave the altitude conflict is the end of this time interval and is called AL2. If AL2 is negative, then no altitude conflict exists and the aircraft pair cannot be in conflict.

Next, a lateral conflict interval is determined. The time at which the two aircraft are predicted to first come in lateral conflict is the beginning of this time interval and is called LAT1. The time at which the aircraft pair is predicted to leave the lateral conflict is the end of this time interval and is called LAT2.

If the altitude and lateral conflict time intervals overlap, then the larger of the numbers AL1 and LAT1 is the time at which the aircraft pair will first come into altitude and lateral conflict concurrently. This number is called the Time of Violation (TOV). If TOV is less than or equal to the Look Ahead Time, then a conflict exists. The Look Ahead Time is an area dependent system parameter CA_T3LKAQ for Type 3, CA_T2LKAQ for Type 2, and CA_T1LKAQ for Type 1.

**Altitude Conflict Interval**

[PR.14] Let the two aircraft being considered be designated as aircraft A and aircraft B. Let aircraft A be at altitude Z1 with an altitude change rate of $\dot{Z}_1$. Let aircraft B be at altitude $Z_2$ with an altitude change rate of $\dot{Z}_2$. Let:

\[
\Delta Z = Z_2 - Z_1 \\
\Delta \dot{Z} = \dot{Z}_2 - \dot{Z}_1 \\
\Delta Z_t = \Delta Z \text{ at some variable time } T
\]

Then the altitude difference between the two aircraft at some variable time T is given by the equation:

\[
\Delta Z_t = \Delta Z + \Delta \dot{Z} \times T
\]

Solving for T yields:

\[
T = \frac{(\Delta Z_t - \Delta Z)}{\Delta \dot{Z}} \quad (1)
\]
The minimum altitude threshold between two aircraft is denoted by ZALQ, an airport area dependent system parameter CA_T3ZALQ for Type 3, CA_T2ZALQ for Type 2, and CA_T1ZALQ for Type 1. The time at which the two aircraft will be separated in altitude by distance ZALQ is denoted by \( Tz \) and is obtained by substituting ZALQ for \( \Delta Zt \) in equation (1).

\[
Tz = \frac{(ZALQ - \Delta Z)}{\Delta Z}
\]  

(2)

Recognizing that the aircraft may be plus or minus ZALQ feet apart, depending on which aircraft we designate as aircraft A, and also that the calculated time \( Tz \) may be the time of ZALQ foot convergence or ZALQ foot divergence, we change equation (2) to:

\[
Tz = \frac{(\pm ZALQ - \Delta Z)}{\Delta Z}
\]

The modified equation gives two values of \( Tz \). The smaller of these values is the Time of Altitude Violation when Converging (TAVC). The larger of these values is the Time of Altitude Violation when Diverging (TAVD). AL1 is set equal to TAVC. If TAVC is negative, AL1 is set to zero. [FUNCTION: Parallel Altitude]

The time at which the two aircraft are in the same altitude is called the Time of Co-Altitude (TOCA) and is obtained by setting \( \Delta Zt \) to zero in equation (1). Therefore:

\[
TOCA = \frac{-\Delta Z}{\Delta Z}
\]

The aircraft are converging in altitude until time TOCA, at which point they begin to diverge. A conflict situation can be considered to still exist for some period of time after they begin to diverge. For this reason, AL2 is defined as follows:

\[
AL2 = TOCA + CA_{ADAQ} \times (TOCA - TAVC)
\]

CA_{ADAQ} is the Altitude Divergence Allowance and specifies the amount of time after the point where altitude divergence begins, that aircraft should still be considered in conflict. CA_{ADAQ} is expressed as a fraction of the TAVC-TOCA interval. Currently, its value is 0.5.
If $\Delta \dot{Z} = 0$, then the aircraft are flying parallel in altitude and the equations for TAVC and TOCA do not apply because of a zero denominator. Even when $|\Delta \dot{Z}|$ is small, a small change in the altitude velocity component of one of the aircraft could result in a large change in TAVC or TOCA. This behavior is undesirable, so different logic is used to calculate the altitude interval when $|\Delta \dot{Z}|$ is small. \cite{PR.11}

If $|\Delta \dot{Z}| \leq CA_{\text{PARALQ}}$ (SP = 300 ft/min), the aircraft pair is considered slow closing in altitude. In this case, if $|\Delta Z| \leq ZALQ$, the aircraft are considered to be in perpetual altitude conflict. Thus, AL1 is set to zero, and AL2 is set to an arbitrarily large number (900 seconds in this case, which is fine as long as the Look Ahead Time is less than 900 seconds). If $|\Delta Z| > ZALQ$, then the aircraft are considered to be in perpetual non-conflict in altitude, and therefore no conflict exists.
Lateral Conflict Interval

[PR.15] In the X-Y plane, consider aircraft A at position A1, and aircraft B at position B1, and the aircraft heading in the direction of the arrows in the following diagram:

The distance between the aircraft is called the Current Lateral Separation (CLS). If both aircraft proceed with their current speed and heading, then at some time in the future, aircraft A will be at position A2 and aircraft B will be at position B2 such that the distance between them at this time will be exactly LATQ miles, where LATQ is the minimum lateral separation threshold. LATQ is an airport area dependent system parameter, CA_T3LATQ for Type 3, CA_T2LATQ for Type 2, and CA_T1LATQ for Type 1. The current value of CA_T3LATQ is 1.2 nautical miles. The time it takes for the aircraft to get to these positions is called the Time of Lateral Violation (TOLV).

If the aircraft continue to proceed at their current speed and heading, they will at some time in the future be at positions A3 and B3 such that the distance between them is as small as it is ever going to be, before they begin to diverge. This time is called the Time of Minimum Approach (TOMA) and the distance between them at this time is called the Lateral Miss Distance (LMD). [←PR.11]
Let aircraft A have coordinates $X_1, Y_1$ and velocity $\dot{X}_1, \dot{Y}_1$. Let aircraft B have coordinates $X_2, Y_2$ and velocity $\dot{X}_2, \dot{Y}_2$.

Using this data, define the following:

\[
\Delta X = X_2 - X_1
\]
\[
\Delta Y = Y_2 - Y_1
\]
\[
\Delta \dot{X} = \dot{X}_2 - \dot{X}_1
\]
\[
\Delta \dot{Y} = \dot{Y}_2 - \dot{Y}_1
\]

Then, CLS, TOMA, LMD, and TOLV are given by the following formulas:

\[
CLS = \sqrt{\Delta X^2 + \Delta Y^2}
\]
\[
TOMA = -\frac{\Delta X \Delta \dot{X} + \Delta Y \Delta \dot{Y}}{\Delta \dot{X}^2 + \Delta \dot{Y}^2}
\]
\[
LMD = \frac{\left|\Delta X \Delta \dot{Y} - \Delta Y \Delta \dot{X}\right|}{\sqrt{\Delta \dot{X}^2 + \Delta \dot{Y}^2}}
\]
\[
TOLV = \frac{-\left(\Delta X \Delta \dot{X} + \Delta Y \Delta \dot{Y}\right) - \sqrt{LATQ^2 (\Delta \dot{X}^2 + \Delta \dot{Y}^2) - (\Delta X \Delta \dot{Y} - \Delta Y \Delta \dot{X})^2}}{\Delta \dot{X}^2 + \Delta \dot{Y}^2}
\]

The beginning of the lateral conflict interval is LAT1 and is set to TOLV. The aircraft are converging laterally up until time TOMA and then they begin to diverge. A conflict situation can be considered to still exist for some period of time after they diverge. For this reason, the end of the lateral conflict interval, which is LAT2, is defined:

\[
LAT2 = TOMA + CA_\_LDAQ \ast (TOMA - TOLV)
\]

$CA_\_LDAQ$ is the Lateral Divergence Allowance and specifies the amount of time after the point where lateral divergence begins, that aircraft should still be considered in conflict. $CA_\_LDAQ$ is expressed as a fraction of the TOLV-TOMA interval. Its current value is 0.3.
Different logic is used to process aircraft pairs that have a small relative speed. Relative speed \( R \) is defined by the equation:

\[
R = \sqrt{\Delta x^2 + \Delta y^2}
\]

Relative speed can best be visualized as the apparent speed of aircraft B to a person traveling in aircraft A. If \( R = 0 \), the aircraft are flying laterally parallel with equal speeds, and the equations for TOMA, LMD, and TOLV do not apply because of a zero denominator. Even when \( R \) is small, a small change in a lateral velocity component of either aircraft could result in a large change in TOMA, LMD, or TOLV. Such an aircraft pair might go in and out of conflict from scan to scan due to this small change. To avoid this, different logic is used to calculate the lateral conflict interval when \( R \) is small.

