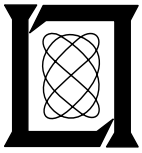
Project Report
ATC-206

Runway Status Light System Demonstration at Logan Airport

J. R. Eggert
R. J. Sasiela
M. P. Kastner
W. H. Harman
H. Wilhelmsen
T. J. Morin
H. B. Schultz
J. L Sturdy
D. Wyschogrod
P. M. Daly

12 October 1995

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS



Prepared for the Federal Aviation Administration,
Washington, D.C. 20591

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
ATC-206		

4. Title and Subtitle	5. Report Date
Runway Status Light System Demonstration at Logan Airport	12 October 1995
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
J.R. Eggert, R.J. Sasiela, M.P. Kastner, W.H. Harman, H. Wilhelmsen, T.J. Morin, H.B. Schultz, J.L. Sturdy, D. Wyschogrod, and P.M. Daly	ATC-206

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Lincoln Laboratory, MIT 244 Wood Street Lexington, MA 02173-9108	11. Contract or Grant No. DTFA01-95-X-02018

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Department of Transportation Federal Aviation Administration Washington, DC 20591	14. Sponsoring Agency Code

15. Supplementary Notes

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology under Air Force Contract F19628-95-C-0002.

16. Abstract

The Runway Status Light System (RSLS), developed under the FAA's Airport Surface Traffic Automation (ASTA) program, is intended to help reduce the incidence of runway incursions and airport surface accidents. It will do so by providing a preventive, back-up system of automatically controlled lights on the airport surface that inform pilots when runways are unsafe for entry or takeoff, and by providing controllers with enhanced surface radar displays. This report documents a proof-of-concept evaluation of the RSLS at Boston's Logan Airport. It details the methods used to provide the necessary surface surveillance and safety logic to allow a computer to operate the runway status lights and associated controller displays without human assistance. The system was installed and tested off-line at Boston's Logan Airport using an inexpensive commercial marine radar as a primary surveillance source. The system operated live and in real time but the runway status lights were not physically installed. They were displayed on a scale model of Logan Airport located in a demonstration room that had a good view of the airport. This allowed visual comparison between the actual aircraft and the resulting lights and displays. In addition to providing a convincing demonstration of the system, real-time viewing of the aircraft movement was an important aid in the development of the surveillance processing and safety logic software. Surveillance performance and runway status light operational performance were evaluated quantitatively. The probability of tracking an aircraft in movement areas with line-of-sight coverage was better than 98%. The false track rate was about four per hour, and the surveillance jitter was about 1 meter rms. From an operational point of view, had there been real lights on the field, it appears that they would have provided the intended safety back-up with little impact on airport capacity or controller and pilot workload. Only once in 15 minutes would the pilot population have observed a light in an incorrect state for more than four seconds. From the point of view of a specific cockpit crew, only once in 36 operations would a runway status light have been seen in an incorrect state for more than four seconds, and, furthermore, only once in 50 operations would light illuminations have interfered with normal, safe traffic flow. These are encouraging results for a system in an early demonstration phase because significant improvement is possible in all of these performance measures. Specific suggestions for improvement are included in this document.

17. Key Words	18. Distribution Statement
Runway Status Light System (RSLS) Airport Surface Traffic Automation (ASTA) Logan Airport Airport Surface Detection Equipment-X (ASDE-X)	This document is available to the public through the National Technical Information Service, Springfield, VA 22161.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	285	

FORM DOT F 1700.7 (8-72) Reproduction of completed page authorized

EXECUTIVE SUMMARY

INTRODUCTION

In the last two decades, at least nine major airport surface conflict accidents have occurred in the United States. In a thirteen-month period in 1990 and 1991, three airport runway conflict accidents together resulted in the loss of 43 lives. Runway conflicts start with runway incursions. About 200 runway incursions occur every year at US airports. With domestic air traffic predicted to increase by 3% annually over the next decade, the airport surface is expected to become increasingly crowded. The runway incursion problem must be addressed if future runway conflict accidents are to be prevented.

The Runway Status Light System (RSLS), developed under the FAA-administered research and development program in Airport Surface Traffic Automation (ASTA), is designed to reduce the incidence of runway incursions and airport surface accidents. The RSLS is a preventive, back-up system intended to provide runway status information to pilots via automatically controlled lights on the airport surface that indicate when particular runways are either unsafe to enter or are unsafe for takeoff. The system will also provide runway status and traffic information to controllers via surface radar display enhancements. The technical approach is to use automatic processing of surface primary and approach secondary radar data to drive the runway-status lights and associated controller surface traffic displays.

This report documents a proof-of-concept evaluation of the RSLS at Boston's Logan airport. It details the methods used to provide the necessary surface surveillance and control logic to allow a computer to operate the runway status lights and associated controller displays without human assistance.

THE RSLS SOLUTION

An important design goal of the RSLS is to enhance airport surface traffic safety with minimum operational impact on air traffic controllers and pilots, and with no reduction in airport capacity. The RSLS operational concepts were devised using extensive knowledge of actual airport operations, drawing on the experience of controllers, pilots, and safety experts. The RSLS is designed to be an all-weather, automatic system providing a safety backup to controllers and pilots, requiring no increase in controller workload, and incurring no decrease in airport capacity. The result of the RSLS design effort is a system that works in concert with existing procedures, and enhances runway safety.

Analysis of past runway accidents and incidents has shown that a complete airport surface traffic safety system should include a good surface radar, controller alerts in the tower cab, and runway status lights. A surface radar aids the controller to acquire and maintain a mental image of the present and pending runway operations. Digital display enhancements such as aircraft icons and identification tags on the surface radar display greatly enhance the display visibility. These enhancements require automatic surveillance processing to provide tracking information. The same tracking information derived for the display enhancements can be used to drive controller alerts and runway status lights. The runway status lights and enhanced surface traffic displays are the chief outputs of the RSLS system.

The improved controller surface traffic display, depicted in Figure ES-1, provides controllers with an advanced digital display of airport surface and approach traffic. Aircraft and surface vehicles are depicted by arrows that point in the direction of motion and whose size represents the size of the radar image. Tags provide velocity information and, when available from the Automated Radar Terminal

System (ARTS) computer, aircraft flight number, equipment type, and altitude. The traffic is depicted on a map of the runways and taxiways.

As shown in Figures ES-2 and ES-3, the runway status lights, composed of runway entrance lights and takeoff hold lights, present runway status information to pilots on and near the airport runways. These lights are driven automatically by computer processing of surface and approach radar information. The RSLS software detects the presence and motion of aircraft and surface vehicles on or near the runways, assesses any possible conflicts with other surface traffic, illuminates red runway entrance lights if the runway is unsafe to enter, and illuminates red takeoff hold lights if the runway is unsafe for departure. The runway status lights will have the effect of preventing runway incursions before they happen. They will prevent runway incursions without interfering with normal and safe airport operations.

A basic controller alerting capability is included in the RSLS for demonstration purposes. The RSLS safety logic was designed to support a full aircraft conflict identification capability in order to demonstrate that an integrated and uniform safety logic can support both lights and alerts. A full controller alerting capability would be required to also provide protection for certain conflict scenarios that cannot be prevented by status lights on the airport surface, such as an arrival to an occupied runway. The RSLS Logan Demonstration actually includes only one type of alert at present. A complete alerting capability was not implemented because of programmatic and scheduling constraints.

RSLS TECHNOLOGY

The development of the RSLS technology involved significant technical challenges. A major challenge was to detect and track aircraft and surface vehicles on the airport surface or in the approach space, with high reliability and with a low false track rate. Another challenge was to use these tracks in a safety logic to operate runway status lights automatically in a way that yields significant safety benefits without interfering with normal runway operations. The RSLS accomplishes both tasks well.

The design philosophy of the RSLS proof-of-concept evaluation was that it use commercial off-the-shelf hardware and custom portable software. A fully operational RSLS system would use the best available surface radar or ASDE (Airport Surface Detection Equipment), namely the ASDE-3. However, because the Logan ASDE-3 was unavailable for this proof-of-concept test, a Raytheon Pathfinder marine X-band radar was used instead. This radar was modified to improve its performance, but the modifications were done without significantly increasing the cost of the radar. The radar-to-computer interface, the only custom hardware component in the whole RSLS system, was designed to operate either with the X-band ASDE radar or with the ASDE-3. The RSLS system thus retains a great degree of hardware flexibility.

The flexibility of the RSLS system is achieved by relying on software to perform nearly all of its functions. The software was written using modular object-oriented programming techniques. This allows performance enhancements and new functions to be incorporated fairly easily. The software, which relies only on non-proprietary (or "open") operating system and display utilities, was successfully ported to several vendors' computers during the development process. It can be easily transferred to less expensive computers as they become available. Since the price of computation is decreasing rapidly, it may thus become economically feasible to field such systems at smaller airports in the future.

By using advanced digital image processing and tracking techniques, the radar tracking system achieved good surface surveillance performance with inexpensive radar hardware. The surveillance

processing software contains specific algorithms designed to overcome problems imposed by clutter, target splits, shadowing, and multipath. Sensor fusion software combines the surface radar tracks with approach radar tracks to provide one coherent picture of the airport surface and approach space. Display software shows the airport, aircraft, and data tags. This surveillance system could in fact be fielded separately from the runway status lights themselves, and represents a major advance in technology.

The RSLS safety logic includes capabilities for identifying the operational state of the aircraft under surveillance, projecting their possible future trajectories, and identifying potential conflicts. The projections are used to generate runway status light commands, and the preliminary conflict identification is used as the first step in an advanced alert-generation capability. Using a single safety logic for both runway status lights and controller alerts results in coherent and complementary safety benefits from the lights and alerts.

THE RSLS LOGAN DEMONSTRATION

Boston's Logan airport has proven to be an excellent testing and demonstration facility for the RSLS. Both Boston and New England Region FAA and Massachusetts Port Authority (Massport) personnel have given exceptional support during system installation, testing, and demonstration. Because of its complex layout, difficult radar environment, good traffic mix, variable weather, and high operations volume, Logan airport is a good airport for the RSLS demonstration.

The demonstration system does not affect airport operations in any way; there are no runway status lights on the airport surface, there are no displays in the Logan tower cab, and there is no interaction with either controllers or pilots. The demonstration is off-line but real-time.

The demonstration site is a Massport conference room on the sixteenth floor of the new air traffic control tower building at Boston's Logan Airport. This room has large windows that offer a good view of the airport and provide an opportunity to observe the live performance of the system. A photograph of the room is shown in Figure ES-4. The demonstration room has a simulated D-BRITE (Digital Bright Radar Indicator Tower Equipment) display, monochrome and color versions of simulated ASDE-3 displays, and (for reference purposes) an unmodified Raytheon Pathfinder marine radar display.

In a deployed system the radar tracking information would be used to control runway status lights visible to pilots on the runways and taxiways. In the RSLS Logan Demonstration, these runway status lights are simulated by a scale model depiction of the airport surface with computer-controlled lights. A photograph of the airport model is shown in Figure ES-5.

The X-band radar antenna is located on the roof of the old control tower building as shown in Figure ES-6. Supporting equipment is located on the fifteenth floor and on the seventh floor of the old control tower building.

The RSLS was first demonstrated in October 1992, its last major component was added in December 1992, and it has been shown to many FAA, other governmental, and industry personnel since then. In the same period the software and hardware have been upgraded to further improve performance.

RSLS PERFORMANCE

The RSLS was subjected to two types of performance assessment: surveillance tracking reliability and runway status light performance. For the surveillance reliability assessment, probability of tracking, false track rate, and surveillance accuracy statistics were evaluated. For the runway status light performance assessment, missed detection, false alarm, interference, and light infringement statistics were evaluated. The assessment was based on a brief but representative excerpt of operations at Logan airport. A brief summary of the results is given here.

The RSLS performed well for a proof-of-concept system. The probability of detection was better than 98%, the false track rate was about 4/hr, and the surveillance jitter was about 1 meter rms. From an operational point of view, had there been real lights on the field, it is likely that they would have provided a valuable safety back-up benefit with little impact on either airport capacity or on the workload of the tower controllers and the cockpit crews. There were few false alarms and no light infringements. Only once in 15 minutes would the pilot population have observed a light in an incorrect state for more than four seconds. From the point of view of a specific crew, only once in 36 operations would a runway status light have been seen in an incorrect state for more than four seconds, and, furthermore, only once in 50 operations would light illuminations have interfered with normal, safe traffic flow. This interference would usually have been of quite short duration. The principal effect of the immature performance of this system would not have been to adversely impact airport capacity or controller workload, but simply to fail to live up to its full potential for detecting unsafe conditions and preventing runway incursions.

These are encouraging results for a proof-of-concept system in an early development phase; improvement would be possible and necessary in every component of the system if it were to be used operationally. Where appropriate, specific suggestions for improvement are given in each chapter of this document.

THE NEXT STEP

The next step in the development of the RSLS concept is to transition from an off-line demonstration to an on-line field demonstration with real lights. That step requires a) the procurement and installation of a set of runway status lights at appropriate locations on an airport, b) improving the RSLS performance (by further analysis, algorithmic improvements, measurements, off-line field testing, and tuning), and c) improving the RSLS software reliability via a rigorous hardening and validation process. With the experience gained in the development of the demonstration system, the required improvement of the system to bring it to a field demonstration would be achievable within two years after a go-ahead decision if such a field demonstration were to be conducted at Logan Airport.

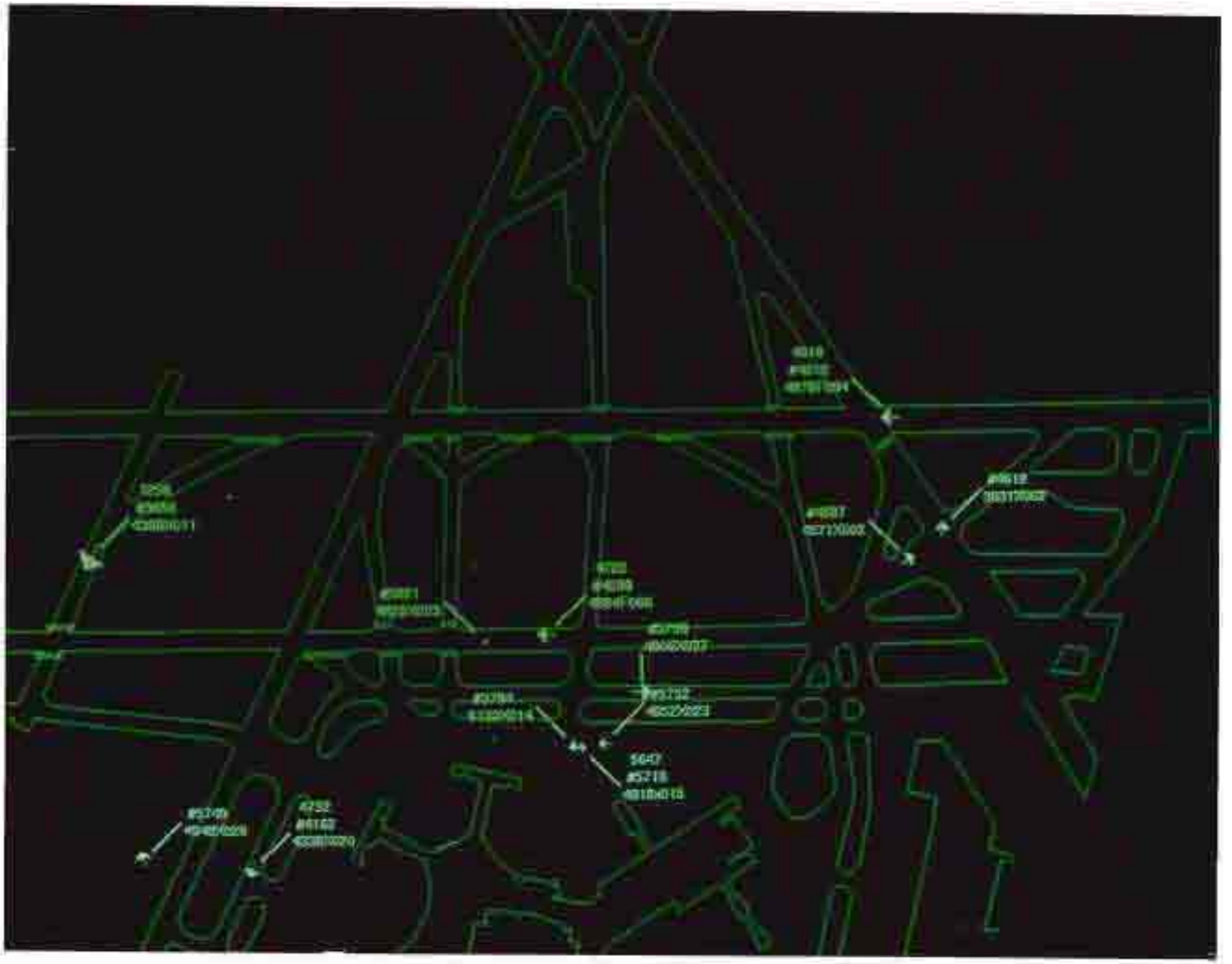


Figure ES-1. Improved controller surface traffic display.