If \( R < CA_{MINRELQ} \) (SP = 60 knots), the aircraft is considered slow closing laterally. In this case, if \( CLS \leq PARQ \), the aircraft are considered to be in perpetual lateral conflict. LAT1 is set to zero and LAT2 is set to an arbitrarily large number (900 seconds). PARQ is an airport area dependent system parameter \( CA_{T3PARQ} \) for Type 3, \( CA_{T2PARQ} \) for Type 2, or \( CA_{T1PARQ} \) for Type 1. If \( CLS > PARQ \), the aircraft are considered in perpetual non-conflict laterally, and therefore no conflict exists.

**Parallel Runways**

[PR.16] Aircraft making simultaneous departures or final approaches on parallel runways may normally approach each other as close as the distance between the parallel runways. This distance is frequently less than the normal conflict lateral separation threshold for that area type. Using the normal threshold for these aircraft could generate false alerts. To avoid this situation, special parameters are used for the conflict lateral...
separation threshold if the aircraft are classified as parallel. Two aircraft are classified as parallel if they satisfy two conditions:

1. The aircraft pair is in a Type 1 or Type 2 area which has parallel runways
2. The aircraft are parallel absolute. They have lines of flight which are parallel to the runway, and parallel to each other within CA\_COSQ (SP = cos 8 degrees)

If the aircraft are parallel, then C\_APAT\_PDS is used in place of parameters LATQ and PARQ. The value of C\_APAT\_PDS is a function of the distance between the parallel runways. Its nominal value is 70% of the distance between parallel runways.

**FUNCTION: Parallel Lateral**

**Imminence**

[PR.17] If an aircraft is determined to be in conflict, it must be determined if the conflict is imminent. An imminent conflict must satisfy the following conditions:

1. The conflict is not of aircraft making parallel landing or departures or parallel runways
2. The combined airport area type is not Type 1
3. LMC < CA\_LMDIMQ (SP = 0.5 NM)
4. TOMA < CA\_TOMIMQ (SP = 25 sec)
5. TOCA < CA\_TOCIMQ (SP = 25 sec)

**FUNCTION: Imminence Check**

If the aircraft were previously found to be slow closing laterally, then the quantities LMD and TOMA are undefined. In this case, conditions 3 and 4 are replaced by condition 6:

6. CLS < CA\_LMDIMQ (SP = 0.5 NM)

If the aircraft were previously found to be slow closing in altitude, then the quantity TOCA is undefined. In this case, condition 5 is replaced by condition 7:

7. |ΔZ| < CA\_DZIMQ (SP = 275 ft)

A LINCON conflict is ordinarily declared if the aircraft pair is determined to be in conflict for at least three out of the last five scans. If the conflict is imminent, the conflict is declared immediately, regardless of the check.
**Maneuver Conflict Detector (MANCON)**

[PR.18] MANCON detects potentially hazardous situations between aircraft when either or both of the aircraft are maneuvering horizontally. In these situations, LINCON cannot detect hazardous conditions quickly enough, because it would still be using old aircraft heading information. MANCON will predict potential conflicts based on each aircraft’s maneuver indicator.

[PR.19] Because the turn indicator applies for horizontal maneuvers only, MANCON tests apply only to those aircraft pairs whose vertical separation and horizontal separation are below their respective predefined thresholds (otherwise, they are too far apart for turning to matter). If, based on the relative positions of the aircraft pair, one aircraft is determined to be turning into the other, then the pair is considered to be in MANCON alert. [→PR.20]

**DESIGN PRINCIPLE**

*Source: NAS-MD-632*

[PR.20] The purpose of the Module for Maneuver and Maneuver Sensitive Algorithm (MFMAMS) is to determine if a conflict situation exists between a pair of aircraft utilizing the turn indication data provided by the Ground Plane Tracker [←PR.19]. It is the intent of MFMAMS to provide additional conflict warning time for aircraft making a lateral turn. An aircraft pair must satisfy the following conditions if it is to be considered by MFMAMS:

1. The difference in altitude $|\Delta Z|$ is less than CA_MFSALQ (SP = 400 ft)
2. The lateral distance between the aircraft is less than airport area type dependent system parameter CA_T3MFSEPB for Type 3, CA_T2MFSEPB for Type 2, and CA_T1MFSEPB for Type 1. The value of CA_T3MFSEPB is 2 miles.

[↓FUNCTION: MANCON check]

[PR.21] If these conditions are satisfied, each aircraft of the pair is processed to see if it is turning into the other aircraft [↓FUNCTION: Turning Into Each Other]. This is the case if any of the conditions are true:

1. The aircraft are converging (the quantity $\Delta X \Delta \dot{X} + \Delta Y \Delta \dot{Y}$ is negative [↓FUNCTION: Converging]
2. The angle formed by the two aircraft velocity vectors is less than or equal to 90 degrees. That is, the quantity $\dot{X}_1 \dot{X}_2 + \dot{Y}_1 \dot{Y}_2$ is positive
3. The angle formed by one aircraft’s velocity vector and the vector from this aircraft’s position to the other aircraft’s position is less than or equal to 90 degrees. Mathematically, this means that $\Delta X \dot{X}_1 + \Delta Y \dot{Y}_1$ is positive if processing aircraft 1, or $-(\Delta X \dot{X}_2 + \Delta Y \dot{Y}_2)$ is positive if processing aircraft 2.

[↓FUNCTION: Acute Velocity-Position Vector]

An MFMAMS conflict is then declared if this aircraft is turning in the direction of the other aircraft. This situation exists if the aircraft is laterally turning such that the angle
formed by the velocity vector and the position vector is getting smaller. In the following figure, this condition would be satisfied if aircraft 1 were turning left or aircraft 2 were turning right. [\textbf{FUNCTION: MANCON check}]
The check is made as follows. For aircraft 1, it must be known whether $V_1$ is on the right side of $\Delta \vec{P}$ (in a clockwise direction) or on the left side of $\Delta \vec{P}$ (in a counterclockwise direction). This is determined by evaluating the cross-product of the vectors $\Delta \vec{P}$ and $V_1$:

$$\Delta \vec{P} \times V_1 = |\Delta \vec{P}| |V_1| \sin \Theta_1 = \Delta X Y_1 - \Delta Y X_1$$

This quantity will be negative if $V_1$ is to the right of $\Delta \vec{P}$ and will be positive if $V_1$ is to the left of $\Delta \vec{P}$. Thus, an MFMAMS conflict exists if either of the following turn conditions is true:

1. $\Delta \vec{P} \times V_1$ is negative and aircraft 1 is turning left
2. $\Delta \vec{P} \times V_1$ is positive and aircraft 1 is turning right

An aircraft’s turning status is denoted by its turn indication field. If the aircraft’s turn indication field has either the right turn bit set or the left turn bit set. Thus, an aircraft is turning right if its right turn bit is set or if its left turn bit is not set (or both). As a result, condition 2 above can be reworded as:

2. $\Delta \vec{P} \times V_1$ is positive and aircraft 1’s left turn bit is not set

An analogous application of this test will determine whether aircraft 2 is in conflict. If either of the aircraft in the pair is in MFMAMS conflict, then the pair is considered to be in MFMAMS conflict.
**Proximity Conflict Detector (PROCON)**

[PR.22] PROCON detects potential conflicts between tracks based on current relative configuration, with minimal dependence on velocity data.

*Assumption: This function is intended for dense traffic areas, where algorithms that use projection on velocity tend to generate nuisance alarms.*

[PR.23] PROCON detects conflicts based on proximity, and thus PROCON checks apply only to aircraft pairs that are separated vertically and laterally by their respective predefined threshold values [→PR.25]. Aircraft pairs that are not making parallel landings on parallel runways are in PROCON conflict if the distance between them is less than predefined threshold value, or if they are fairly close and converging too quickly (this is the only check that involves velocity data). [→PR.27]

[PR.24] If the aircraft are making parallel landing on parallel runways, then they are in PROCON conflict if the distance between the aircraft is too close relative to the distance between the parallel runways. [→PR.26]

**DESIGN PRINCIPLE**

*Source: NAS-MD-632*

The purpose of the Proximity Conflict algorithm (PROCON) is to determine if a conflict situation exists between a pair of aircraft while being less dependent on velocity data. PROCON will provide a Conflict Alert function in areas where traffic is dense. Using algorithms which use projections in such areas can generate excessive false alerts.