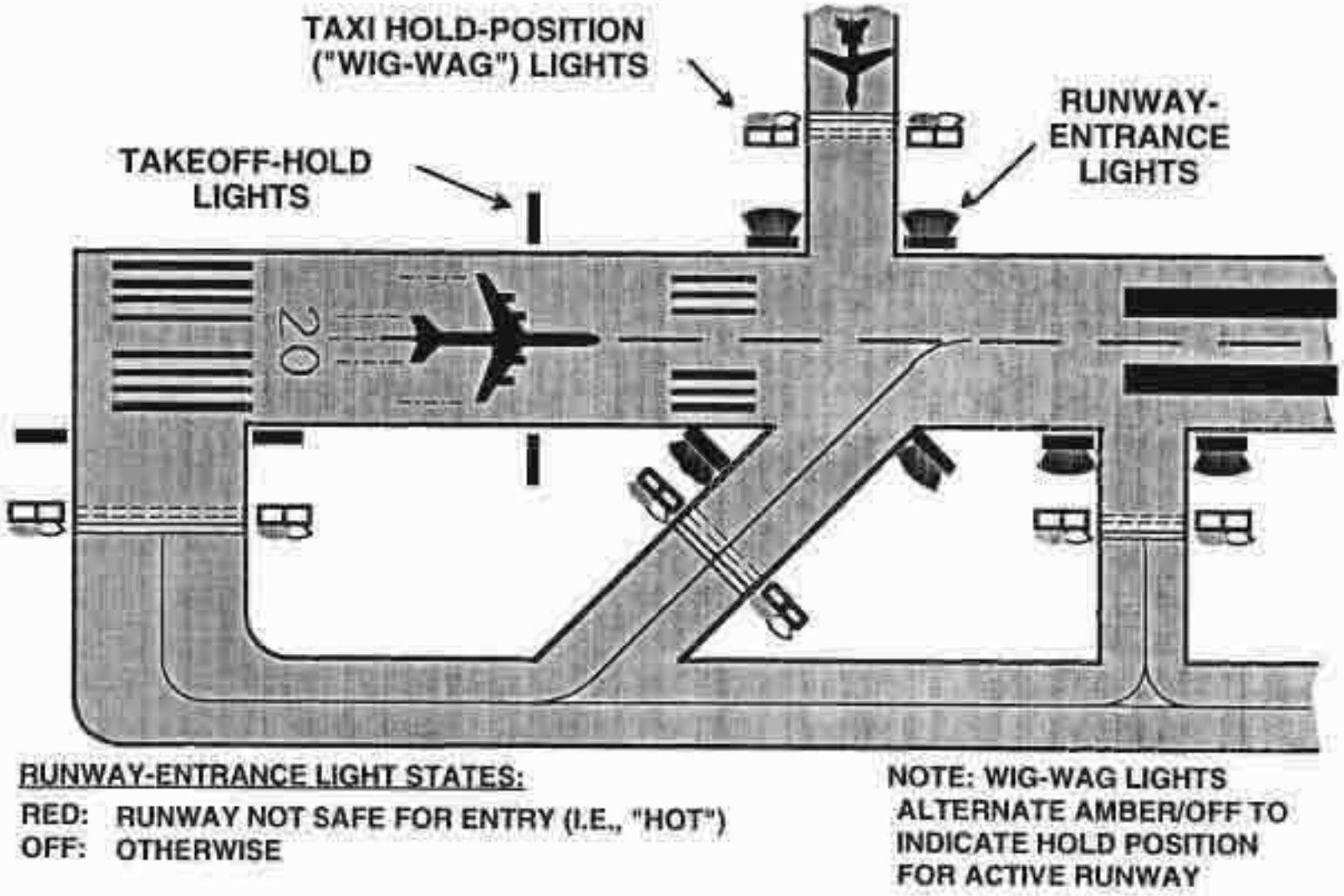


Figure ES-2. Runway-entrance lights in operation for an aircraft landing on a runway.

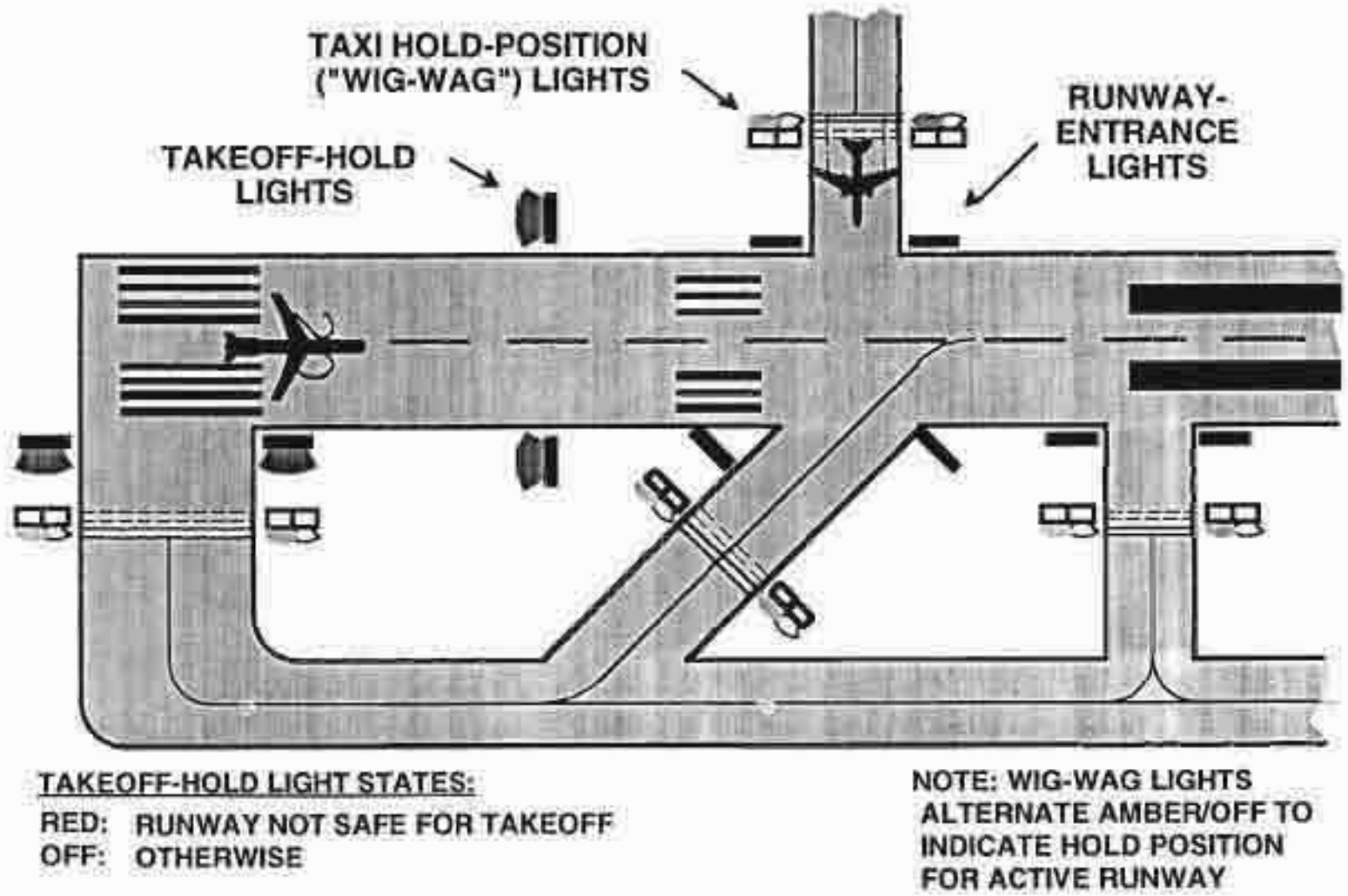


Figure ES-3. Takeoff-hold lights in operation for an aircraft in position for departure with crossing traffic.

Figure ES-4. RSLS Logan demonstration room.

Figure ES-5. RSLS Logan demonstration model board.

Figure ES-6. Radar antenna and view of Logan airport.

ACKNOWLEDGMENTS

The primary authors of the sections of the report were as follows.

Executive Summary, James R. Eggert

The RSLS Demonstration System Architecture, James R. Eggert

Ground Surveillance Radar, Richard J. Sasiela

Surveillance Processing, Richard J. Sasiela

ARTS Interface, James R. Eggert

Sensor Fusion, James R. Eggert

Safety Logic, Marcia P. Kastner

RSLS System Software Architecture, Peter M. Daly, James R. Eggert, Theodore J. Morin, Hayden B. Schultz, James L. Sturdy, Daniel Wyschogrod.

ASDE Performance, William H. Harman

Status Light Performance, Harald Wilhelmsen

In addition, the following were significant contributors during the development of the demonstration.

Radar Equipment and High-Speed Data Hardware: John L. Cataldo, Ralph S. Cataldo, Joseph DiBartolo, Richard L. Ferranti, James M. Flavin, Robert, D. Kenney, Melvin Labitt, John J. O'Rourke, John A. Macinni, Wallace A. Reid, Kenneth W. Saunders, Melvin L. Stone.

Surveillance Processing: Richard W. Bush, Archon Fung, Walter S. Heath, M. Loren Wood.

Sensor Fusion: Teresa L. Hall

Safety Logic: Walter L. Brown, Steven Bussolari, Walter M. Hollister, Ervin F. Lyon, Douglas Marquis, Steven D. Thomson, Jerry D. Welch.

Documentation: David Kassay

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES
(continued)

LIST OF FIGURES
(continued)

LIST OF TABLES

1. THE RSLS DEMONSTRATION SYSTEM ARCHITECTURE

1.1 INTRODUCTION

The Runway Status Light System (RSLS) Logan Demonstration System was developed by the Massachusetts Institute of Technology (MIT), Lincoln Laboratory, under an interagency agreement between the Department of Transportation – Federal Aviation Administration (FAA) and the United States Air Force (USAF). The RSLS was previously called Airport Surface Traffic Automation with Runway Status Lights (ASTA-1). The goal of the RSLS Demonstration System is to use automatic processing of surface primary and approach secondary radar data to drive simulated runway-status lights (Figure 1-1). This goal has been accomplished.

This document presents a description of the design motivation, methodology, and implementation for the RSLS Demonstration System, and presents an analysis of the system performance. This system architecture section provides an overview of the entire system on a functional block scale; detailed descriptions of the various subsystems are presented in individual sections.

1.2 MOTIVATION

In the past twenty years, there have been at least six fatal and two major non-fatal runway-conflict accidents in the United States. In a thirteen-month period in 1990 and 1991, runway-conflict accidents occurred at Atlanta, Detroit, and Los Angeles, which together resulted in the loss of 43 lives. With domestic air traffic predicted to increase by 3% annually over the next decade, and with little airport construction envisioned in the near future, it is clear that the airport surface traffic control problem must be addressed if future runway-conflict accidents are to be prevented.

A first step towards addressing surface traffic control is better surveillance. Indeed, the FAA is presently engaged in the procurement of 33 ASDE-3 (Airport Surface Detection Equipment) radars, which will provide improved surface surveillance with higher resolution, reduced clutter, and better performance in rain than the older ASDE-2 radars now in operational use. Significant additional performance enhancements are also possible using appropriate processing technology.

Better surveillance by itself, however, will not address the whole problem. Controllers typically do not rely on surface radar in times of good visibility. A surface radar by itself does not offer conflict identification or prevention information to the controller. Also, the surveillance information is not directly available to the aircraft pilots and vehicle operators on the airport surface. What is needed is an automation system that provides aids to the controllers and surface status to the pilots.

Progress towards the design of such automation and the prevention of surface accidents requires an understanding of the basic nature of the surface traffic problem. The surface traffic problem can be placed into three increasingly encompassing classes: accidents, high-hazard incidents, and runway incursions. Accidents, though great in consequence, are relatively few in number, so an analysis of airport surface accidents benefits greatly by the statistical inclusion of high-hazard incidents. High-hazard incidents denote those where at least one aircraft was at high speed, and where the minimum

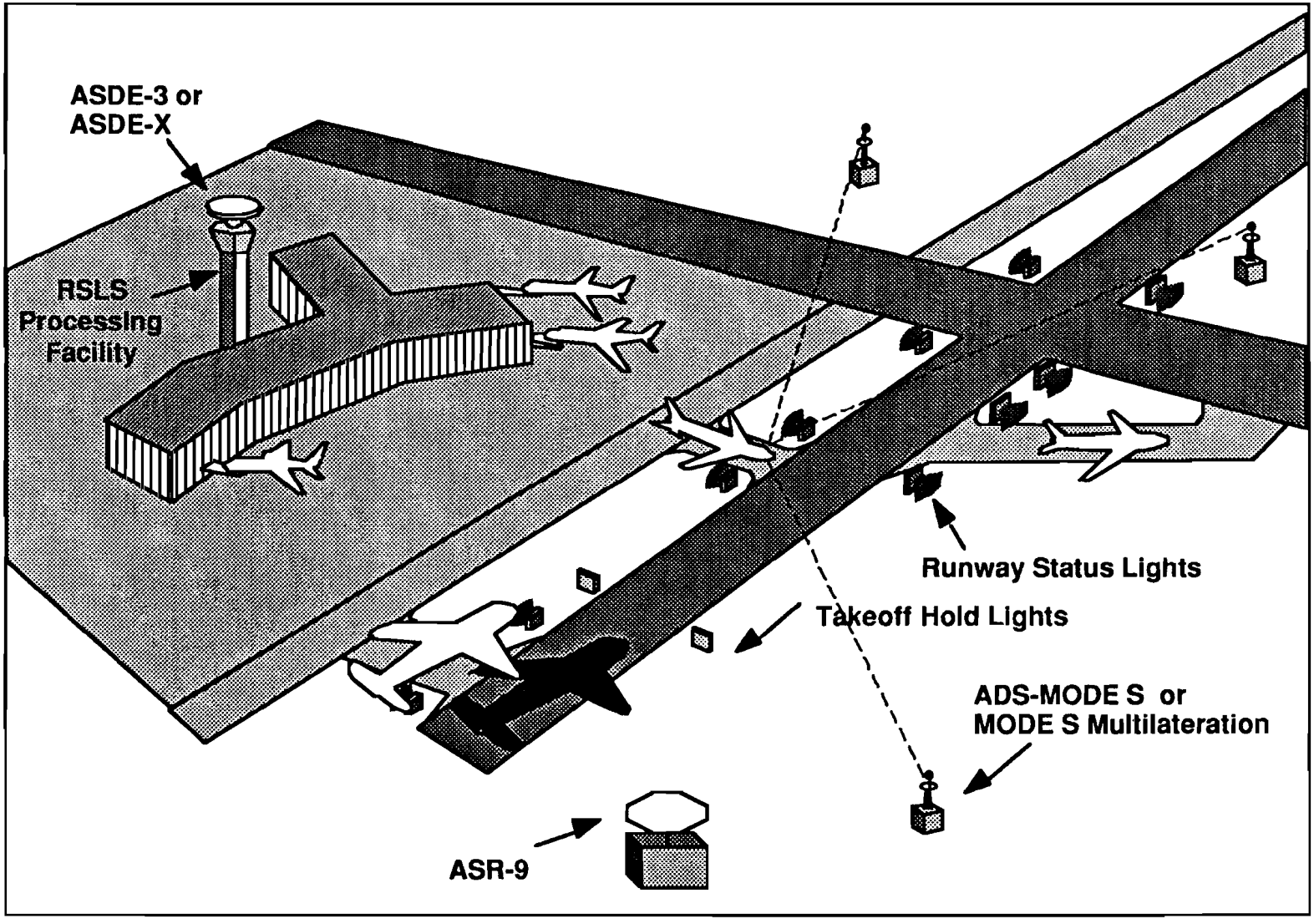


Figure 1-1. RSLS system concept.

separation was 50 feet or less. Figure 1-2 shows the frequency of accidents and incidents for various conflict scenarios, accumulated separately by scenario geometry. The top eight scenario geometries are depicted in Figure 1-3. It is clear from Figure 1-2 that these top eight conflict scenarios represent more than 90% of the accidents and incidents. Any surface traffic safety system must address these top scenarios to be effective.

Runway incursions represent a larger class of events than accidents and high-hazard incidents. A runway incursion is defined as any occurrence at an airport involving an aircraft, vehicle, person, or object on the ground that creates a collision hazard or results in loss of separation with an aircraft taking off, intending to take off, landing or intending to land. Clearly, preventing runway incursions is an effective way to prevent airport surface accidents, and a good airport surface traffic automation system must also be effective at reducing runway incursions.

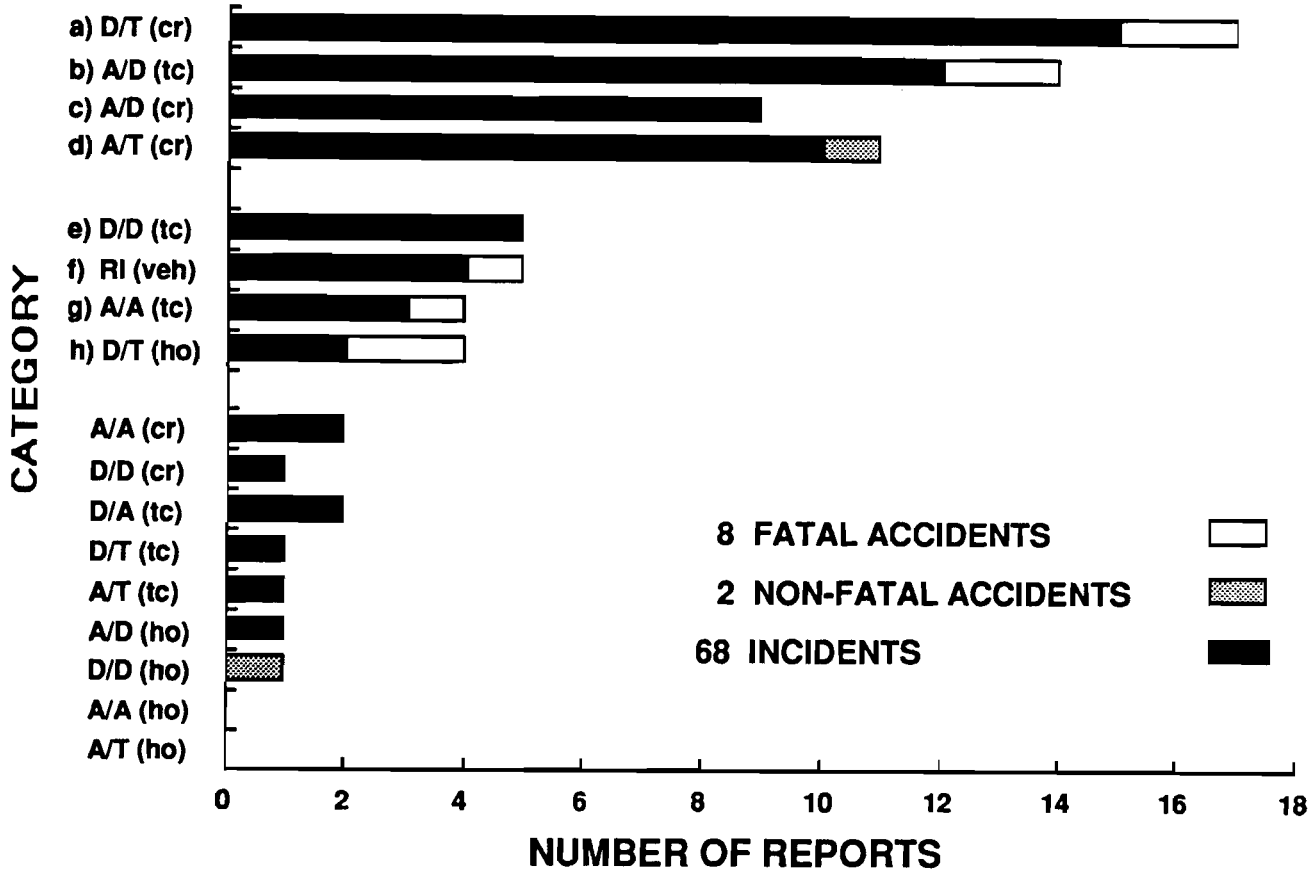


Figure 1-2. Accident and incident scenario frequencies. D=departure, A=arrival, T=taxi, veh=vehicle; cr=crossing, tc=tail-chase, ho=head-on. Data represent all fatal and US airport major non-fatal accidents in the period 1972-1992, and US airport high-hazard incidents from 1985-1992.

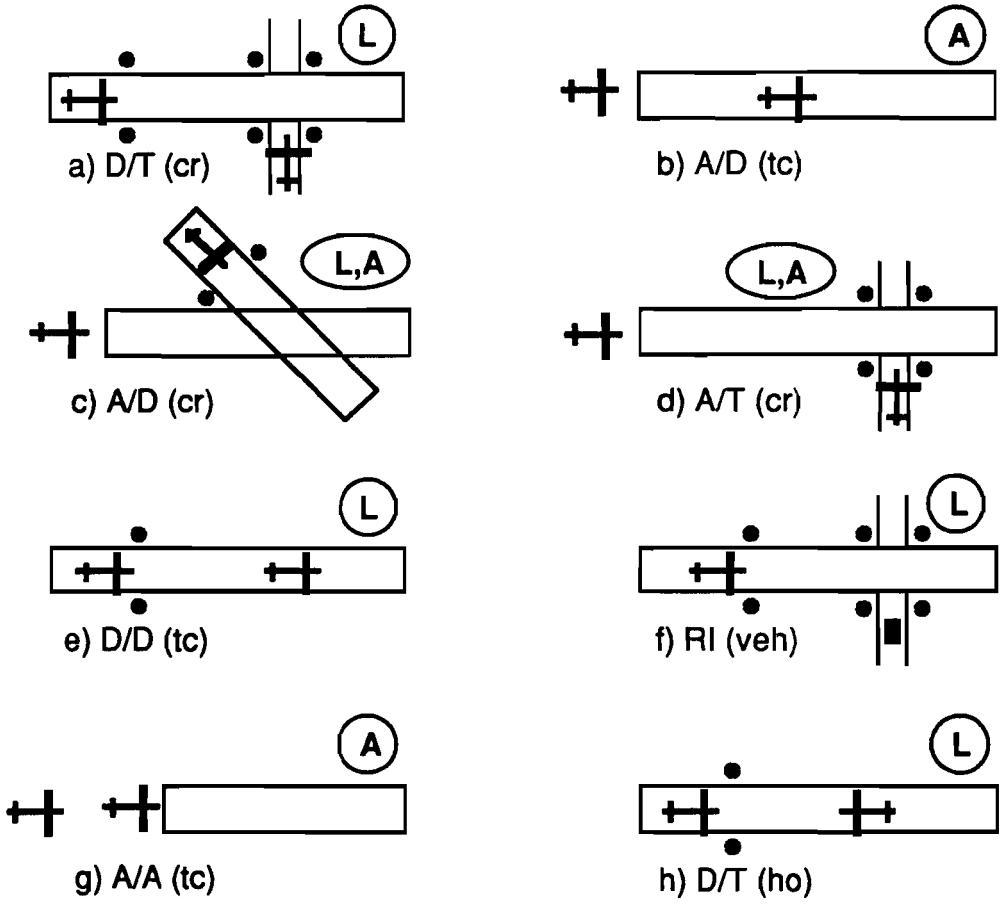


Figure 1-3. The eight most frequent accident and incident scenarios. L=lights address the scenario, A=controller alerts address the scenario.

1.3 APPROACH

A complete airport surface traffic safety system should include three products that together will address all of the top eight airport surface conflict scenarios. These three products are runway-status lights, controller alerts, and enhanced controller displays. Runway-status lights will be used to provide current runway status information to pilots and vehicle operators, to prevent runway incursions before they happen. Controller alerts will be used to direct controllers' attention to existing conflicts between aircraft on or near the runways. Because runway-status lights do not address some of the top accident and incident scenarios, and because controller alerts do not always provide sufficient time for controllers and pilots to correct a situation once it has developed, only a combination of lights and alerts will address all the most common scenarios. The FAA has contracted for the development of an operational controller alerting system, known as the Airport Movement Area Safety System (AMASS). Enhanced ASDE displays will present symbology to describe aircraft position, size, direction and speed of motion, altitude, aircraft flight number, and equipment type. In addition to airport surface traffic, aircraft on approach to runways will also be depicted on the ASDE displays.

The RSLS Demonstration System incorporates only simulated runway-status lights and enhanced controller displays, and does not include a complete controller alert system. Runway-status lights, composed of runway-entrance lights and takeoff-hold lights, provide the greatest part of the protection afforded by the safety system, for several reasons. First, in any time-critical conflict scenario, the most

effective safety system product is one that is directly accessible by the pilots. That direct access is allowed by runway-status lights, but not by controller alerts. Second, runway-status lights act to prevent runway incursions before they happen, whereas controller alerts only occur after a conflict has been identified. Third, runway-status lights are effective in a greater fraction of the accident and incident scenarios than are controller alerts. Therefore, for a combination of reasons, including maximizing system effectiveness in the face of developmental schedule constraints, and reducing the duplication of research efforts, the RSLS Demonstration System does not include controller alerts except for demonstration purposes.

1.3.1 RSLS Runway-status Lights

There are two types of RSLS runway-status lights: runway-entrance lights and takeoff-hold lights. The two types of lights are driven in concert by a single safety logic, and operate together to prevent runway incursions and accidents. The runway-status lights function fully automatically in response to real-time surveillance.

The runway-status lights have two states: on (red) and off. The runway-status lights indicate runway status only; they do not indicate clearance. A green state was specifically avoided to prevent any false impression of clearance. Clearance is to remain the sole responsibility of the air traffic controller, and is not to be provided or implied by the runway-status-light system. An amber state was also avoided, because in the case of runway-entrance lights it could tend to be confused with the amber color of the International Civil Aviation Organization (ICAO) wigwag taxi-hold (wigwag) lights. We also avoided using an amber state because, when used in Runway Entrance Lights, it could be confused with an amber ICAO taxi-hold (wigwag) light. The lights are designed to be as conspicuous as possible while minimizing the possibility of confusion with other light systems.

Runway-status lights are designed to be generally invisible to pilots of aircraft at high speed. This design decision was made so that red lights, especially lights that suddenly turn red, will not be shown to pilots whose aircraft speed precludes them from making sudden maneuvers. Runway-entrance lights are hooded so as not to be visible to pilots of aircraft on the runway. Runway-entrance lights are not generally active at runway-runway intersections. Takeoff-hold lights are also hooded, and require that an aircraft be in position for takeoff to be illuminated. The design of the fixtures and light logic thus generally prevents pilots of aircraft at high speed from seeing red runway-status lights.

A suggested fixture for the runway-status lights would be the standard fixture used for the ICAO wigwag lights, with the amber lenses replaced by red, and the lamps upgraded to 75 W bulbs (Figure 1-4). These fixtures use redundant light bulbs and other electrical components to minimize the impact of single-component failures on the operation of the system. They are also in current production, allowing off-the-shelf delivery. The RSLS Demonstration System does not in fact incorporate an actual field lighting system, but simulates the runway-status lights by the use of a model board and computer-driven displays.

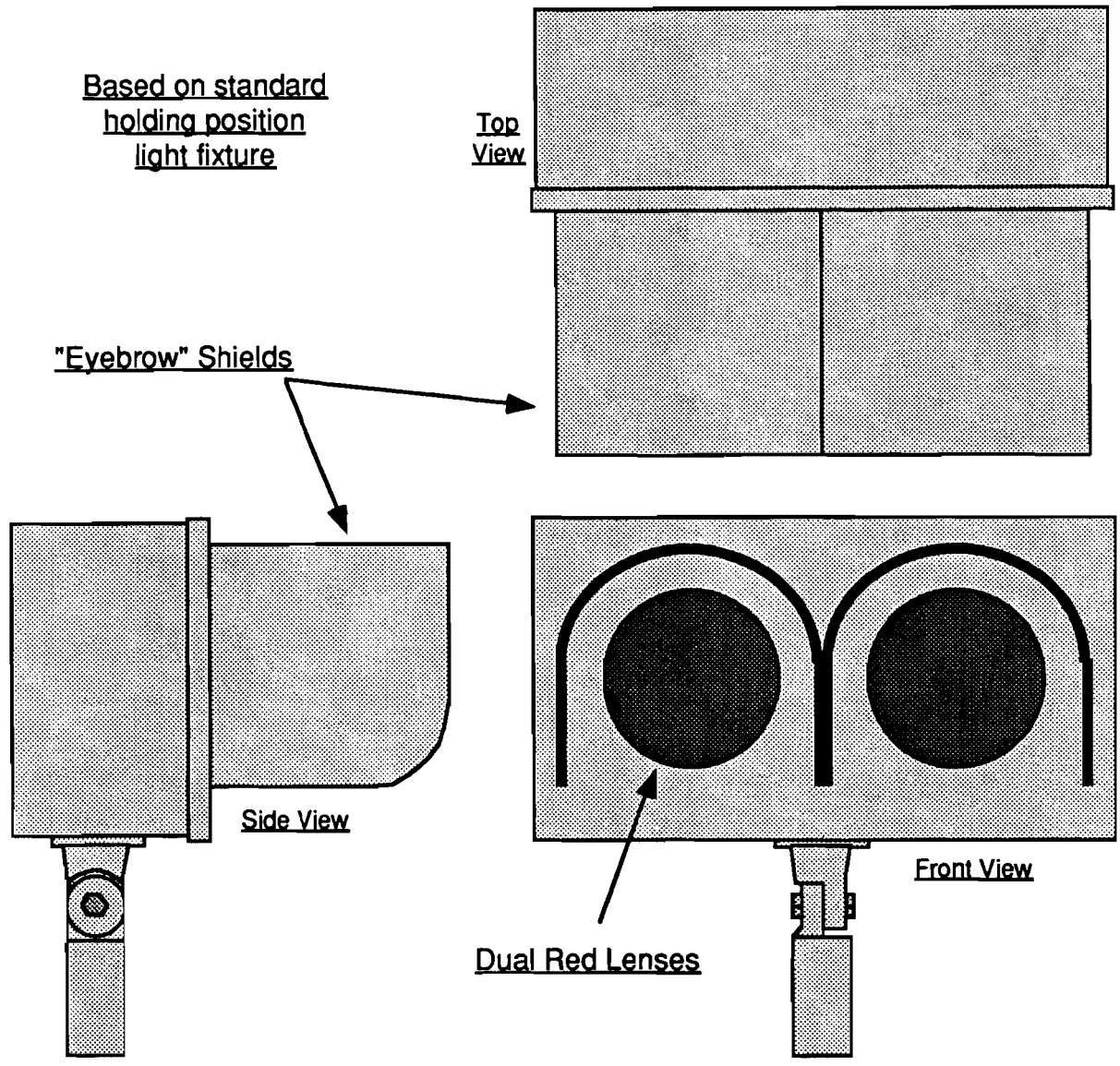


Figure 1-4. RSLS status light fixture.