[PR.25] An aircraft pair must satisfy the following conditions if it is to be considered by PROCON:

1. Their difference in altitude is less than CA_DZP1Q (SP = 375 ft), or their difference in altitude is less than CA_DZP2Q (SP = 650 ft) and they are converging in altitude by more than CA_ACPQ (SP = -3 ft/sec)
2. The lateral distance between the aircraft is less than CA_PMCQ (SP = 5 nm)
[→PR.23] [↓FUNCTION: PROCON check]

[PR.26] If the aircraft are making parallel landing on parallel runways, they are declared in conflict if the distance between them is less than C_APAT_PDS [→PR.24]. The nominal value of C_APAT_PDS is 70% of the distance between the parallel runways. [↓FUNCTION: Parallel Absolute] [↓FUNCTION: PROCON check]

[PR.27] If the aircraft are NOT making parallel landings on parallel runways, they are declared in conflict if either of the following conditions is satisfied:

1. The distance between the aircraft is less than airport area dependent system parameter CA_T3PTSQ for Type 3, CA_T2PTSQ for Type 2, and CA_T1PTSQ for Type 1. The value of CA_T3PTSQ is 0.75 nm.
2. The distance between the aircraft is less than airport area dependent system parameter CA_3PTH01 for Type 3, CA_2PTH01 for Type 2, and CA_1PTH01 for Type 1, AND the aircraft are converging by more than airport area dependent...
system parameter $\frac{CA_{T3EDO}}{\sqrt{12}}$ for Type 3, $\frac{CA_{T2EDO}}{\sqrt{12}}$ for Type 2, and $\frac{CA_{T1EDO}}{\sqrt{12}}$ for Type 1. The value of CA_T3PTHO1 is 1 nautical mile, and the value of CA_T3EDO is 120 knots.

[PR.23] [FUNCTION: PROCON check]

Convergence is defined as the instantaneous rate of change of the lateral distance between the aircraft. For linear aircraft with non-changing speeds, convergence is a variable quantity that is negative when the aircraft are converging, approaching zero when the aircraft reach their point of closest approach, and positive when the aircraft are diverging. If the distance between the aircraft is $\Delta P$, convergence is $\Delta \dot{P}$, the time derivative of $\Delta P$. The formulas for convergence are derived as follows:

$$\Delta P = \sqrt{\Delta X^2 + \Delta Y^2}$$

$$\Delta \dot{P} = \frac{d(\Delta P)}{dt} = \frac{\Delta X \dot{X} + \Delta Y \dot{Y}}{\sqrt{\Delta X^2 + \Delta Y^2}}$$
Prediction Maneuver Conflict Detector (MANCONP)

[PR.29] MANCONP detects potential conflicts between tracks that are maneuvering horizontally and possibly vertically. MANCOMP uses position and altitude change rates to predict conflicts. Instead of using horizontal velocities from the tracker that usually lag the real velocities in a maneuver, MANCONP will derive its own relative horizontal velocities using the tracker’s smoothed positions for the candidate aircraft pair. [→PR.33]

[PR.30] Aircraft pairs can potentially be in MANCONP conflict only if they meet the following criteria:
1. the aircraft are converging vertically, and their vertical separation is less than the predefined threshold
2. the aircraft are converging laterally and at a fast enough closing rate, and are separated laterally by less than the predefined threshold [↓FUNCTION: Converging] [↓FUNCTION: Convergence Rate] [↓FUNCTION: Acute Velocity Vectors] [↓FUNCTION: Acute Velocity-Position Vectors] [↓FUNCTION: MANCONP check]

[PR.31] MANCONP calculates the approach speed of the pair of tracks, which is defined as the rate at which the distance between the two aircraft changes. When calculating the approach speed, the positions of the targets have to be synchronized so that the positions at the same time reference are used (which usually means that one track will use its tracked smoothed position, and the other track will use a position interpolated to the time of update of the first track).

[PR.32] For valid aircraft pairs, MANCONP calculates three time intervals during which the pair is predicted to be in horizontal or vertical conflict:
- Slowest Horizontal Approach – the interval bounded by the earliest and latest time the pair is in horizontal conflict if flying at the approach speed.
- Fastest Horizontal Approach – the interval bounded by the earliest and latest time the pair is in horizontal conflict if flying head-on at the maximum possible closing speed (sum of tracker’s smoothed speed for each aircraft)
- Vertical Approach – the interval bounded by the earliest and latest time the pair is in vertical conflict if flying with the tracker’s smoothed z-velocities
[↓FUNCTION: MANCONP check]

If there is an overlap with all three intervals, and the start of the interval is later than the current time but earlier than the current time plus a predefined look ahead time, then the aircraft pair is considered to be in MANCONP conflict.
DESIGN PRINCIPLE

Source: NAS-MD-632

[PR.33] The purpose of the Proximity Ordinary Predictor Logic (POPL) is to provide a Conflict Alert algorithm to supplement the look ahead warning capability of LINCON. It is the intent of POPL to provide look ahead warning while being less dependent on aircraft velocity data. An aircraft must satisfy the following preliminary conditions if it is to be considered by POPL:

1. Since POPL is called from PROCON, the PROCON preliminary conditions must be satisfied
2. \( \frac{\Delta P}{R} < CA_{-TiTAUQ} \)
3. \( \Delta \dot{P} < -CA_{-TiEDO} \)

\( \Delta P = \) Current Lateral Separation = \( \sqrt{\Delta X^2 + \Delta Y^2} \)
\( \Delta \dot{P} = \) Convergence as defined in PROCON
\( R = \) Relative Speed = \( \sqrt{\Delta \dot{X}^2 + \Delta \dot{Y}^2} \)

\( CA_{-TiTAUQ} \) is an airport area dependent time parameter, where \( i = \) Type value (either 1, 2, or 3). Its Type 3 value is 45 seconds.

\( CA_{-TiEDO} \) is an airport area dependent time parameter, where \( i = \) Type value (either 1, 2, or 3). Its Type 3 value is 120 knots.

[←PR.29]

[PR.34] If the preliminary conditions are satisfied, POPL declares the aircraft pair in conflict if the Lateral Miss Distance (LMD) is less than \( CA_{-TiLMDQ} \) (SP).

\( CA_{-TiLMDQ} \) is an airport area dependent parameter whose Type 3 value is 0.8 nm. This functionality is essentially identical to LINCON, but uses a different method to calculate the LMD. POPL attempts to minimize its dependence on the velocity data of the two aircraft.

Most of the POPL logic is used to calculate a quantity \( \frac{da}{dt} \) which is needed to calculate LMD. This quantity is defined as follows:
Let aircraft A be at position $A_1$ and have velocity vector $V_1$.
Let aircraft A be at position $B_1$ and have velocity vector $V_2$.

The angle $a_1$ of the position of aircraft B relative to aircraft A is called the position angle. At some time $\Delta t$ later, let aircraft A be at position $A_2$ and aircraft B be at position $B_2$. The new position angle is $a_2$ (in general, $a_2$ will be different from $a_1$). If both aircraft have a linear path with non-changing speeds, they are on a collision course if and only if the position angle does not change. The quantity $\frac{da}{dt}$ is the instantaneous rate of change of the angle with respect to time.

The quantity $\frac{da}{dt}$ could be approximated by calculating $\frac{a_2 - a_1}{\Delta t}$, but this would necessitate the use of an inverse tangent or inverse sine routine for each scan. To avoid this, a function $F(a)$ is defined:

- $F(a) = -2 - \cot a$ for $-\pi/2 \leq a < -\pi/4$
- $F(a) = \tan a$ for $-\pi/4 \leq a < \pi/4$
- $F(a) = 2 - \cot a$ for $\pi/4 \leq a < \pi/2$
Because $F(a)$ is implicitly defined as a function of $t$, the chain rule is used to calculate $\frac{dF}{dt}$.

\[
\frac{dF}{dt} = \frac{dF}{da} \cdot \frac{da}{dt}
\]

For a given aircraft pair, $F(a)$ is calculated each scan. Let $\Delta F$ be the difference between $F(a)$ for the current scan and $F(a)$ for the previous scan, and let $\Delta t$ be the corresponding difference in time. Then $\frac{dF}{dt} \approx \frac{\Delta F}{\Delta t}$.