Runway-entrance lights (RELs) will be located at the taxiway entrances to runways and will be positioned on either side of the taxiway, near the runway edge and well beyond the hold line (Figure 1-5). RELs will also be located at runway-runway intersections, though they will not always be implemented or actuated there. RELs will be illuminated to indicate to aircraft pilots and vehicle operators that the runway is hot, i.e., that it is being used for a high-speed operation (e.g., takeoff or landing), and that the runway is presently unsafe to enter at that intersection. RELs will be extinguished when the runway is no longer unsafe to enter at that intersection.

Takeoff-hold lights (THLs) will be located at takeoff-hold positions and positioned on either side of the runway near the runway edge (Figure 1-6). THLs will indicate to aircraft pilots that the runway is "dirty," i.e., that it is presently occupied or is about to be occupied, and that the runway is presently unsafe for departure from that takeoff-hold position. THLs will be extinguished when the runway is clear, or when the takeoff-hold position is empty.

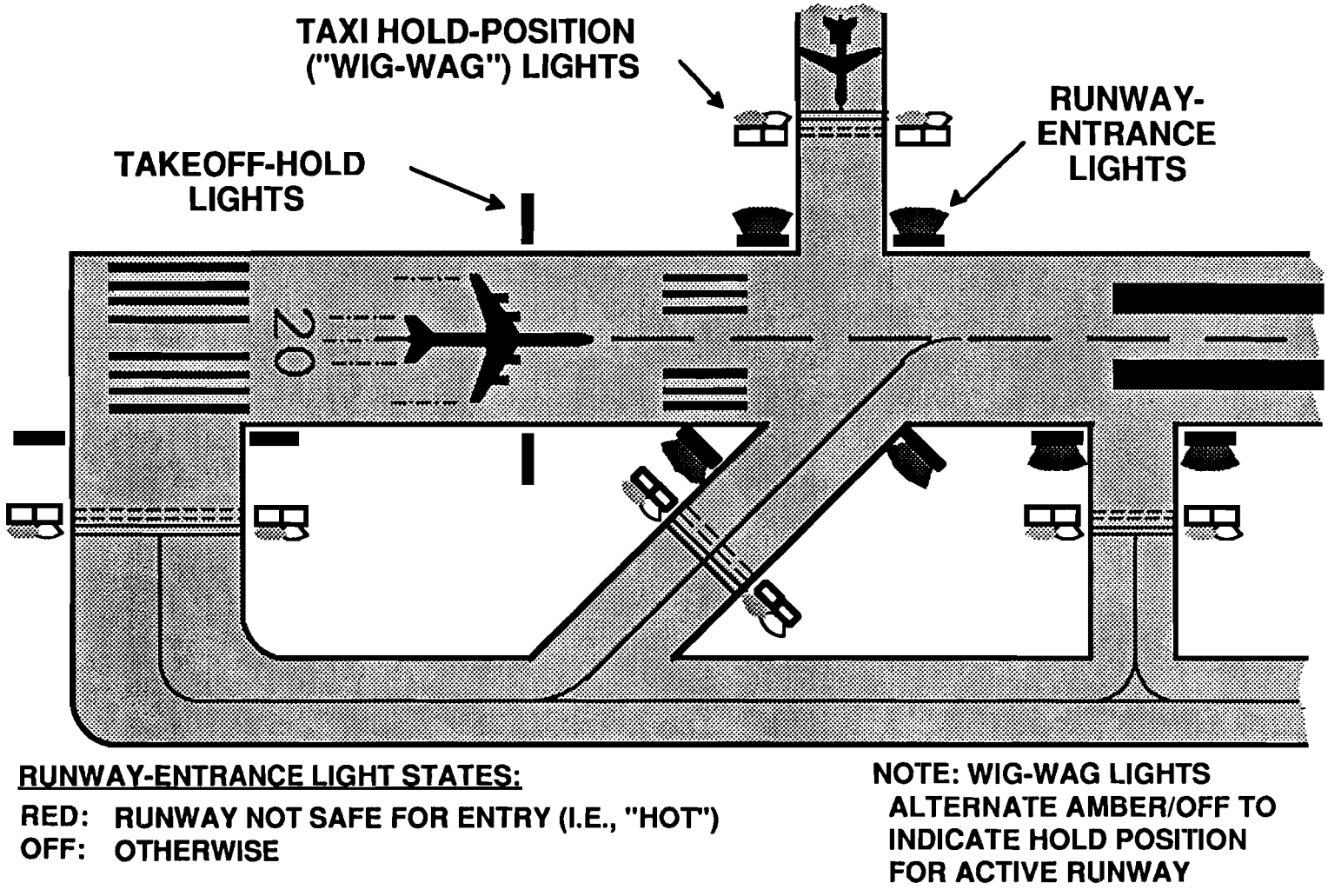


Figure 1-5. Runway-entrance lights in operation for an aircraft landing on a runway.

1.3.2 ASDE Display Enhancements

The RSLS system provides several display enhancements to a surface radar or ASDE display. An ASDE-3 display contains radar video with blanking to reduce visible clutter, and line graphics to depict runway and taxiway edges and building outlines. RSLS provides iconic depiction of tracked traffic, symbolic tags for each icon, approach bars for aircraft inside the outer marker, depiction of runway-status-light state, and special markings for aircraft identified as being in conflict. For demonstration and debugging purposes, some additional safety logic internal information can also be displayed. The RSLS Demonstration System supports both monochrome and color ASDE displays.

The RSLS Demonstration System does not include radar video on its display. This was done to reduce development time and equipment expenses. This omission is not envisioned for a complete RSLS system.

All tracked traffic that is considered to be reliable is displayed as icons on the ASDE display (Figure 1-7). The icon represents the position and direction of motion of the track, and, for tracks with ASDE image information, is drawn with a size proportional to the area of the ASDE image.

Each displayed track has a symbolic tag, connected to the icon with a leader line. The display software selects the leader line direction to eliminate possible overlapping tags and crossing leader lines. The tag can be displayed in two formats. The primary format shows aircraft altitude in hundreds of feet, and track velocity in knots. When available, this format also shows aircraft flight code and equipment type, the latter alternating with the velocity field. The secondary format is meant primarily for system debugging, and shows internal track numbers, track surveillance source or sources, altitude in feet, velocity in knots, and aircraft flight code when available.

Aircraft on approach to runways and inside the outer marker are displayed on approach bars. An approach bar is a short line segment drawn near the approach end of the runway. It is drawn at a different scale and represents the approximately five nautical mile distance from the outer marker to the runway threshold. Aircraft identified as being on approach to a runway are shown as diamonds on the approach bar. When the aircraft is near enough to the runway to appear on the scale of the ASDE display, it disappears from the approach bar and appears as a normally displayed target.

Runway-status-light state information is also rendered to the enhanced ASDE display. It can be drawn in two different symbologies. An illuminated Runway Entrance Light (REL) can be represented by a bar across the intersecting taxiway and THLs as a bar across the runway. Alternatively, RELs and THLs can be drawn as acute triangles on either side of the taxiway or runway, oriented to depict the directionality of the actual lights. If the RSLS safety logic identifies targets as being in conflict, this information can be drawn to the ASDE display. The targets are circled, and remain so until the conflict is resolved.

Additional RSLS internal information can also be shown on the ASDE display. This information includes the target state identification (taxi, stopped, arrival, etc.), the range of predicted target positions produced by the safety logic, and artificial target (sprite) positions and control information.

The ASDE display enhancements also allow the possibility that future tower displays may be in color. The color displays show the runway, taxiway, and building outlines and approach bars in green, target icons and tags in yellow, illuminated runway-status lights in red, and conflict alert circles in white. The use of color in an ASDE display is extremely useful in enhancing its visibility to controllers, and thus the rate and efficiency of information comprehension by controllers.

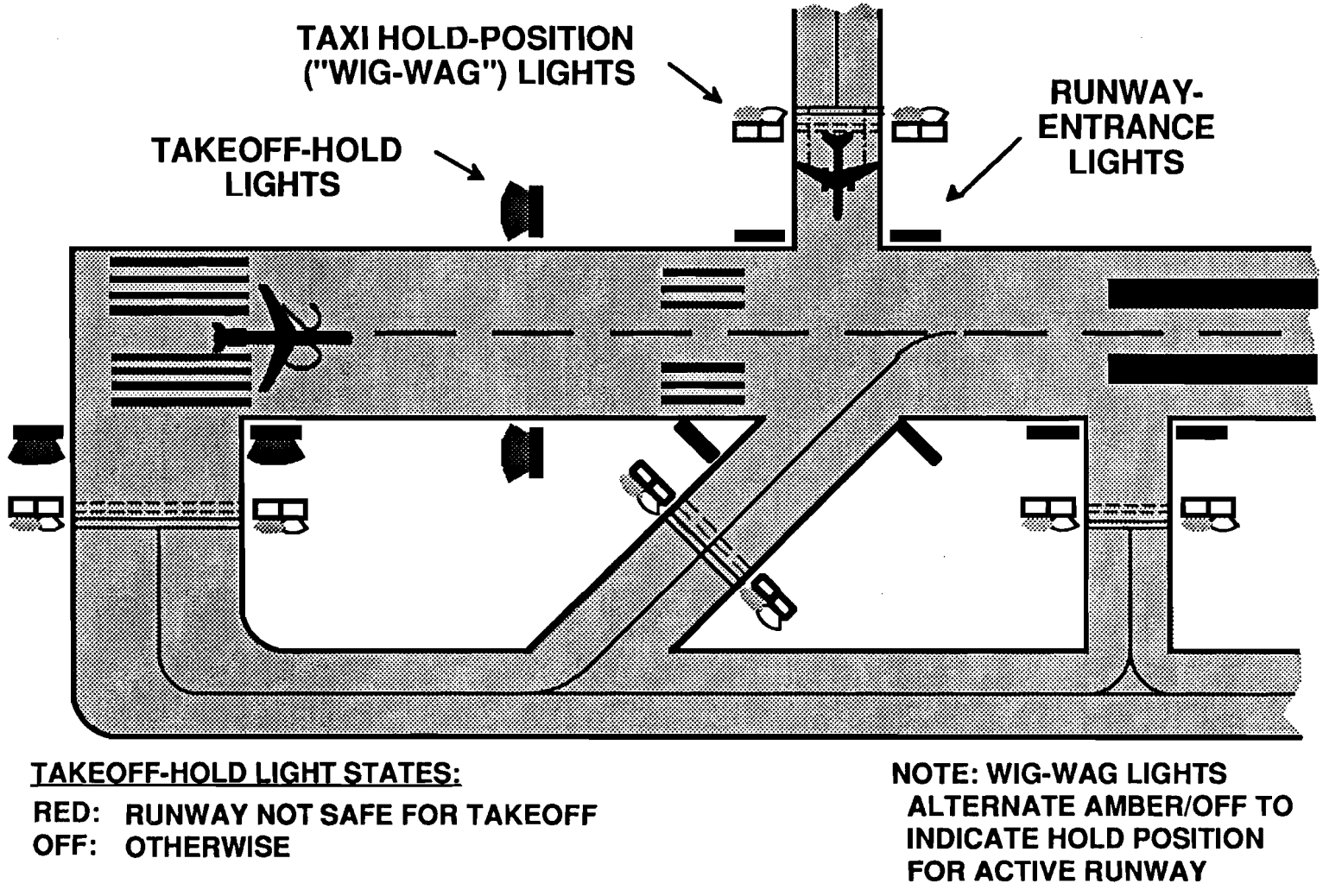


Figure 1-6. Takeoff-hold lights in operation for an aircraft in position for departure with crossing traffic.

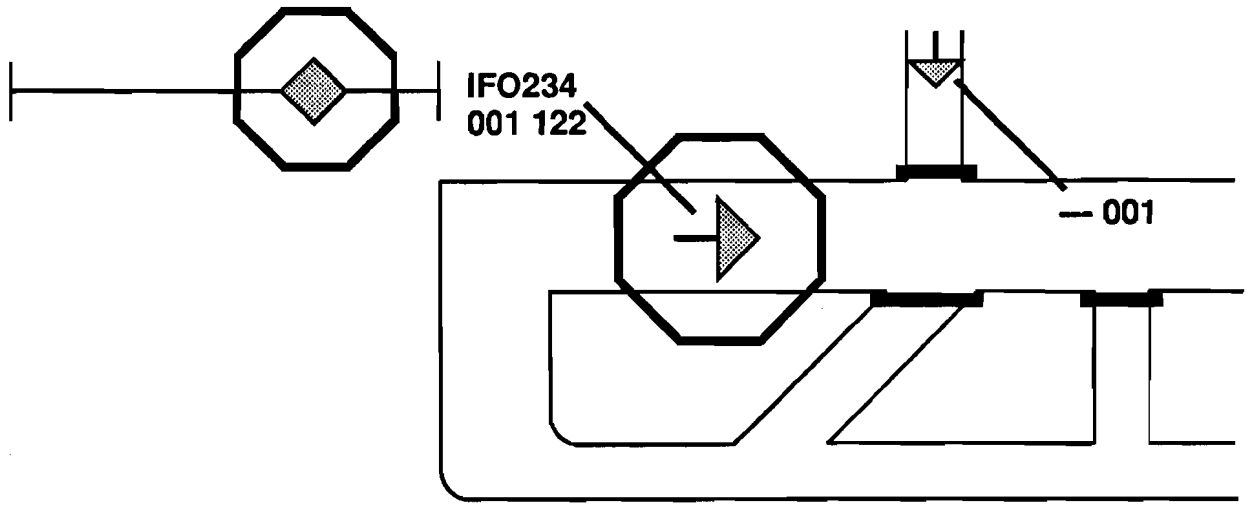


Figure 1-7. ASDE display enhancements. Arrows indicate position, direction of motion, and size of the radar tracks. Circles may be used for stationary tracks. Data tags present the track's altitude, velocity, aircraft flight number, and equipment type. An approach bar depicts an aircraft on approach to a particular runway; its two endpoints represent the runway threshold and the outer marker.

1.3.3 Controller Alerts

Although the RSLS Demonstration System does not supply a complete controller alerting system, it does have an architecture that supports such an alerting system. To demonstrate this capability, one type of alert was included in the system. The conflict that can be detected is between an arriving or landing aircraft and a stopped target on the arrival runway. When a conflict is detected, the conflicting targets are circled on the ASDE display, and a synthesized voice alert is generated. The voice alert gives a warning signal and then the location and type of the conflict. A complete controller alerting system would include the capability of detecting perhaps a dozen general conflict types.

1.4 RSLS DEMONSTRATION METHODOLOGY

The main objective of the RSLS Demonstration System is to develop a surface safety system that is capable of preventing most runway incursions and of identifying impending surface conflicts. This objective required the development of several significant capabilities:

- An ASDE surface radar to provide radar images with sufficient resolution and scan frequency for tracking surface traffic.
- A surface radar processing system to process automatically information from the ASDE, performing clutter rejection, target morphological processing, and scan-to-scan association.
- An interface to the Automated Radar Terminal System (ARTS) computer to provide surveillance data for aircraft on approach to the runways.
- A sensor fusion process to merge automatically tracks from the ASDE processing system with tracks from the ARTS computer, and perform multipath rejection.