We notice that $F(a)$ is closely approximated by a linear function, and make the assumption that $F(a)$ is linear. Then, $\frac{dF}{da}$ is the slope of the $F(a)$ curve, and we approximate its value by:

\[
\frac{dF}{da} = \Delta F/\Delta a = \frac{2}{\pi/2} \approx 1.27
\]

Substituting the approximations for $\frac{dF}{dt}$ and $\frac{dF}{da}$ into equation 1 yields:

\[
\frac{da}{dt} = \frac{\Delta F}{1.27 \Delta t} \quad (2)
\]

Let $\tan \alpha = \frac{\Delta Y}{\Delta X}$

Taking the time derivative of both sides: $(\sec^2 \alpha) \frac{da}{dt} = \frac{\Delta X \Delta \dot{Y} - \Delta Y \Delta \dot{X}}{\Delta X^2}$

Setting $\sec^2 \alpha = \frac{\Delta P^2}{\Delta X^2}$: $(\Delta P^2) \frac{da}{dt} = \Delta X \Delta \dot{Y} - \Delta Y \Delta \dot{X} \quad (3)$

The standard formula for LMD is: $LMD = \frac{\Delta X \Delta \dot{Y} - \Delta Y \Delta \dot{X}}{R}$

Substituting in equation 3, we get: $LMD = \frac{-\Delta P^2}{R} \sqrt{\frac{\Delta a}{dt}} \quad (4)$

Substituting in equation 2, we get the equations used by POPL to calculate LMD:
\[ LMD = \frac{\Delta P^2}{R} \cdot \frac{\Delta F}{1.27 \Delta t} \]

\[ LMD = \frac{\Delta F \cdot \Delta P^2}{1.27 \cdot \Delta t \cdot R} \]

There is a caveat in the calculation of \( \Delta F \) that should be explained. The function \( F(a) \) possesses a discontinuity as the value of "a" crosses the value \( \pi/2 \) and \(-\pi/2\). This discontinuity is handled by special logic, which is explained through an example.

Let the position angle "a" during a scan be some value less than 90 degrees such that \( F(a) = 1.9 \). One scan later, let the position angle change only slightly to some value slightly greater than 90 degrees. According to the normal definition of "a", "a" would assume a value slightly larger than 90 degrees, and \( F(a) \) would have a value of approximately \(-1.9\). Then:

\[ \Delta F = |F(a) - \text{previous } F(a)| = |-1.9 - 1.9| = 3.8 \]

According to our example, however, the value of \( F(a) \) should have changed by only 0.2. Thus, \( \Delta F \) must be corrected by subtracting it from 4. This logic assumes that the actual change in the value of \( F(a) \) from one scan to the next does not exceed 2. If it does, then it is assumed that a crossover of the discontinuity has occurred.

**Generating Conflict Alerts**

[PR.35] The `GenerateConflictAlerts` subfunction generates alert messages for all track pairs with a conflict alert condition, and maintains a database of tracks that are suppressed for conflict alert based on controller requests. If any event occurs that requires that the display of a conflict alert begin or end, the module will send a message to the SDDs (Situational Data Displays) and the data recording facility (DRF). [†FR.8] [†EO.1]

[PR.36] `GenerateConflictAlerts` will be active regardless of the CA on-off switch and the MCI on-off switch states. Even if conflict alert processing is idle or the `GenerateConflictAlerts` is running in the standby RDPS, processing will continue. In this case, however, no alert messages will be sent to the SDD’s.

[PR.37] To avoid generating false alerts, `GenerateConflictAlerts` sends alert messages only when a conflict violation is persistent [†C.5]. A conflict violation is persistent if the same conflict detection test has raised an alert at least three times in the last five scans. If the current or predicted conflict is persistent, the module will generate an MSAW-CA alert message identifying the tracks involved in the violation and send the message to all SDDs and DRFs. The alert message will contain the alert start time, time to violation, time to non-violation, imminent indicator, coordinates, and altitude. The
alerts message only runs for a fixed time duration (CA_alert_duration), and will be terminated after the duration. [\texttt{FUNCTION: Persistent LINCON alert}] [\texttt{FUNCTION: Persistent MANCON alert}] [\texttt{FUNCTION: Persistent PROCON alert}] [\texttt{FUNCTION: Persistent MANCONP alert}]

[PR.39] All alert messages contain identical conflict alert data so that all SDDs can display identical conflict alerts. If the alert is newly eligible for display, then the message will indicate the start of a warning. If the condition is a new one, the message will indicate the start of a condition. If the track is associated to a suspended flight plan, the flight plan will be unsuspended.
Controller Intervention

[PR.40] CA/MCI can also receive and process requests made by the controller through the SDD. These requests are used to suppress or enable conflict alerts for individual aircraft, aircraft pairs, or all aircraft in the controller’s airspace. The various commands and their resulting actions are described below: [↓ OUTPUT MESSAGE: Controller Input Acknowledgement]

**Aircraft Inhibit**

This message toggles the inhibit status for a particular aircraft in the controller’s airspace.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>VALIDATION</th>
<th>CONTEXT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specified flight is not displaying a CA alert</td>
<td>The aircraft ID uniquely identifies a flight and the selected flight is owned by the requesting TCP</td>
<td>Flight is not currently individually CA suppressed</td>
<td>Update Flight Plan to indicate flight is individually CA suppressed and terminate any pair-wise CA inhibits in which this flight is a partner</td>
</tr>
<tr>
<td>Specified flight is displaying a CA alert</td>
<td>The aircraft ID uniquely identifies a flight and the selected flight is owned by the requesting TCP</td>
<td>Conflict is simple with other flight owned by requesting TCP</td>
<td>Update Flight Plan file to indicate that both the selected flight and the conflicting flight are individually CA suppressed and terminate any pair-wise CA inhibits in which either flight is a partner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conflict is non-simple or conflicting flight is MCI or not owned by requesting TCP</td>
<td>Update Flight Plan to indicate flight is individually CA suppressed and terminate any pair-wise CA inhibits in which this flight is a partner</td>
</tr>
</tbody>
</table>

57
**Aircraft Pair Inhibit**

This message toggles the inhibit status for a particular aircraft pair in the controller’s airspace.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>VALIDATION</th>
<th>CONTEXT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message identifies a single track and that track is displaying a CA alert</td>
<td>Specified an/or co-conflicting flight is owned by requesting TCP and conflict is simple</td>
<td>Any</td>
<td>Update the Flight Plan such that a pair-wise CA alert conflict is simple</td>
</tr>
<tr>
<td>Message Identifies two tracks</td>
<td>One or both tracks are owned by the requesting TCP and neither track is individually CA suppressed</td>
<td>Tracks are not pair-wise CA inhibited with each other</td>
<td>Update the Flight Plan file such that the indication of a pair-wise CA inhibit exists between the two tracks</td>
</tr>
<tr>
<td>Message identifies a single track and that track is not displaying a CA alert</td>
<td>Specified track is involved in a single pair-wise inhibit and partner track is pair-wise inhibited only with specified track and one or both tracks are owned by requesting TCP</td>
<td>Any</td>
<td>Update the Flight Plan file such that the indication of a pair-wise CA inhibit between the two is removed</td>
</tr>
</tbody>
</table>

**Volume Inhibit**

This message toggles the inhibit status for all aircraft in the controller’s airspace.
A: HMI → CA/MCI
   CA_on_off
   MCI_on_off
   MSAW_CA_suppression_request

B: CA/MCI → HMI
   MSAW_CA_alert
   Operator_controller_entry_error

C: Radar Data Processing System → CA/MCI
   Adaptation_Data

D: Radar Data Processing Tracker → CA/MCI
   Track_data

E: Operational Sites → CA/MCI
   MSAW_CA_airport_data
OUTPUT MESSAGE:
Conflict Alert Message

**Destination:** Situation Data Display; Data Recording Facility

**Acceptable Values:** \{on, imminent\}

**Timing Behavior:**
- **Initiation delay:** 0 milliseconds
- **Completion Deadline:**
- **Exception-handling:**

**Feedback Information:**
- **Variables:** alert start time, time to violation, time to non-violation, imminent indicator, coordinates of aircraft1 and aircraft2, and altitude of aircraft1 and aircraft2
- **Values:** \{ok\}
- **Relationship:** should be on until controller acknowledgement
- **Min. time (latency):** 10 seconds
- **Max. time:** none
- **Exception handling:**
- **Description:** has a fixed duration time of 10 seconds, a pre-defined value that is set to be long enough so that the controller notices, but short enough to minimize noise
- **Comments:** alert messages should contain identical conflict alert data so that all SDD’s can display identical conflict alerts

**References:** MSAW_CA_alert

| = ON |
| Persistent LINCON alert | T |
| Persistent MANCON alert | T |
| Persistent PROCON alert | T |
| Persistent MANCONP alert | T |
| Aircraft1 Conflict Detection = On | T T T T T |
| Aircraft2 Conflict Detection = On | T T T T T |
| Aircraft Pair Conflict Detection = On | T T T T T |
| Volume Conflict Detection = On | T T T T T |
= IMMINENT

<table>
<thead>
<tr>
<th>Persistent LINCON alert</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent Imminence alert</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1 Conflict Detection = On</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft2 Conflict Detection = On</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft Pair Conflict Detection = On</td>
<td>T</td>
</tr>
<tr>
<td>Volume Conflict Detection = On</td>
<td>T</td>
</tr>
</tbody>
</table>
OUTPUT MESSAGE:
Controller Input Acknowledgement

**Destination:** Situation Data Display; Data Recording Facility

**Acceptable Values:** {on}

**Timing Behavior:**
- **Initiation delay:** 0 milliseconds
- **Completion Deadline:**
- **Exception-handling:**

**Feedback Information:**
- **Variables:** Controller acknowledgement
- **Values:** {ok}
- **Relationship:** should be on until controller acknowledgement
- **Min. time (latency):** 10 seconds
- **Max. time:** none
- **Exception handling:**

**Reversed by:**
- **Description:** acknowledges controller request to toggle CA/MCI inhibit status

**Comments:** returns a generic “success” or “error” message, regardless of input request type

**References:** [SUPR.40]

---

<table>
<thead>
<tr>
<th>= SUCCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSAW_CA_suppression_request = Aircraft_Inhibit_Toggle</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Aircraft_Pair_Inhibit_Toggle</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Volume_Inhibit_Toggle</td>
</tr>
<tr>
<td>Validation(aircraftID) = True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>= ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSAW_CA_suppression_request = Aircraft_Inhibit_Toggle</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Aircraft_Pair_Inhibit_Toggle</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Volume_Inhibit_Toggle</td>
</tr>
<tr>
<td>Validation(aircraftID) = True</td>
</tr>
</tbody>
</table>
OUTPUT MESSAGE:
Terminate Warning Message

**Destination:** Situation Data Display; Data Recording Facility

**Acceptable Values:** \{terminate\}

**Timing Behavior:**
- **Initiation delay:**
- **Completion Deadline:**
- **Exception-handling:**

**Feedback Information:**
- **Variables:** MSAW_CA_alert, alert start time, time to violation, time to non-violation, imminent indicator, coordinates of aircraft1 and aircraft2, and altitude of aircraft1 and aircraft2
- **Values:** \{ok\}
- **Relationship:**
- **Min. time (latency):**
- **Max. time:** none
- **Exception handling:**
- **Reversed by:** None

**Description:** When an alert for an aircraft pair is first raised, a timer is started. If the alert is over 10 seconds old, CA/MCI will terminate the warning message.

**Comments:** terminate alert messages should contain identical conflict alert data so that all SDD’s can display identical conflict alerts

**References:**

| Conflict Alert Message = On | T |
| Message Timer > CA_alert_duration | T |
**CONTROL MODE**

**Obsolescence:**

**Exception handling:**

**Description:** determines whether or not CA and MCI alerts are currently inhibited

**Comments:**

**References:**

**Appears in:**

= CA on, MCI on

<table>
<thead>
<tr>
<th>Startup</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>CA_on_off = True</td>
<td>F T</td>
</tr>
<tr>
<td>MCI_on_off = True</td>
<td>F</td>
</tr>
<tr>
<td>PREVIOUS VALUE = CA on, MCI on</td>
<td>T</td>
</tr>
</tbody>
</table>

= CA on, MCI off

<table>
<thead>
<tr>
<th>Startup</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>F F</td>
</tr>
<tr>
<td>CA_on_off = True</td>
<td>F F</td>
</tr>
<tr>
<td>MCI_on_off = True</td>
<td>F T</td>
</tr>
<tr>
<td>PREVIOUS VALUE = CA on, MCI on</td>
<td>F T</td>
</tr>
<tr>
<td>PREVIOUS VALUE = CA on, MCI off</td>
<td>T F</td>
</tr>
<tr>
<td>PREVIOUS VALUE = CA off, MCI off</td>
<td>F F</td>
</tr>
</tbody>
</table>

= CA off, MCI off

<table>
<thead>
<tr>
<th>Startup</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>F F</td>
</tr>
<tr>
<td>CA_on_off = True</td>
<td>F T</td>
</tr>
<tr>
<td>MCI_on_off = True</td>
<td>F F</td>
</tr>
<tr>
<td>PREVIOUS VALUE = CA on, MCI on</td>
<td>F T</td>
</tr>
<tr>
<td>PREVIOUS VALUE = CA on, MCI off</td>
<td>F F</td>
</tr>
<tr>
<td>PREVIOUS VALUE = CA off, MCI off</td>
<td>T F</td>
</tr>
</tbody>
</table>
**STATE VARIABLE:**  
*Aircraft_Conflict_Detection*

**Obsolescence:**

**Exception handling:**

**Description:** determines whether or not the aircraft is suppressed for CA/MCI alerts

**References:**

**Appears in:**

= **UNKNOWN**

<table>
<thead>
<tr>
<th>Startup</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
</tbody>
</table>

PREVIOUS VALUE = Unknown  
T

= **ON**

<table>
<thead>
<tr>
<th>Startup</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Aircraft_Inhibit_Toggle</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft.CA = On</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Aircraft.Catype = Simple</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation(aircraft.ID) = Ok</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

PREVIOUS VALUE = On  
T | F | F

= **OFF**

<table>
<thead>
<tr>
<th>Startup</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Aircraft_Inhibit_Toggle</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft.CA = On</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Aircraft.Catype = Simple</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Validation(aircraft.ID) = Ok</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

PREVIOUS VALUE = On  
F | T | F
**STATE VARIABLE:**
**Aircraft_Pair_Conflict_Detection**

**Obsolescence:**
**Exception handling:**

**Description:** determines whether or not an aircraft pair (specified by an aircraft and its partner) is suppressed for CA/MCI alerts

**Comments:**

**References:**

**Appears in:**

= **UNKNOWN**

<table>
<thead>
<tr>
<th>Startup</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>PREVIOUS VALUE = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

= **ON**

<table>
<thead>
<tr>
<th>Startup</th>
<th>F F F F F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>F F F F F</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Aircraft_Pair_Inhibit_Toggle</td>
<td>F T T T T T</td>
</tr>
<tr>
<td>Aircraft.CA = On</td>
<td>F</td>
</tr>
<tr>
<td>Aircraft.Aircraft_Conflict_Detection = On</td>
<td>F</td>
</tr>
<tr>
<td>Aircraft.partner.CA = On</td>
<td>F</td>
</tr>
<tr>
<td>Aircraft.partner.Aircraft_Conflict_Detection = On</td>
<td>F</td>
</tr>
<tr>
<td>Aircraft.CAtype = Simple</td>
<td>F</td>
</tr>
<tr>
<td>Validation(aircraft.ID) = Ok</td>
<td>F T T</td>
</tr>
<tr>
<td>PREVIOUS VALUE = On</td>
<td>T T T F F</td>
</tr>
</tbody>
</table>