- A safety logic system to classify and predict aircraft behavior, identify surface conflicts, and drive runway-status lights and controller alerts.
- A display system to allow basic evaluation and demonstration of the entire system.
- A performance analysis suite to allow a detailed evaluation of the runway-status-light-system operation.

An overview of the system architecture is shown in Figure 1-8. The ASDE surface radar analog signal is digitized and processed in the radar processing system. Its tracks, and those derived from Airport Surface Radar (ASR) surveillance using the ARTS tap, are passed on to sensor fusion. The output of sensor fusion is a single set of tracks presenting a coherent view of the surface and approach traffic to the safety logic. Safety logic identifies aircraft states, predicts future target positions, determines runway-status-light states, and generates alert commands. The system output is shown on several displays.

Several system requirements resulted in basic engineering design choices. These include the following:

- December 1992 demonstration
- Off-line, non-interfering demonstration
- Real-time response to live traffic
- Minimal system response time
- Minimal hardware design time
- Adequate design flexibility.

The Demonstration System was required to be functional in the December 1992 timeframe. This schedule precluded the use of the ASDE-3 radar at Logan, because that radar was not expected to be operational in sufficient time. Thus another surface radar system, a Raytheon Pathfinder X-band marine radar was installed on the roof of the old control tower building at Logan for use in development and demonstration of the system. This radar is called the ASDE-X.

The Demonstration System was required to have no operational impact. Thus there are no actual runway-status lights, there is no RSLS presence in the control tower cab, and RSLS does not interfere with normal FAA or aircraft operations. All demonstration displays and system control screens are located in a demonstration room on the 16th floor of the Boston Logan tower, or in other non-interfering areas. All demonstration equipment operates on a non-interfering basis; a failure in any demonstration subsystem can not result in operational interference.

The real-time nature of the application mandates sufficient processing throughput to keep up with peak data loads. For the case of surface radar processing, this requires the use of a fairly powerful computer. Most of the subsystems operate on separate computer platforms to spread the computation load and to reduce the system impact of a single-point failure.

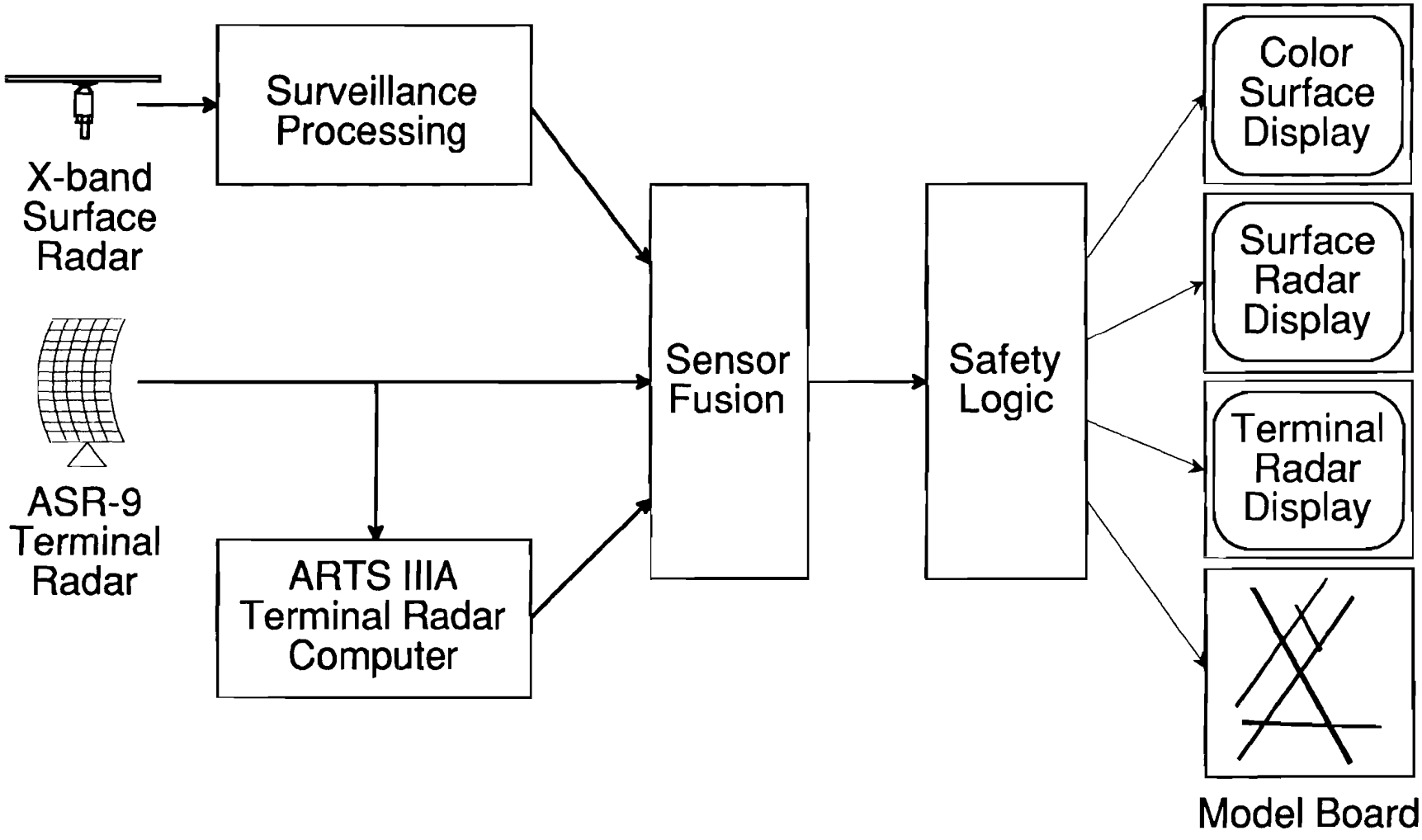


Figure 1-8. RSLS Demonstration System architecture overview.

A real-time safety system must take into account that time-critical situations can occur where processing delays would be intolerable. Several design choices, most notably the order of operation in the ASDE clutter rejection process, and the design of a dual tap to the ARTS computer, were a result of this consideration. Because the RSLS Demonstration System development overlapped design and implementation, it was clear that design changes along the way would be inevitable. This lead to the requirement that the use of custom hardware would be avoided wherever possible, and as much of the system functionality would be performed in software using commercial off-the-shelf (COTS) equipment. This proved to be of great benefit throughout the system design, and was made possible by the explosion in computer system performance in the past few years. In the case of the radar interface and certain required improvements to the marine radar, however, custom hardware was required.

1.5 RSLS DEMONSTRATION DESCRIPTION

The RSLS Demonstration System is installed at Boston's Logan Airport (Figure 1-9). The demonstration room is on the 16th floor of the Logan control tower, in the Massport conference room (Figure 1-10). This room provides a clear view of most of the airport's runways and taxiways, allowing a good means to check the operation of the system. This room has several displays showing various aspects of the system operation. A Raytheon Pathfinder radar display shows a one-bit digitized image of the raw radar surveillance. Two monochrome high-brightness displays (manufactured by Orwin Associates) simulate an enhanced ASDE display and a DBRITE display. A third high-brightness display uses backlit active-matrix liquid crystal color technology to show how a color display could be possible in a high ambient light environment.

An airport model board (Figure 1-11) includes architectural models of the terminal buildings, depictions of the runways and taxiways, and a variety of actively controlled field lighting systems. The field lighting systems are simulated using fiber optics, and include the RSLS runway-status lights, runway-centerline and edge lights, taxiway-edge lights, approach lights, taxi-hold lights, stopbars, and so on. These systems are driven actively by an integrated lighting control system, which is interfaced to the rest of the RSLS Demonstration System using an RS-232 interface. Transition from an off-line demonstration of the runway-status lights using the model board to a real field lighting system can in principle be performed by unplugging the model board and plugging in the field lighting controller.

A DECTalk digital voice synthesizer system is used to generate audible voice alerts in response to the alert commands from safety logic. The DECTalk voice quality is probably insufficient for a real tower alerting system, but it is adequate for a demonstration system.

The Demonstration System also has two control displays located in the demonstration room. These are the control displays for the surveillance processing computer and for the sensor fusion and safety logic workstation. The former can also be used to display real-time radar images either before or after clutter rejection. The latter display functions as an additional color ASDE display, although it is not a high-brightness display, and is used to generate and control artificial targets.

The other components of the RSLS Demonstration System are located outside the demonstration room itself. The ASDE radar is located on the roof of the old control tower building, and its associated electronics are located nearby and on the fifteenth floor of the new tower. The ARTS interface hardware is located in the ARTS equipment rooms on the sixth and seventh floors of the old control tower building.

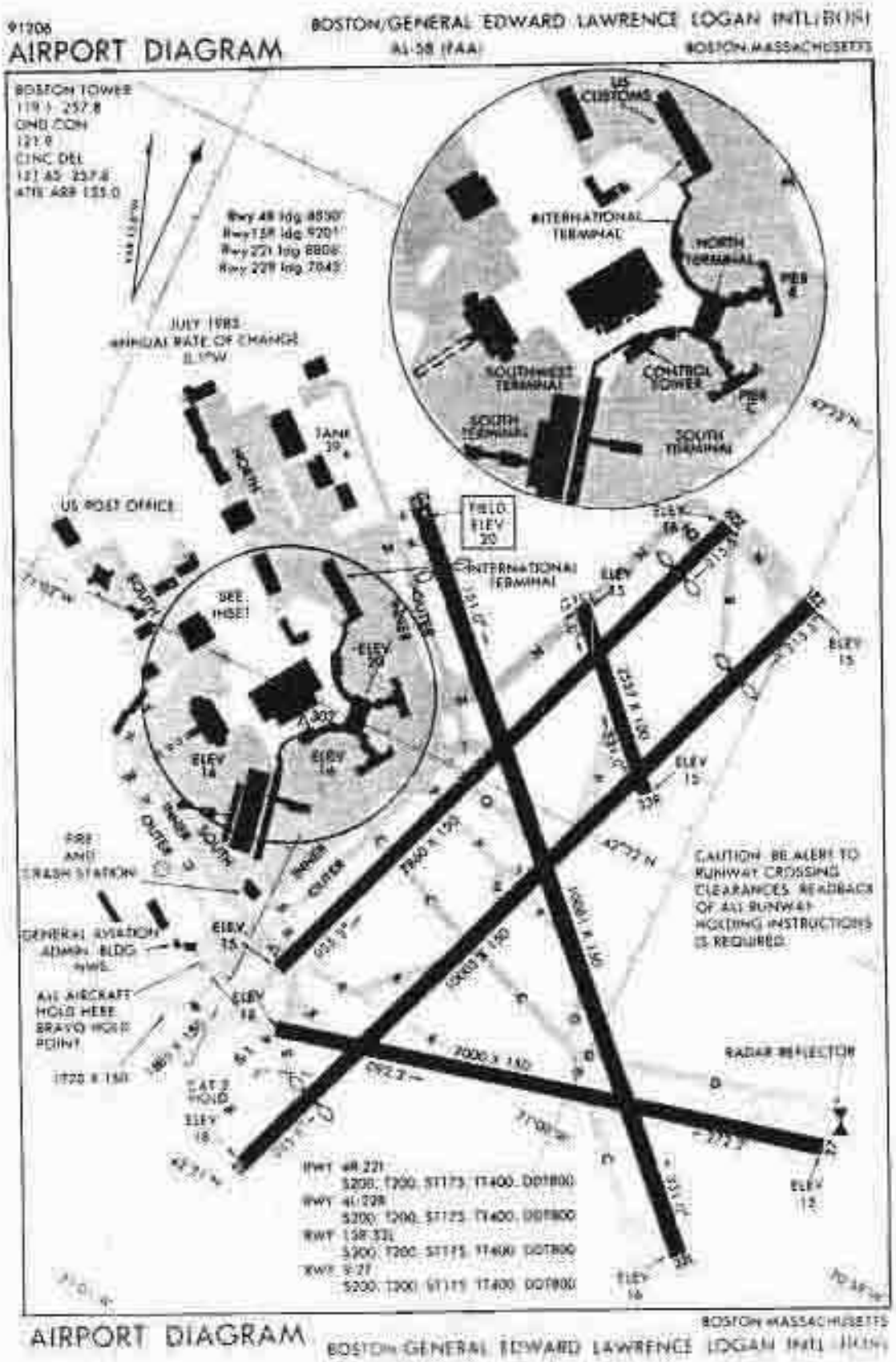


Figure 1-9. Boston Logan airport runway and taxiway map

Figure 1-10. RSLS Logan demonstration room.

Figure 1-11. RSLS Logan demonstration model board.

The computers used to drive the two high-brightness monochrome displays and to receive the information from the ARTS interface are located on the fifteenth floor of the new tower.

Normal system operation includes a startup procedure that takes about five minutes. Thereafter the system is completely functional, and normally functions without operator interaction.

1.6 FUTURE IMPROVEMENTS

The performance of the RSLS Demonstration System can be improved by modification of the system architecture and by modifying the various components of the system. Improvements in the various system components are discussed in this document in the respective sections. Certain system capabilities that might improve reliability were considered out of scope for the research system developed here. These include redundant hardware, automatic built-in test procedures, real-time performance logging, etc.

Other system-level improvements that should be considered for future incorporation into the Demonstration System center on greater information sharing between the various system components. For example, the ASDE processing component could better initiate new tracks for arriving aircraft if it were given information about these aircraft derived from the ARTS interface. Sensor fusion could better fuse tracks through surveillance gaps if it were given the arrival runway predictions computed by safety logic.

Another way of improving the performance of the RSLS system is through improved surveillance. On the system level this would be accomplished by incorporating new surveillance technologies, such as ADS-Mode S, Mode S multilateration, or multiple ASDEs.

1.7 DOCUMENTATION OVERVIEW

The rest of this document presents detailed descriptions of the various components of the RSLS Demonstration System and an evaluation of its performance. The purpose of these descriptions is to present a sufficiently detailed overview of the system to allow a basic understanding of the purpose and behavior of its various components.

Section 2 covers the X-band radar improvements that were necessary to provide radar images with sufficient detail and repeatability to allow effective surveillance processing to produce good tracks.

Section 3 discusses the sophisticated clutter rejection, image processing, and scan-to-scan association techniques used to provide high-quality radar tracks from suboptimal primary radar data.

Section 4 presents the ARTS interface used to provide surveillance on aircraft on approach.

Section 5 discusses the sensor fusion algorithms used to merge surveillance information from two radar sources and to suppress unreliable tracks.

Section 6 covers the safety logic algorithms, which assign target behavior states, determine present and predicted target positions, and drive the runway-status lights.

Section 7 outlines the system software architecture, and discusses how the various processing algorithms were instantiated in operating code.

Section 8 presents an analysis of the system performance in acquiring and tracking targets.

Section 9 presents an analysis of the end-to-end system performance in operating the runway-status lights.

2. GROUND SURVEILLANCE RADAR

This chapter describes the equipment that RSLS uses to monitor surface traffic. This equipment includes a harbor radar that was modified to satisfy the requirements of the safety system, and a recording system that enables one to digitize, censor, and record radar data. Part of the recording system serves as the interface to the radar surveillance processing. This recorded data during play back has the same characteristics as the real-time data, and it enables one to develop and verify detection and tracking algorithms. The radar modifications and their justification are described in this chapter.

2.1 RADAR EQUIPMENT

The surface detection radar used in this project is a modified Raytheon Pathfinder harbor radar, which operates at the X-band. The radar center frequency is 9.375 GHz. We refer to this radar as ASDE-X. The radar is incoherent and relies on short pulses for range resolution. It does not have monopulse nor Doppler capability, and it relies on a narrow antenna beam to obtain azimuthal resolution.

The transmitter and receiver are located on the top floor and the antenna is located on the roof of the old control tower building at Logan Airport. A photograph of the antenna in this location is shown in Figure 2-1. The video signal from the receiver is sent along 450 feet of frequency-compensated cable to the Raytheon display processor, which is located on the fifteenth floor of the new control tower building. A picture of the modified Raytheon display is shown in Figure 2-2. Lines representing the edges of runways are generated by the standard display. This representation can be compared to the runway and taxiway designations at Logan airport as shown in Figure 1-9.