= **OFF**

<table>
<thead>
<tr>
<th>Startup</th>
<th>F F F F F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>F F F F F</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Aircraft_Pair_Inhibit_Toggle</td>
<td>F T T T T T</td>
</tr>
<tr>
<td>Aircraft.CA = On</td>
<td>T F T</td>
</tr>
<tr>
<td>Aircraft.Aircraft_Conflict_Detection = On</td>
<td>F</td>
</tr>
<tr>
<td>Aircraft.partner.CA = On</td>
<td>F T T</td>
</tr>
<tr>
<td>Aircraft.partner.Aircraft_Conflict_Detection = On</td>
<td>F</td>
</tr>
<tr>
<td>Aircraft.CAtype = Simple</td>
<td>F T T T</td>
</tr>
<tr>
<td>Validation(aircraft.ID) = Ok</td>
<td>F T T T T</td>
</tr>
<tr>
<td>PREVIOUS VALUE = On</td>
<td>F F F F F T</td>
</tr>
</tbody>
</table>
**STATE VARIABLE:**
*Volume_Conflict_Detection*

**Obsolescence:**

**Exception handling:**

**Description:** determines whether or not the volume in the particular controller’s airspace is inhibited for CA/MCI alerts. Volume conflict suppression disables CA/MCI alerts for Type_I or Type_II volumes for a specific airport, or globally for all airports

**Comments:**

**References:**

**Appears in:**

<table>
<thead>
<tr>
<th>= UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
</tr>
<tr>
<td>PREVIOUS VALUE = Unknown</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>= ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Volume_Inhibit_Toggle</td>
</tr>
<tr>
<td>PREVIOUS VALUE = On</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>= OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
</tr>
<tr>
<td>MSAW_CA_suppression_request = Volume_Inhibit_Toggle</td>
</tr>
<tr>
<td>PREVIOUS VALUE = On</td>
</tr>
</tbody>
</table>
**FUNCTION:**

*Acute Velocity-Position Vectors (aircraft1, aircraft2)*

**Obsolescence:**

**Exception handling:**

**Description:** a metric used to determine if aircraft1 is turning into aircraft2.

This condition is true of the angle formed by aircraft1's velocity vector and aircraft2's position vector relative to aircraft1 is less than or equal to 90 degrees

**Comments:**

**References:** [↑PR.30]

**Appears in:** MANCONP check

### = UNKNOWN

<table>
<thead>
<tr>
<th>Startup</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1.vel_x = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft2.vel_y = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1.vel_x = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft2.vel_y = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Relative_pos (aircraft1, aircraft2) = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

### = TRUE

<table>
<thead>
<tr>
<th>Startup</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>(Relative_pos (aircraft1, aircraft2).x * Aircraft1.vel_x) + (Relative_pos (aircraft1, aircraft2).y * Aircraft1.vel_y) &gt; 0</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>(Relative_pos (aircraft1, aircraft2).x * Aircraft2.vel_x) + (Relative_pos (aircraft1, aircraft2).y * Aircraft2.vel_y) &lt; 0</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

### = FALSE

<table>
<thead>
<tr>
<th>Startup</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
</tr>
<tr>
<td>(Relative_pos (aircraft1, aircraft2).x * Aircraft1.vel_x) + (Relative_pos (aircraft1, aircraft2).y * Aircraft1.vel_y) &gt; 0</td>
<td>F</td>
</tr>
<tr>
<td>(Relative_pos (aircraft1, aircraft2).x * Aircraft2.vel_x) + (Relative_pos (aircraft1, aircraft2).y * Aircraft2.vel_y) &lt; 0</td>
<td>F</td>
</tr>
</tbody>
</table>
FUNCTION:
Acute Velocity Vectors (aircraft1, aircraft2)

Obsolescence:
Exception handling:
Description: a metric used to determine if aircraft1 is turning into aircraft2.
This condition is true of the angle formed by aircraft1’s velocity vector and
aircraft2’s velocity vector is less than or equal to 90 degrees

Comments:
References: [↑PR.30]
Appears in: MANCONP check

\[
\begin{array}{|c|c|c|}
\hline
\text{Startup} & \text{F} & \text{T} \\
\hline
\text{Controls.Reset = True} & \text{F} & \text{T} \\
\hline
\text{Aircraft1.vel_x = Unknown} & \text{T} & \\
\hline
\text{Aircraft2.vel_x = Unknown} & \text{T} & \\
\hline
\text{Aircraft1.vel_y = Unknown} & \text{T} & \\
\hline
\text{Aircraft2.vel_y = Unknown} & \text{T} & \\
\hline
\end{array}
\]

\[= \text{UNKNOWN}\]

\[= \text{TRUE}\]
\[
\begin{array}{|c|}
\hline
\text{Startup} & \text{F} \\
\hline
\text{Controls.Reset = True} & \text{F} \\
\hline
(Aircraft1.vel_x \times Aircraft2.vel_x) + (Aircraft1.vel_y \times Aircraft2.vel_y) > 0 & \text{T} \\
\hline
\end{array}
\]

\[= \text{FALSE}\]
\[
\begin{array}{|c|}
\hline
\text{Startup} & \text{F} \\
\hline
\text{Controls.Reset = True} & \text{F} \\
\hline
(Aircraft1.vel_x \times Aircraft2.vel_x) + (Aircraft1.vel_y \times Aircraft2.vel_y) > 0 & \text{F} \\
\hline
\end{array}
\]
FUNCTION:
Aircraft Volume Type (aircraft, RCB_data)

Obsolescence:

Exception handling:

Description: To determine the minimum separation standards for an aircraft pair, the pair must be assigned a volume type. A pair’s volume type (either Type_I, Type_II, and Type_III) is determined by the volume type of the individual aircraft in the pair. An aircraft is assigned a volume type by comparing its location coordinates (x-coordinate, y-coordinate, and altitude) with a list of runway capture boxes (RCB’s) for a match.

Comments:

References: [↑PR.30]

Appears in:

Let \( V = \) set of RCB_i’s in RCB_data for which the following is true:
1. \( \text{aircraft.pos_z} < \text{RCB_data[RCB_i].RCB_ceiling_altitude} \)
2. \( \text{aircraft.pos_x and aircraft.pos_y in RCB_data[RCB_i].RCB_lat_area} \)
3. \( |\text{aircraft.heading - RCB_data[RCB_i].RCB_heading}| < \text{trk_hdrng_dev} \)
4. If \( V = \emptyset \) then Aircraft Volume Type = Type_III

Let \( \text{final_RCB} = \text{RCB_i} \) in \( V \) that minimizes \( |\text{aircraft.heading - RCB_data[RCB_i].RCB_heading}| \)

If \( \text{aircraft.pos_x and aircraft.pos_y in RCB_data[RCB_i].Type_II_area} \) then Aircraft Volume Type = Type_II

If \( \text{aircraft.pos_x and aircraft.pos_y in RCB_data[RCB_i].Type_I_area} \) then Aircraft Volume Type = Type_I
FUNCTION:
Converging (aircraft1, aircraft2)

Obsolescence:
Exception handling:
Description:
Comments:
References: [↑PR.30]
Appears in: MANCONP check

= UNKNOWN

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>T</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>Relative_pos(aircraft1, aircraft2) = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Relative_vel(aircraft1, aircraft2) = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

= TRUE

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>F</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
</tr>
<tr>
<td>Relative_pos(aircraft1, aircraft2).x * Relative_vel(aircraft1, aircraft2).x + Relative_pos(aircraft1, aircraft2).y * Relative_vel(aircraft1, aircraft2).y &lt; 0</td>
<td>T</td>
</tr>
</tbody>
</table>

= FALSE

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>F</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
</tr>
<tr>
<td>Relative_pos(aircraft1, aircraft2).x * Relative_vel(aircraft1, aircraft2).x + Relative_pos(aircraft1, aircraft2).y * Relative_vel(aircraft1, aircraft2).y &lt; 0</td>
<td>F</td>
</tr>
</tbody>
</table>
FUNCTION:
Convergence Rate (aircraft1, aircraft2)

Obsolescence:
Exception handling:
  Description: Convergence is the instantaneous rate of change of the lateral distance between a pair of aircraft [↑ PR.28]. For aircraft with non-changing speeds, convergence is a variable quantity that is negative when the aircraft are converging, approaching zero when the aircraft reach their point of closest approach, and positive when the aircraft are diverging.

Comments:
References: [↑ PR.28]
Appears in: PROCON check (FUNCTION)

Convergence Rate = ((Relative_pos.x * Relative_vel.x) + (Relative_pos.y * Relative_vel.y)) / SQRT((Relative_pos.x)^2+(Relative_pos.y)^2)
FUNCTION:
Imminence Check

Obsolescence:
Exception handling:
Description: checks to see if an aircraft pair in LINCON alert is in an imminent
conflict situation
Comments:
References: [↑ PR.17]
Appears in: CONFLICT ALERT SIGNAL

= UNKNOWN

<table>
<thead>
<tr>
<th></th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>T</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>Separation_lateral = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Separation_vertical = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Relative_vel = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

= TRUE

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Relative_vel_x + Relative_vel_y &gt; MNRELQ</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>LMD &lt; LMDIMQ</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOMA &lt; TOMIMQ</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separation_lateral &lt; LMDIMQ</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separation_vertical &lt; DZIMQ</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOCA &lt; TOCIMQ</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

= FALSE

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Relative_vel_x + Relative_vel_y &gt; MNRELQ</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>LMD &lt; LMDIMQ</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOMA &lt; TOMIMQ</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separation_lateral &lt; LMDIMQ</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separation_vertical &lt; DZIMQ</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOCA &lt; TOCIMQ</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**FUNCTION:**
LINCON Check

**Obsolescence:**

**Exception handling:**

**Description:** checks for a conflict between an aircraft pair based on the projection of each aircraft’s current speed and heading

**Comments:**

**References:** [↑ PR.10] [↑ PR.11]

**Appears in:** CONFLICT ALERT SIGNAL

<table>
<thead>
<tr>
<th>= UNKNOWN</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>T</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>Separation_lateral = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Separation_vertical = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>= TRUE</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Parallel Altitude = True</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Separation_vertical &lt; TiZALQ</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel Lateral = True</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Separation_lateral &lt; TiLATQ</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 ≤ Time_of_conflict &lt; CA_lkhd_time</td>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**FUNCTION:**
**MANCON Check**

**Obsolescence:**

**Exception handling:**

**Description:** checks for potentially hazardous situations between a pair of aircraft when either or both are maneuvering horizontally

**Comments:**

**References:** [↑ PR.20] [↑ PR.21]

**Appears in:** CONFLICT ALERT SIGNAL

---

<table>
<thead>
<tr>
<th>= UNKNOWN</th>
<th>T</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Separation_lateral = Unknown</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Separation_vertical = Unknown</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>= TRUE</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Separation_lateral &lt; TiMFSEPB</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Separation_vertical &lt; MFZALQ</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Turning into each other = True</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>= FALSE</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Separation_lateral &lt; TiMFSEPB</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separation_vertical &lt; MFZALQ</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turning into each other = True</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FUNCTION:
MANCONP Check

Obsolescence:

Exception handling:

Description: checks for potential conflicts between an aircraft pair where one or both are maneuvering horizontally and possibly vertically

Comments: Δ is a design parameter that is initialized by the STARS controller

10.1.1.1 References: [↑ PR.30] [↑ PR.32]

Appears in: CONFLICT ALERT SIGNAL

| = UNKNOWN | T |  |
| Control.Reset = True | T |  |
| Separation_lateral = Unknown | T |  |
| Separation_vertical = Unknown | T |  |

| = TRUE | F |  |
| Control.Reset = True | F |  |
| Converging = True | T |  |
| Separation_lateral < TiLATQ + Δ | T |  |
| Approach_interval_overlap = True | T |  |
| Approach_interval_start > current_time | T |  |
| Approach_interval_start < current_time + lookahead | T |  |

| = FALSE | F | F | F | F | F |  |
| Control.Reset = True | F | F | F | F | F |  |
| Converging = True | F |  |
| Separation_lateral < TiLATQ + Δ | F |  |
| Approach_interval_overlap = True | T | T | T |  |
| Approach_interval_start > current_time | F | T | F |  |
| Approach_interval_start < current_time + lookahead | T | F | F |  |
FUNCTION:
Parallel Absolute

Obsolescence:
Exception handling:
Description: two aircraft are considered to be flying in parallel absolute if the following conditions are true:
   1. the heading difference between each aircraft’s line of flight and its approach runway is less than COS1Q
   2. the heading difference between the aircrafts' lines of flights is less than COS2Q
Comments: this check is performed when an aircraft pair is approaching parallel runways at the same airport
References: [↑ PR.26]
Appears in: PROCON check

= UNKNOWN

<table>
<thead>
<tr>
<th></th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>T</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>Line_of_flight(aircraft1) = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Line_of_flight(aircraft2) = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Runway_data(runway(aircraft1)) = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Runway_data(runway(aircraft2)) = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

= TRUE

<table>
<thead>
<tr>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>F</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
</tr>
<tr>
<td>Heading_diff (Line_of_flight(aircraft1), runway(aircraft1)) &lt; COS1Q</td>
<td>T</td>
</tr>
<tr>
<td>Heading_diff (Line_of_flight(aircraft2), runway(aircraft2)) &lt; COS1Q</td>
<td>T</td>
</tr>
<tr>
<td>Heading_diff (Line_of_flight(aircraft1), Line_of_flight(aircraft2)) &lt; COS2Q</td>
<td>T</td>
</tr>
</tbody>
</table>

= FALSE

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Heading_diff (Line_of_flight(aircraft1), runway(aircraft1)) &lt; COS1Q</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heading_diff (Line_of_flight(aircraft2), runway(aircraft2)) &lt; COS1Q</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heading_diff (Line_of_flight(aircraft1), Line_of_flight(aircraft2)) &lt; COS2Q</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**FUNCTION:**
Parallel Altitude

**Obsolescence:**

**Exception handling:**

**Description:** two aircraft are considered to be flying in parallel altitude if the absolute value of the difference of their vertical velocities is less than PARALQ

**Comments:**

**References:** [↑ PR.14]

**Appears in:** LINCON check

---

### = UNKNOWN

<table>
<thead>
<tr>
<th></th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1.vel.z = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft2.vel.z = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

### = TRUE

<table>
<thead>
<tr>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
</tr>
<tr>
<td>Aircraft1.vel.z &lt; Aircraft2.vel.z</td>
<td>T</td>
</tr>
</tbody>
</table>

### = FALSE

<table>
<thead>
<tr>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
</tr>
<tr>
<td>Aircraft1.vel.z &lt; Aircraft2.vel.z</td>
<td>F</td>
</tr>
</tbody>
</table>
FUNCTION:
Parallel Lateral

Obsolescence:

Exception handling:

Description: Let x_diff be the absolute value of the difference of the x-velocities of an aircraft pair. Let y_diff be the absolute value of the difference of the y-velocities of the same aircraft pair. Then the two aircraft are flying in parallel lateral if the sum of x_diff and y_diff is less than CMC_vel_dif_tol

Comments:

References: [↑ PR.15]

Appears in: LINCON check

<table>
<thead>
<tr>
<th>= UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
</tr>
<tr>
<td>Aircraft1.vel_x= Unknown</td>
</tr>
<tr>
<td>Aircraft2.vel_x = Unknown</td>
</tr>
<tr>
<td>Aircraft1.vel_y = Unknown</td>
</tr>
<tr>
<td>Aircraft1.vel_y = Unknown</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>= TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CMC_vel_dif_tol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>= FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
</tr>
<tr>
<td>Controls.Reset = True</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CMC_vel_dif_tol</td>
</tr>
</tbody>
</table>
FUNCTION:
Persistent LINCON alert

Obsolescence:

Exception handling:

Description: A LINCON alert is considered to be persistent if the LINCON test returned an alert at least three times in the last five scans. Persistent alerts are the only ones displayed by the Conflict Alert module.

Comments:

References: [↑ PR.38]

Appears in: CONFLICT ALERT MESSAGE (OUTPUT MESSAGE)

Persistent LINCON alert =
   True, if during the last 5 scans, 60+% registered LINCON alerts
   False, otherwise
FUNCTION:
Persistent MANCON alert

Obsolescence:

Exception handling:

Description: a MANCON alert is considered to be persistent if the MANCON test returned an alert at least three times in the last five scans. Persistent alerts are the only ones displayed by the Conflict Alert module.

Comments:

References: [↑ PR.38]

Appears in: CONFLICT ALERT MESSAGE (OUTPUT MESSAGE)

Persistent MANCON alert =
    True, if during the last 5 scans, 60+% registered MANCON alerts
    False, otherwise
FUNCTION:
Persistent MANCONP alert

Obsolescence:

Exception handling:

Description: a MANCONP alert is considered to be persistent if the MANCONP test returned an alert at least three times in the last five scans. Persistent alerts are the only ones displayed by the Conflict Alert module.