2.1.1 Radar Characteristics

This radar is typically used in harbors to detect ships, obstructions, and channel markers. While it is suitable for that task, it is not used for precision measurements. After working with it, we found that the original radar had deficiencies that would have prevented us from providing surveillance good enough to operate the safety system so that it did not have an excessive number of false alarms and missed detections.

Consequently a series of modifications was implemented to make the radar suitable for our purposes. We increased the scan rate to increase the surveillance rate, we changed the shaft encoder to make the firing point of the transmitter reproducible from scan to scan, we added range-sensitive attenuation to increase the useful dynamic range of the receiver, we modified the receiver to reduce saturation effects, we shortened the transmitted pulse width to increase the range resolution, and we added test equipment to monitor the radar's status. These changes are described in detail in the next section.

These changes were made to the radar so that it did not compromise the performance of the safety system. Each of these changes can be incorporated into the design of a new marine radar without significantly increasing the cost of a standard marine radar. Our experience in working with this radar gives us confidence to say that a good ground safety system can be built using an inexpensive radar as the front end. A block diagram of the radar subsystems is given in Figure 2.3. Table 2.1 gives the configuration of the X-band radar used in the RSLS demonstration.

TABLE 2.1

Characteristics of the Modified Raytheon Harbor Radar Installed at Logan Airport

Antenna height above airport surface	118 feet
Power transmitted	30 kW peak
Power source	Magnetron
Frequency	9.375 GHz
Waveform	CW rectangular pulse
Pulse width	40 nsec
Scan time	1.7 sec
Pulses/scan	4096
Polarization	Electric field horizontal
Horizontal beamwidth to the ± 3 dB points	0.45° one way
Vertical beamwidth to the ± 3 dB points	19° one way
Peak antenna gain	34 dB
First sidelobe level	–25 dB down from the main lobe
Far out sidelobes	< –40 dB down from the main lobe
Noise figure	20 dB
Sensitivity	17 dB S/N for 3 m2 target at 10,000 feet

There are several significant differences between this radar and the ASDE-3, which is the surface detection radar that is currently being installed at the larger airports in the United States. The ASDE-3 operates at the higher frequency of 15.5 GHz. This makes it more sensitive to rain; however, it has frequency diversity and circular polarization to allow the rain clutter to be reduced. The horizontal beamwidth of the ASDE-3 is 0.25°, which enables it to separate close objects at 12,000 feet. The vertical beam profile of the ASDE-3 is narrower than our radar and is shaped to give a return that is almost constant with range. This characteristic also makes the ASDE-3 less sensitive to rain. The ASDE-3 has four times the number of pulses per scan, which enables more pulse averaging, thereby further decreasing the effect of rain clutter.

The ASDE-3 at Logan is located on top of the control tower at a height of 275 feet, which makes it less susceptible to the multipath and shadowing effects that are serious problems with the 118-foot height of the ASDE-X. These problems are described later. The relative positions of the ASDE-X and ASDE-3 are shown in Figure 2-4.

2.1.2 Radar Modifications

To work successfully in the RSLS system, several improvements were made to the Raytheon marine radar. These changes and their rational are as follows:

1) A typical marine-radar antenna has a maximum width of 12 feet. Our system uses an 18-foot harbor antenna to obtain a narrower beamwidth of 0.45 degrees.

2) The original scan period was 2.8 seconds. To provide more frequent surveillance updates, the scan rate was increased to once per 1.7 seconds.

3) To eliminate clutter to the greatest extent possible by using a stored high-resolution clutter map, the pixel areas that are illuminated by the radar in successive scans should be as identical as possible. To ensure this, the analog resolver, which had jitter in the measurement of azimuth position, was replaced by an optical shaft encoder with a digital readout. The ACP (Azimuth Change Pulse), which is used to trigger the transmitter, is derived from this signal. Because of wind loading on the exposed linear antenna, the Pulse Repetition Interval (PRI) can vary by 5% during a scan, yet the azimuth position at which the transmitter fires is the same for each scan. The digital encoder also eliminated azimuthal nonlinearities that initially caused the runways to appear curved.

4) With a digital encoder it is most convenient to make the number of pulses per scan a power of two. To accomplish this, the PRF (Pulse Repetition Frequency) was changed to about 2,400/sec (exactly 4096/scan).

5) There were several deficiencies in the receiver. With the 70 dB dynamic range of the receiver, it is impossible to avoid receiver saturation from the specular returns of objects on the airport surface. The original log-amplifier, which has a slow overload recovery time, saturated before the receiver sections preceding it, resulting in greatly stretched pulses for large targets. This resulted in inaccurate estimates of the centroid position. A buffered tap was placed in the Raytheon receiver before the log amplifier, and a new receiver section added to this tap that corrected this deficiency. Presently, the receiver saturates before the log amplifier. Even though the log-amplifier did not saturate, its bandwidth was such that there was still some stretching of targets due to the time it would take to change levels. A log-amplifier with faster recovery characteristics was procured and installed. This has greatly reduced pulse stretching, thus reducing errors in measuring the centroid of objects.

6) An object of a given size produces a much more intense return as it gets closer to the radar. The original receiver had STC (Sensitivity Time Control) to reduce the return level of closer targets (hundreds of feet). A new STC was added that gives a nearly constant receiver output for a constant cross-section target in the area of most interest, which is the range from 1250 to 5000 feet. This prevents targets of average cross section close to the radar from saturating the receiver, increases the useful dynamic range, and provides more flexibility in designing the processing algorithms.

7) The radar performance deteriorated gradually on two occasions because of the gradual failure of the rotary joint and the T-R (transmit-receive) tube. It took us time to discover and to diagnose these problems because of the limited built-in test equipment. Additional test equipment was incorporated to correct this deficiency. A two-way directional coupler was added in the microwave section that allows us to determine the match to the antenna, from which signal one can assess the health of the rotary joint and the antenna. We can also insert test signals in the other directional coupler port to measure the receiver characteristics to calibrate the system.

8) The display processing unit was moved to the new tower to be closer to the processing equipment and the FAA radar displays. The Raytheon radar is being used as an interim ASDE while the ASDE-2 is replaced by an ASDE-3. The increase in distance between the transmitter and the display processor required the addition of buffer amplifiers and frequency-compensation networks to preserve the pulse shape.

9) The transmitted pulse length was shortened from 60 to 40 nsec to improve the range resolution to 20 feet, which is the same as that in the ASDE-3.

2.2 RADAR PERFORMANCE

The radar that we are using for airport surface surveillance was designed to detect ships and buoys on the water. We are mainly using it to detect airplanes on paved surfaces. The radar performance in the airport environment can be significantly improved over the present system by taking advantage of the different requirements for these two tasks. This is briefly addressed in the section on possible radar improvements.

2.2.1 Calculated Performance

A computer program was developed to predict the performance of the radar at various rain rates. This code calculates the interference level that is a sum of system noise and rain clutter and compares it to the predicted target return. The rain clutter in a given pixel is determined by calculating the rain clutter of any rain that is at the range of the pixel. This return is multiplied by the rain attenuation and antenna gain to that volume. The rain return is the sum of the returns from four combinations of outgoing and incoming paths: the direct path in both directions, the direct outgoing path and the incoming path that bounces off the ground, the bounced outgoing path and the direct incoming path, and the bounced outgoing and incoming paths. Only the direct paths are used to determine the target return since the ground bounce may be obscured by buildings or other targets in the foreground. For this reason the calculated signal-to-clutter plus noise ratio is a conservative estimate. On average when the target is in the clear and the ground is a good reflector, then the ratio is six times higher.

Figure 2-5 shows plots of the signal-to-clutter plus noise ratios of the X-band radar for various rain rates for a target that has a cross section of 3 square meters. Also plotted on this graph is the performance of the ASDE-3 system for the same cross section when the rain rate is 16 mm/hr. The ASDE-3 has a significantly higher signal-to-clutter plus noise ratio than the ASDE-X at longer ranges. For real targets the difference is not as marked because the cross section of a complicated target is 3 to 6 dB less in circular polarization than in horizontal polarization. The major cause of the difference in performance is the circular polarization system of the ASDE-3 radar; the rain return is at the incorrect circular polarization for the receiver. This reduces the rain return by the 17 dB cancellation ratio of the receiver.

The signal-to-clutter plus noise ratio is not the only important consideration in evaluating system performance; one must also determine what ratio is required to detect the target. Because the ASDE-3 has frequency diversity and the beamwidth and scan rate are such that one obtains returns from 12 adjacent PRIs from a point target, adjacent PRIs can be averaged to reduce the noise variance. With no averaging, one requires for a detection of a single pixel a 21 dB signal-to-clutter plus noise ratio to obtain a 0.9 probability of detection with a false alarm rate of 10^{-6}. With the averaging across the beam in the ASDE-3, this ratio only has to be about 8 dB. Because the ASDE-X does not have frequency diversity, it has an additional 13 dB performance disadvantage in detecting a point target in a single PRI. Fortunately this is not the way the system is operated. In the system the threshold is set low to achieve a high probability of detection. However, this incurs a high false detection rate. The subsequent processes of discarding detections unless they also occur in adjacent PRIs at the same range, discarding targets unless they are above a certain size, and requiring tracks to be correlated from scan to scan reduces the false detections. The net result is that the required signal-to-clutter plus noise ratio for comparable detection statistics on airplanes is just a few dBs more in the ASDE-X than in the ASDE-3.

2.2.2 Disturbances in the Radar Data

In looking at the radar display in Figure 2-2, one is immediately struck by the large amount of clutter that is present. Since this data was taken in clear weather, this clutter is not due to rain as the clutter described in the last section. Most of the clutter is due to ground, water, and building returns. The amplitude of the clutter changes from scan-to-scan and has significant spatial inhomogeneity, thus giving the appearance of speckle. The radar returns have additional deficiencies, which are as follows:

- Radial streaks that exist for one PRI are caused by other marine radars that are operating on the same frequency. There can be several of these radial streaks each scan.
- Since the radar is not very high, objects can cast a long shadow that is radially outrange. This shadowing effect can significantly decrease the return amplitude from an object by either obscuring the direct return from that object or by shielding one of the ground bounces that comprise the normal return. This problem is particularly noticeable for planes that are moving in a line on radial taxiways, runways, and run-up pads. At Logan airport this problem occurs on taxiways Sierra (S) and Charlie (C), which is directly in front of the radar, and on the run-up pad area on Charlie. These taxiways are depicted in Figure 1-9.
- Planes that are far from the radar on taxiways such as November (N) can appear to merge because of the finite azimuthal beamwidth. This is particularly troublesome because planes that are on the short transversal part near the end of N can move onto the runway in a short time.
- Some planes such as the MD-80 have large returns from the nose and tail areas and very little from their midsection. These planes appear as two distinct objects, and the surveillance system has to declare them as a single object in order for the safety system to work correctly.

- Aside from real vehicles in the movement areas, there are false returns of significant amplitude caused by multipath. Each of these erroneous returns is caused by a specular reflection off the real object that is then specularly reflected from a second object back towards the original object. This signal is then reflected back towards the radar and appears as a target that is radially outbound from the real object. The low height of the antenna exacerbates the problem. The number of these false images often exceeds the number of real objects. These false objects can have movement characteristics similar to real objects. Eliminating these objects is a significant challenge. We have identified several places where there are buildings at right angles to each other that form a corner reflector for the radar signals. These cause some of the multipath signals, and they can be eliminated by calculating the expected position of the multipath return. Unfortunately, a significant number of these false targets are caused by vehicles in-range of the range of closest surveillance, and these cannot be eliminated by using a fixed stored file of multipath reflectors. Because specular reflections are necessary to get a significant multipath return, the objects typically cannot move much before the multipath amplitude is greatly reduced. This observation is the basis for the most effective algorithms that are implemented to eliminate multipath.
- The lights and their supports for the approach lighting system on the extended runways have a very large cross section that can obscure all or part of the return from an airplane.

2.3 DATA RECORDING AND PLAY BACK EQUIPMENT

The radar returns are detected with a receiver that has a logarithmic amplifier to compress the difference in output voltage between returns of different amplitudes. The receiver output is digitized with an 8-bit A/D converter that samples the signal at 42 MHz. A capability to record this data and to play it back at real-time rates through the SGI (Silicon Graphics Inc.) multiprocessor has been developed. This capability facilitates the development of algorithms and their debugging, and allows one to store interesting scenarios. A block diagram of the recording system, which is called LARS (Lincoln ASDE Recording System), is given in Figure 2-6.

LARS is controlled by a Sun workstation board by a menu-driven program, which provides an easily learned interface to control data recording. Operator-selected options are downloaded to the Mizar 7130 computer, which controls the details of the recording process. Most of the digitized data is of no interest to the ground surveillance system. The transmitter fires 2,400 times a second thus giving an ambiguous range of about 35 nmi. The major interest is with data that is within 1.5 nmi of the radar; therefore, a major reduction in the average data rate can be achieved by discarding most data at longer ranges. An exception is the data from the area several thousand feet along the runway approaches; this ASDE data allows earlier fusion with the ARTS data. The 42-MHz data rate is reduced by a Lincoln-Laboratory-built censoring-map board that selects sections of this steady stream of digital data, formats it, and adds time stamps.

The censoring reduces the data rate by eliminating data from areas of little interest. A censoring map must be developed for individual airports. The data selected by the censoring map at Logan airport is shown in Figure 2-7. Data in the black areas around the airport and in the infield areas within the airport confines are not recorded or processed. The censored data is put into two 32-MByte ping-pong memories on a VME bus by a Mercury MC3200 processor. When the Mizar 7130 computer on this bus detects that a buffer is full, it initiates a data transfer to tape using a Ciprico SCSI (Small Computer System Interface) controller card, which is also located on the VME bus. This scheme is used to reduce the original 42 MByte/sec data stream at Logan airport to one that is 660 kByte/sec, which is below the

2.2 MByte/sec recording limit of our SCSI-interfaced Honeywell Very Large Data Store (VLDS) tape system and the processing capability of our real-time system. This censoring board is the only hardware in the processing system that is not COTS (Commercial-Off-The-Shelf) equipment.

This censoring board is duplicated in the VME bus of the SGI multiprocessor to allow radar data to be accessed at real-time rates. This real-time censored data can be handled by the real-time processor, which is a multi-processor. The processing system has the flexibility to access data supplied at real-time rates either directly from the radar after having the data rate reduced by the process described above or by playing back data stored on the VLDS tape.

In addition, we have developed a capability to play back data at reduced rates. The data on the VLDS tape can be transcribed onto Exabyte tapes that can be played back at rates up to 500 Kbytes/sec. These units are used because the tapes can be played back many times without deteriorating, the units are inexpensive, and they are easily controlled by the Sun workstations used in software development.

2.4 POSSIBLE RADAR IMPROVEMENTS

The performance of the radar system can be significantly improved by making changes that should not add significantly to the cost. As pointed out in Section 2.2.1, shadowing is particularly troublesome on radial runways. If the radar were sited so that it was higher and did not look radially along a runway or taxiway, then this problem would be reduced. The low height of the antenna also contributes to the multipath problem. A higher siting or a location where there are not many objects to reflect the signals would alleviate this somewhat. An antenna that transmitted one type of linear polarization and received the other would tend to reject the strongest multipath signals since they tend to come back with the same linear polarization.

As pointed out above, because of processing that occurs after detection, the radar can work with about a ten dB signal-to-noise plus clutter ratio and still have good detection statistics. This can be further improved if PRI averaging could be implemented. This requires that the clutter in adjacent PRIs be uncorrelated. For rain this can be achieved by frequency diversity and by range staggering.

An even more significant eight dB increase in performance can be achieved by using an antenna with a different polarization than the horizontal polarization of the present system. The rain clutter is particularly troublesome because with the small vertical antenna aperture the beam is wide vertically. Rain clutter comes in from many angles. Since reflections from the ground at significant angles change the sense of the circular polarization, a circularly polarized system with a large elevation beamwidth would not have as significant an advantage over the present system as one with a smaller elevation beamwidth. Vertical polarization has a null in the ground reflection at the Brewster's angle, which significantly reduces the rain clutter return that is bounced off the ground, however, it does have a significant return from the clutter in the main beam. The cross section of complicated targets like airplanes is about 3 dB higher. in vertical polarization than in circular polarization. The net result is that for these systems the performance of a vertically polarized system and a vertically polarized system are about equal. Figure 2-8 shows the performance of a vertically polarized system with the same antenna gains as the present system. The noise figure has been lowered to a level that is achievable with modern receivers. Note the significant improvement in rain performance over the present system, which is due mainly to the use of vertical polarization. The improved noise performance only significantly improves the performance in rain-free weather at long range.

Ground and rain clutter are not the only spurious returns that affect the detectibility of targets. Because of the presence of many surfaces on the airplanes and terminal structures that can act as specular reflectors one finds a great deal of multipath in this environment. A system that transmits vertical polarization and receives horizontal polarization not only has good rain rejection but it should also significantly reject multipath returns., For these reasons the use of this system should be further explored.

Decreasing the pulse width to 40 nanoseconds required us to increase the bandwidth of the receiver. The IF frequency of the receiver is at 45 MHz. We did not change this because of the requirement to be compatible with the Raytheon display system. Because the desired bandwidth of 25 MHz is a significant fraction of the IF frequency, it was not possible to obtain a log amplifier with ideal characteristics. A receiver with a higher IF frequency should be specified.

Figure 2-1 Radar antenna and view of Logan airport.

Figure 2-2. Radar output shown on modified Raytheon display

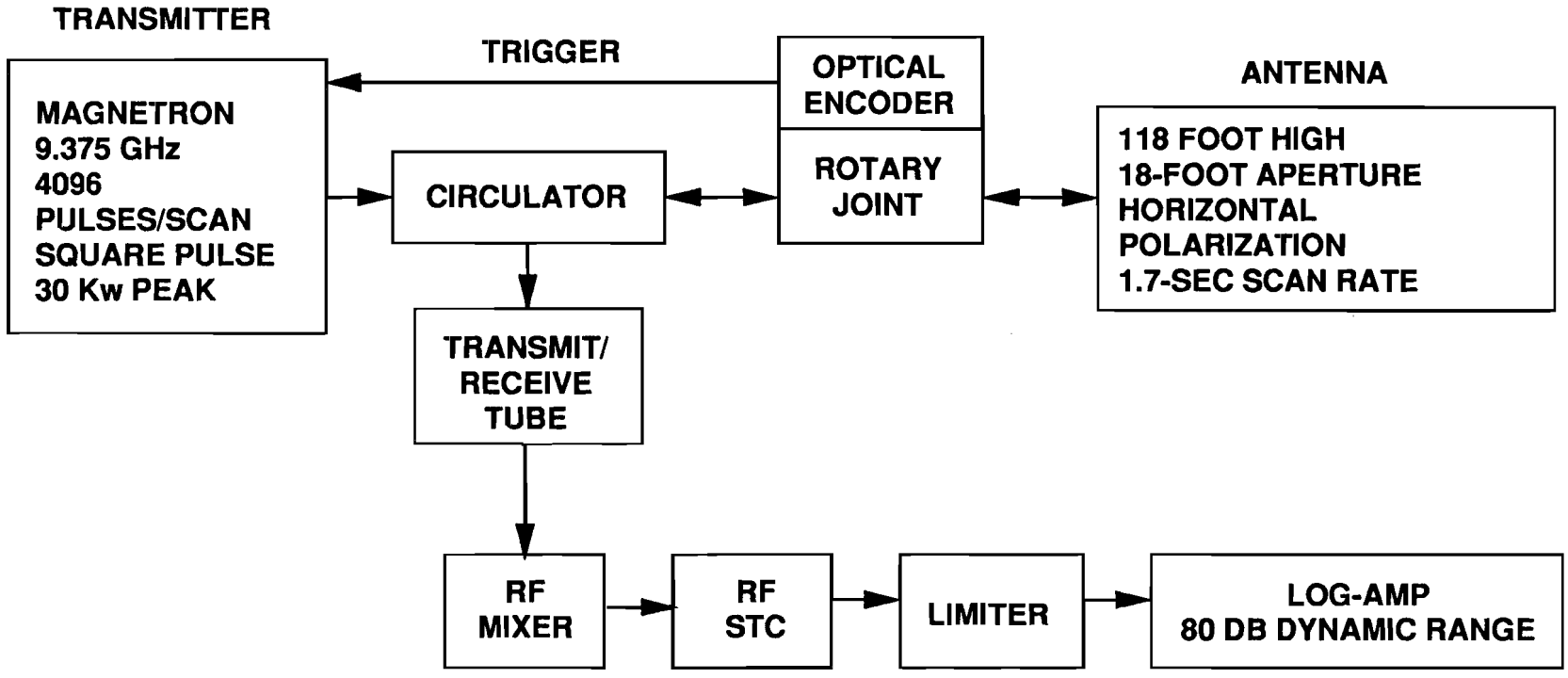


Figure 2-3. Block diagram of radar.

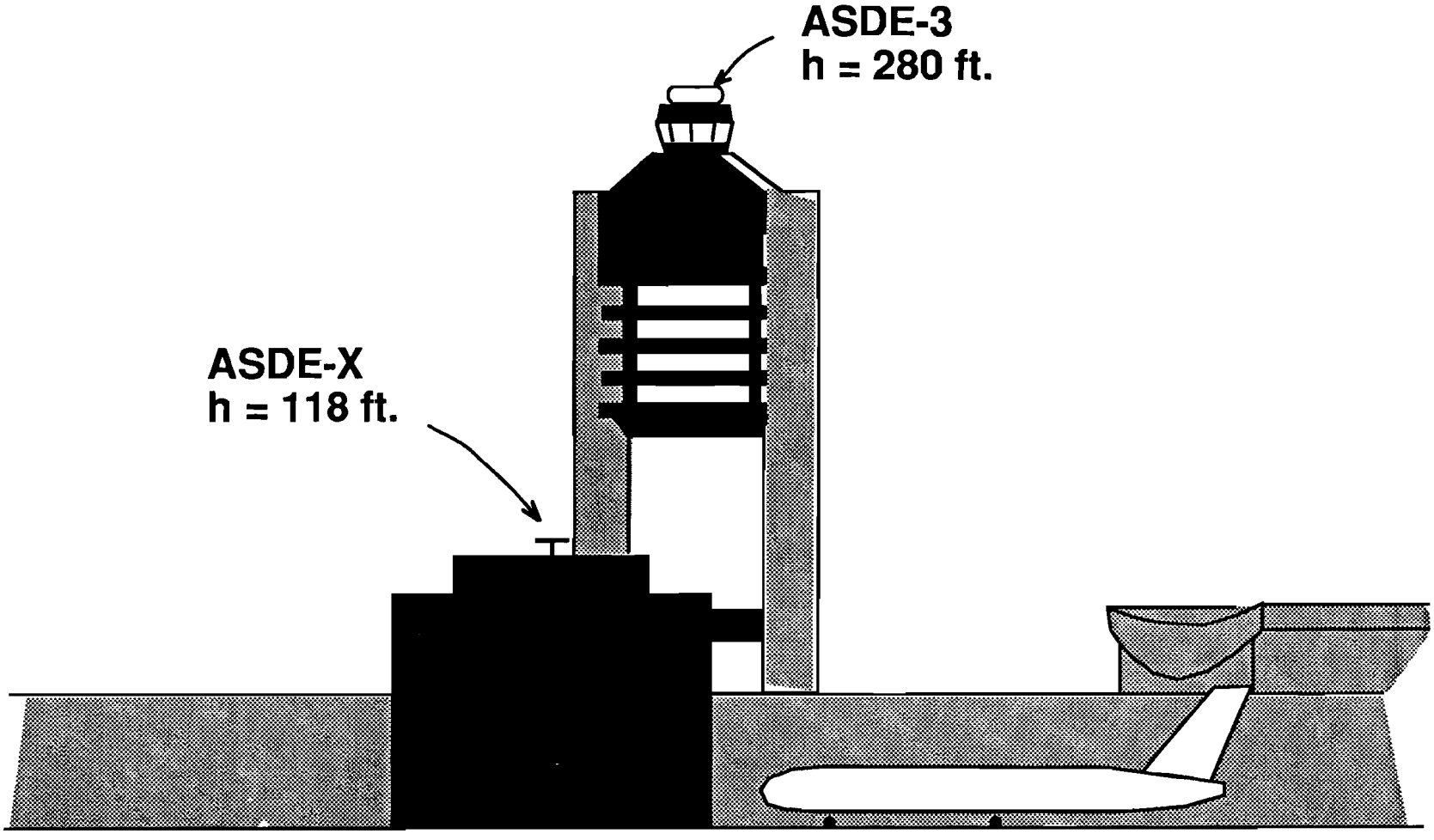


Figure 2-4. Relative positions of the ASDE-3 and ASDE-X radars.

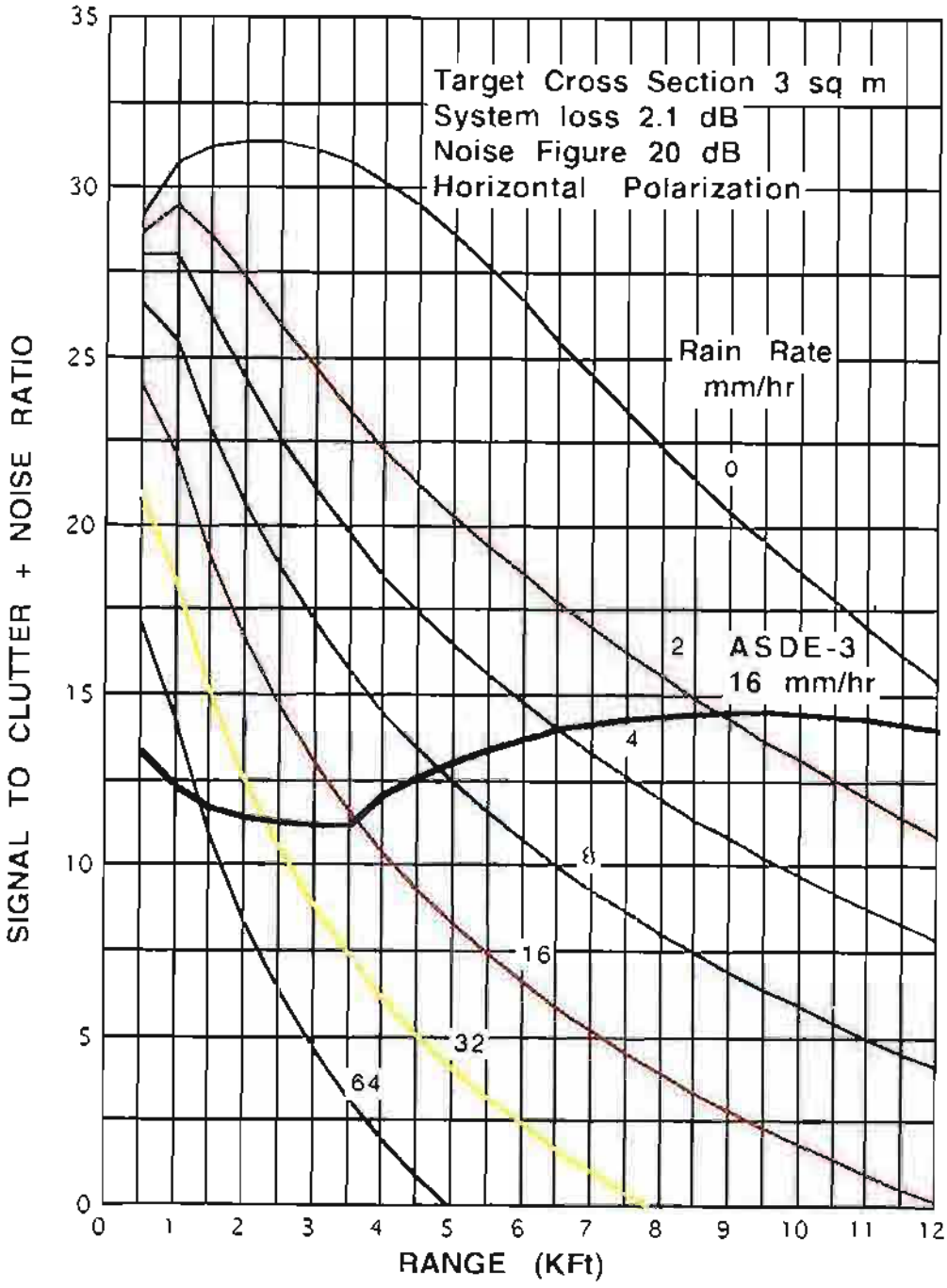


Figure 2-5. Performance of the ASDE-X versus range and rain rate.

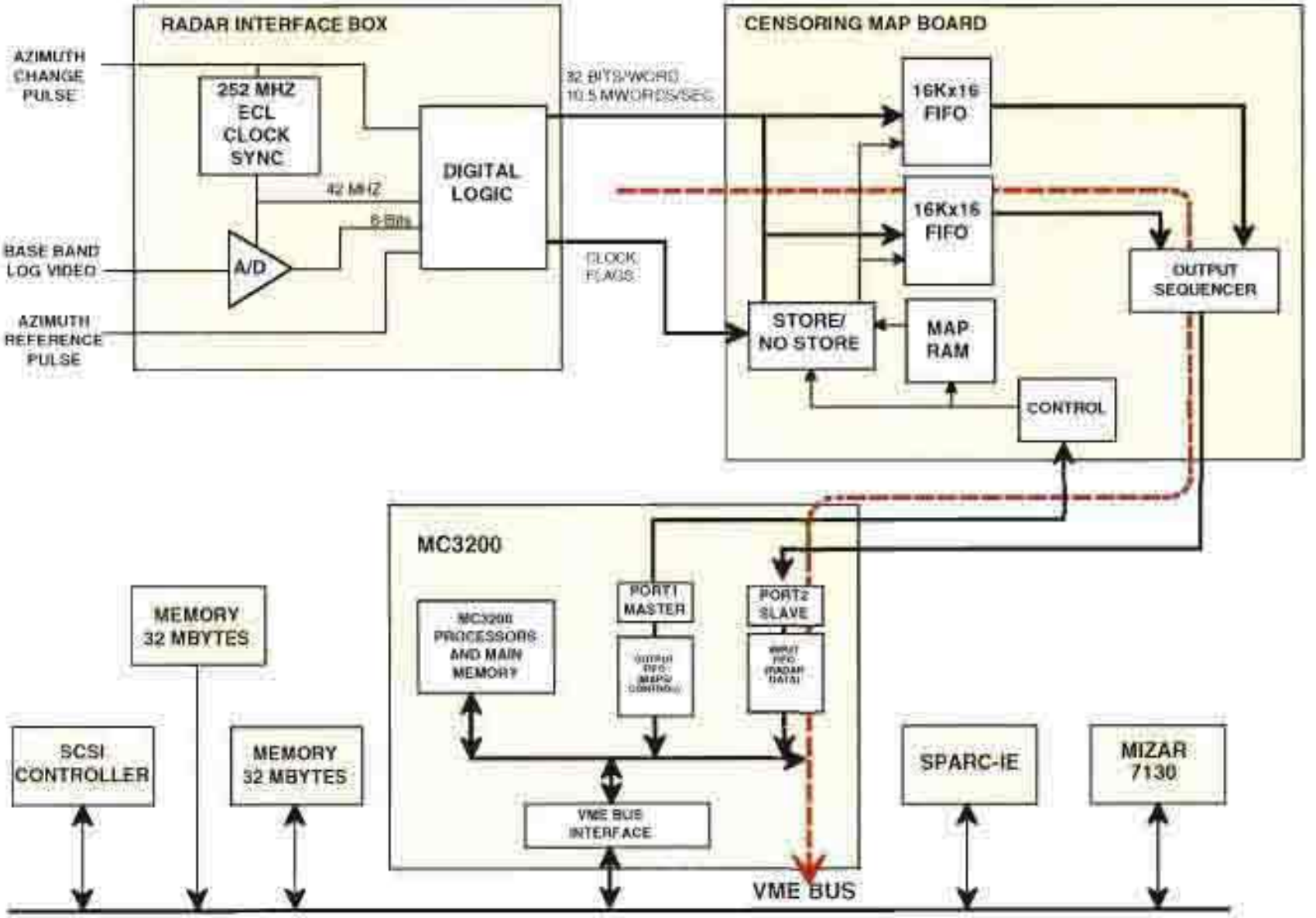


Figure 2-6. Block diagram of LARS (Lincoln ASDE Recording System).