Comments:

References: [PR.38]

Appears in: CONFLICT ALERT MESSAGE (OUTPUT MESSAGE)

Persistent MANCONP alert =
   True, if during the last 5 scans, 60+% registered MANCONP alerts
   False, otherwise
FUNCTION:
Persistent PROCON alert

Obsolescence:

Exception handling:

Description: a PROCON alert is considered to be persistent if the PROCON test returned an alert at least three times in the last five scans. Persistent alerts are the only ones displayed by the Conflict Alert module.

Comments:

References: [↑ PR.38]

Appears in: CONFLICT ALERT MESSAGE (OUTPUT MESSAGE)

Persistent PROCON alert =
  True, if during the last 5 scans, 60+% registered PROCON alerts
  False, otherwise
FUNCTION:
PROCON Check

Obsolescence: Exception handling:
Description: checks for potential conflicts between a pair of aircraft based on relative configuration, with minimal dependence on velocity data [↑ PR.22]

10.1.1.2 References: [↑ PR.25] [↑ PR.26] [↑ PR.27]

Appears in: CONFLICT ALERT SIGNAL

= UNKNOWN

<table>
<thead>
<tr>
<th></th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>Separation_lateral = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Separation_vertical = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

= TRUE

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Separation_lateral &lt; PMCQ</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Separation_lateral &lt; PDS*runway_centerline_distance</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Separation_lateral)^2 &lt; PTSQi</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Separation_lateral)^2 &lt; PTH011</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separation_vertical &lt; DZP1Q</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separation_vertical &lt; DZP2Q</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel Absolute = True</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Converging = True</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convergence Rate (aircraft1, aircraft2) &gt; P*EDO</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

= FALSE

|                     | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| Startup             |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Controls.Reset = True | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| Separation_lateral < PMCQ | F |
| Separation_lateral < PDS*runway_centerline_distance | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| (Separation_lateral)^2 < PTSQi | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| (Separation_lateral)^2 < PTH011 | F | F |
| Separation_vertical < DZP1Q | F | F |
| Separation_vertical < DZP2Q | F |
| Parallel Absolute = True | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| Converging = True | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| Convergence Rate (aircraft1, aircraft2) > P*EDO | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
FUNCTION:
Relative_pos (aircraft1, aircraft2)

Obsolescence:

Exception handling:

Description: The relative position of aircraft2 with respect to aircraft1 is the vector drawn from aircraft1’s position to aircraft2’s position. It is represented as a vector of two components: x-coordinate and y-coordinate.

10.1.1.3 References:

Appears in: Acute Velocity-Position Vectors (FUNCTION), Converging (FUNCTION)

Relative_pos.x = aircraft2.pos_x – aircraft1.pos_x
Relative_pos.y = aircraft2.pos_y – aircraft1.pos_y
FUNCTION:
Relative_vel (aircraft1, aircraft2)

Obsolescence:

Exception handling:

Description: The relative velocity of aircraft2 with respect to aircraft1 is the vector derived by subtracting aircraft1’s velocity vector from aircraft2’s velocity vector. This vector is represented by an x-component and a y-component.

10.1.1.4 References:

Appears in: Converging (FUNCTION), Imminence check (FUNCTION)

Relative_vel.x = aircraft2.vel_x – aircraft1.vel_x
Relative_vel.y = aircraft2.vel_y – aircraft1.vel_y
Relative_vel.z = aircraft1.vel_z – aircraft1.vel_z
FUNCTION:
Separation_lateral (aircraft1, aircraft2)

Obsolescence:

Exception handling:

Description: the physical lateral separation distance between aircraft1 and aircraft2

10.1.1.5 References:

Appears in: Imminence check (FUNCTION), LINCON check (FUNCTION), MANCONP check (FUNCTION)

Separation_lateral = SQRT(aircraft1.pos_x – aircraft2.pos_x)^2 + (aircraft1.pos_y – aircraft2.pos_y)^2
FUNCTION:
Separation_vertical (aircraft1, aircraft2)

Obsolescence:

Exception handling:

Description: the physical altitude separation distance between aircraft1 and aircraft2

10.1.1.6 References:

Appears in: Imminence check (FUNCTION), LINCON check (FUNCTION), MANCONP check (FUNCTION)

Separation_vertical = | aircraft1.pos_z – aircraft2.pos_z |
**FUNCTION:**
Turning Into Each Other

**Obsolescence:**

**Exception handling:**

**Description:** determines if one aircraft in a pair is turning into the other, based on turning information.

**References:** [PR.21]

**Appears in:** MANCON check

= UNKNOWMN

<table>
<thead>
<tr>
<th></th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td>T</td>
</tr>
<tr>
<td>Converging (aircraft1, aircraft2) = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Separation vertical = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Acute Velocity Vectors (aircraft1, aircraft2) = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Acute Velocity-Position Vectors (aircraft1, aircraft2) = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1.turn = Unknown</td>
<td>T</td>
</tr>
<tr>
<td>Relative_pos (aircraft1, aircraft2) = Unknown</td>
<td>T</td>
</tr>
</tbody>
</table>

= TRUE

<table>
<thead>
<tr>
<th></th>
<th>F F F F F F F F F F F F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td></td>
</tr>
<tr>
<td>Converging (aircraft1, aircraft2) = True</td>
<td>T</td>
</tr>
<tr>
<td>Acute Velocity Vectors (aircraft1, aircraft2) = True</td>
<td>T</td>
</tr>
<tr>
<td>Acute Velocity-Position Vectors (aircraft1, aircraft2) = True</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1.turn = Left</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1.turn = Right</td>
<td>T</td>
</tr>
<tr>
<td>(Relative_pos (aircraft1, aircraft2).x * Aircraft1.vel_y) - (Relative_pos (aircraft1, aircraft2).y * Aircraft1.vel_x) &lt; 0</td>
<td>T</td>
</tr>
<tr>
<td>(Relative_pos (aircraft1, aircraft2).x * Aircraft1.vel_y) - (Relative_pos (aircraft1, aircraft2).y * Aircraft1.vel_x) &gt; 0</td>
<td>T</td>
</tr>
</tbody>
</table>

= TRUE

<table>
<thead>
<tr>
<th></th>
<th>F F F F F F F F F F F F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td></td>
</tr>
<tr>
<td>Controls.Reset = True</td>
<td></td>
</tr>
<tr>
<td>Converging (aircraft1, aircraft2) = True</td>
<td>T</td>
</tr>
<tr>
<td>Acute Velocity Vectors (aircraft1, aircraft2) = True</td>
<td>T</td>
</tr>
<tr>
<td>Acute Velocity-Position Vectors (aircraft1, aircraft2) = True</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1.turn = Left</td>
<td>T</td>
</tr>
<tr>
<td>Aircraft1.turn = Right</td>
<td>T</td>
</tr>
<tr>
<td>(Relative_pos (aircraft1, aircraft2).x * Aircraft1.vel_y) - (Relative_pos (aircraft1, aircraft2).y * Aircraft1.vel_x) &lt; 0</td>
<td>F</td>
</tr>
<tr>
<td>(Relative_pos (aircraft1, aircraft2).x * Aircraft1.vel_y) - (Relative_pos (aircraft1, aircraft2).y * Aircraft1.vel_x) &gt; 0</td>
<td>F</td>
</tr>
</tbody>
</table>
FUNCTION:
Validation(planeID)

Obsolescence:

Exception handling:

Description: A plane’s ID is valid if it uniquely identifies a track, and the track is owned by the requesting TCP

Comments:

References:

Appears in: CONTROLLER INPUT ACKNOWLEDGEMENT (OUTPUT MESSAGE)

Validation =
   True, if planeID uniquely identifies a track, and it is owned by the requesting TCP
   (if more than 1 planeID is passed for validation, returns true if at least one ID is owned by the requesting TCP)
   False, otherwise
Functions not implemented (insufficient documentation)

LMD (Lateral Miss Distance) for LINCON
TAVC (Time of Altitude Violation Conflict) for LINCON
TOANC (Time of Altitude Non-Violation Conflict) for LINCON
TOC (Time of Conflict) for LINCON, MANCON, PROCON
TOCA (Time of Co-Altitude) for LINCON
TOLNC (Time of Lateral Non-Conflict) for LINCON
TOLV (Time of Lateral Violation when converging) for LINCON
TOMA (Time of Minimum Approach) for LINCON
11 REFERENCES