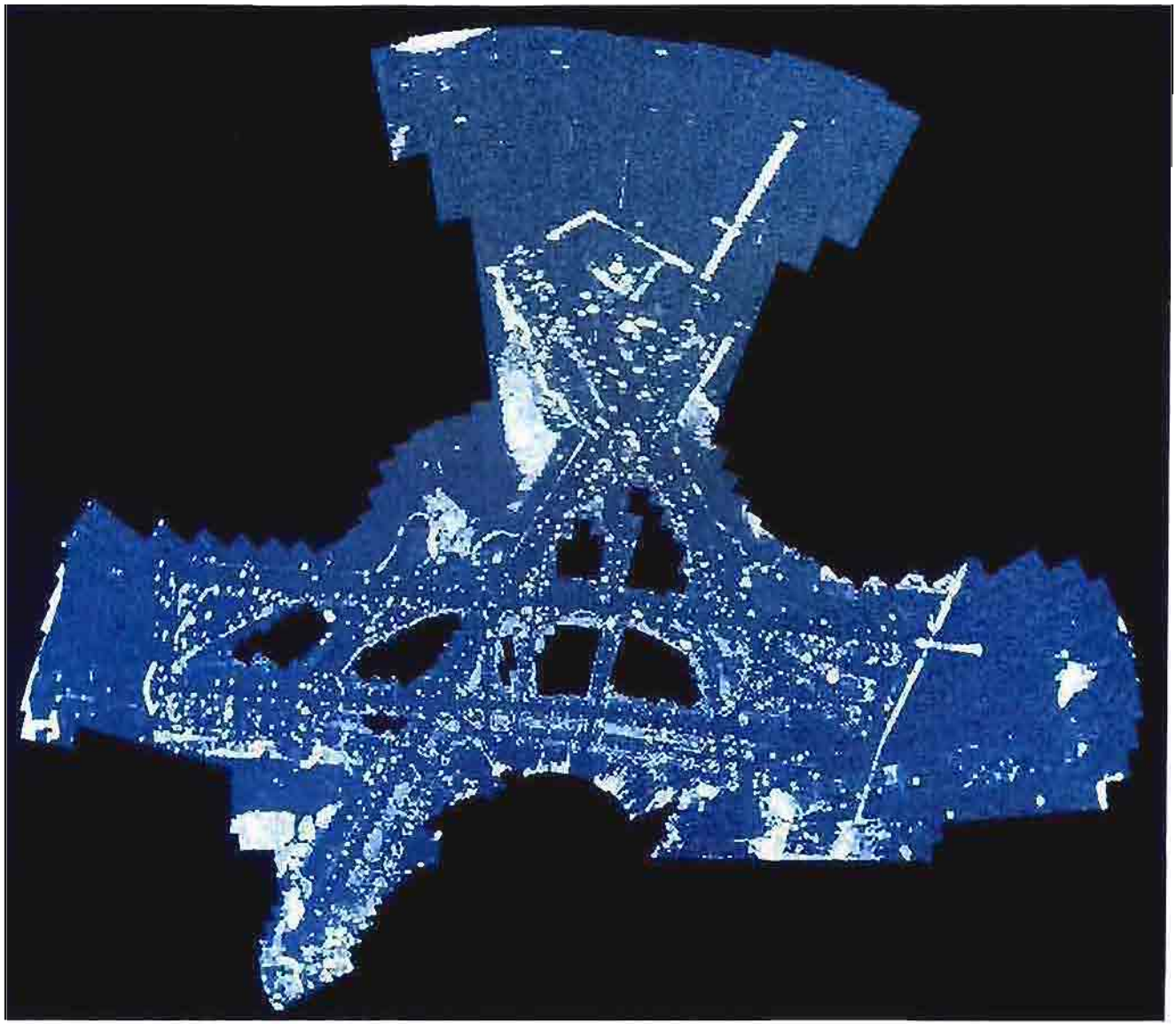


Figure 2-7. Censoring map of Logan airport.

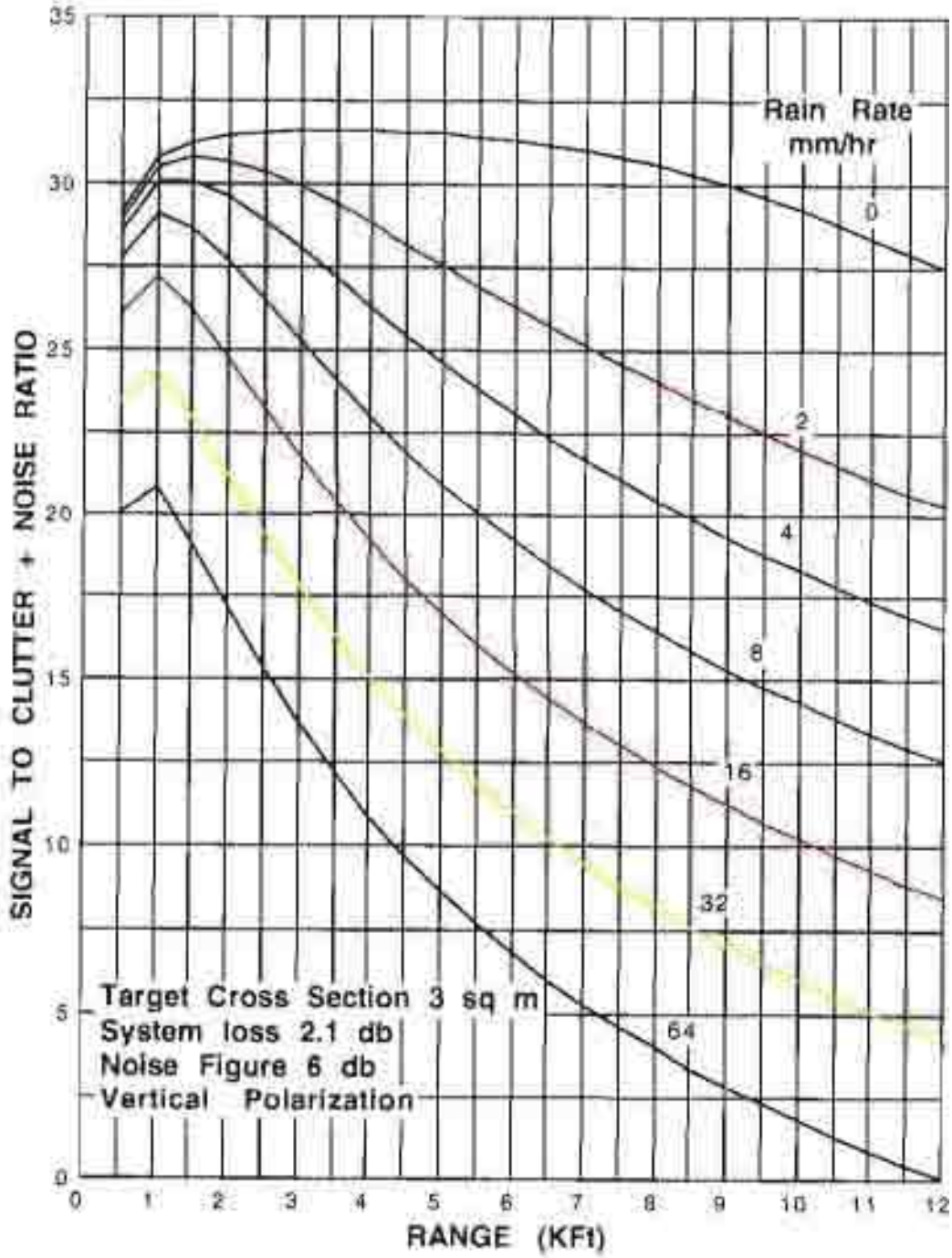


Figure 2-8. Performance of a vertically polarized radar.

3. SURVEILLANCE PROCESSING

In the last chapter the processes of detecting, digitizing, and censoring the ASDE radar data were described. This censored data is entered into a buffer memory and is read out at a reduced rate into a multi-processor computer – an eight-processor SGI (Silicon Graphics Inc.) computer. Here, in a combination of parallel and serial architecture, the surveillance processes of rejecting clutter and interference, eliminating false objects, and grouping returns from the same object are performed. Also, by correlating returns from scan to scan, tracks and other object features are calculated and sent to the sensor fusion. This chapter describes the hardware and algorithms used to accomplish these tasks.

The system operates very well in a real-time environment. Because of time constraints in developing the system, many of the system parameters have yet to be fully optimized. There are additional proposed changes that would improve system performance, which are described in Section 3.3.

3.1 OVERVIEW OF THE SURVEILLANCE ALGORITHMS

The most difficult tasks in developing the surveillance system have been generating and refining suitable algorithms and in implementing them in a real-time system. We have tried to keep the number of airport-specific parameters to a minimum to make the code easily portable. To move to a new airport one would have to input the location of certain types of areas. We think that there would be very little tuning of parameters in this process. The system has to be tested at airports other than Logan to verify this.

The tasks of rejecting clutter and establishing tracks are complicated because of the severe clutter environment found at Logan airport. The object of surveillance processing is to achieve a high probability of detection while maintaining a low probability of false detection. To do this we take the somewhat unconventional approach of using a low threshold to detect radar returns. This results in a very high probability of detection and a high probability of false detection in the initial clutter-rejection stage.

The false objects are eliminated by a series of processing steps each of which decreases the probability of a false detection without significantly affecting the probability of a correct detection. The false-detection rate is reduced by requiring some overlap of the pixels contained within the threshold crossings between adjacent Pulse Repetition Intervals (PRIs), by morphological processing, by using size criteria, by correlating tracks with those from previous scans, and by using the location of a detection and its amount of movement.

In this chapter, the steps for each of the surveillance processes are first listed, and then some details of their implementation are described. Greater detail is provided in the chapter on system software. Figure 3-1 shows a block diagram of the processes and the processors assigned to them.

Most radar clutter is removed in the initial clutter rejection processing. Radar interference and random noise are eliminated in the PRI correlation processing that occurs in the clutter-rejection processors. Out of this process come threshold crossings; a contiguous group of pixels from several PRIs within the threshold crossings is referred to as a component. The process of connected components

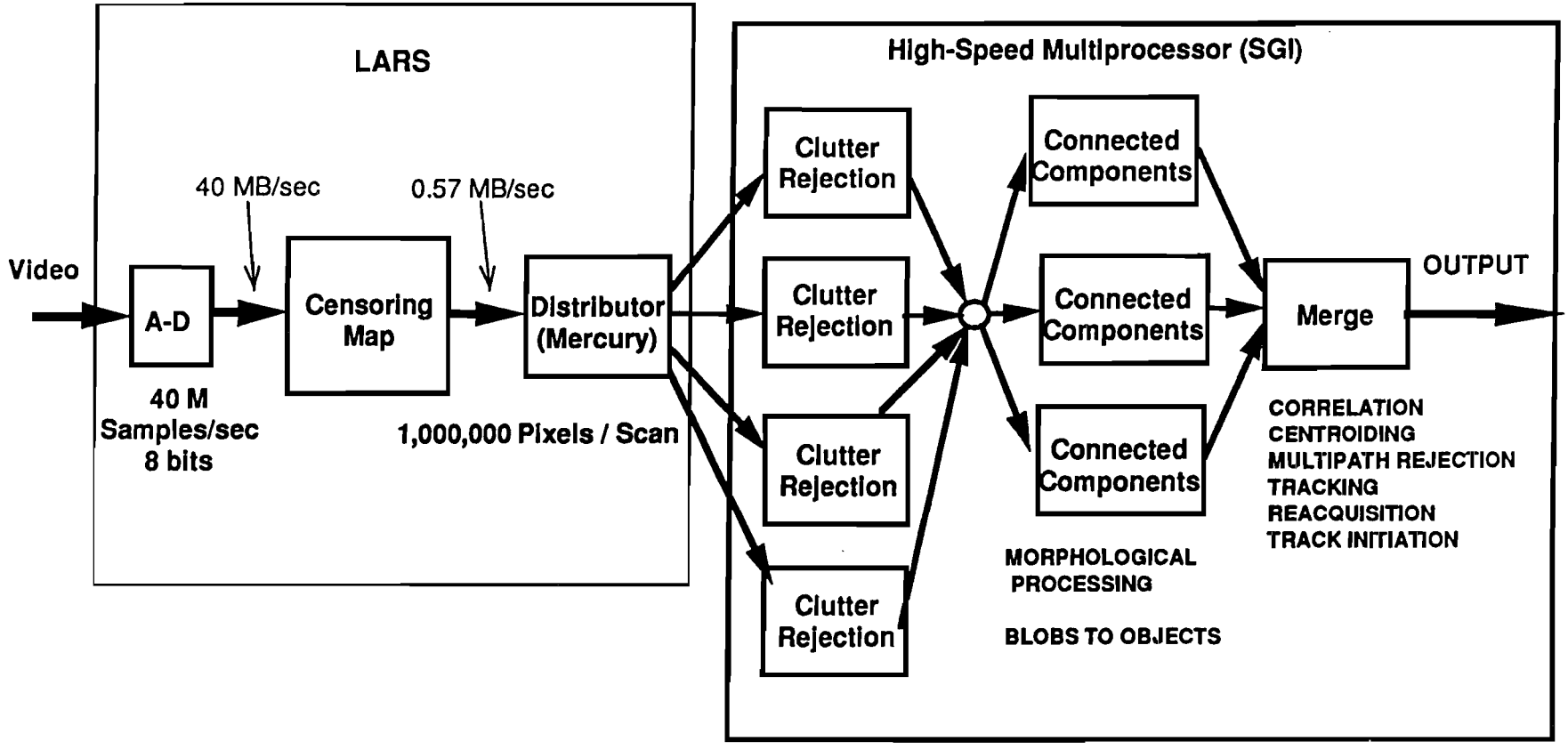


Figure 3-1. Surveillance processing block diagram.

identifies regions that belong to the same component. The morphological or shape processing eliminates small components, cleans up images, and associates components that are part of the same object. Scan-to-scan correlation associates objects with those present in previous scans. This enables tracks to be updated and new ones to be generated. These files and other object features are sent to the safety system.

Since both the clutter-rejection and the connected-components modules each use more than one processor, each of which processes part of the airport surface, special care is necessary to merge components that are close to the processing boundaries. This requires special treatment of the areas that are close to the surface wedge boundaries.

3.2 PROCESSING SOFTWARE AND EQUIPMENT

It is only because of the rapidly decreasing cost of processing that it was possible to build this system at reasonable cost with COTS equipment. Even with the fast processor, we have had to implement some algorithms in an approximate fashion to allow real-time operation. Because of computer advances since the system was procured, one can now purchase a more capable processor at lower cost, which would allow better algorithms to be implemented.

The surveillance software is written mainly in the object-oriented language C++, and the interprocessor communication software is written in both C++ and C. The capability to develop the software on many platforms and to test it with real data reduced the developmental time. This environment enabled us to evaluate the performance of the various algorithms and to develop the links between the software modules.

For surveillance processing we have two SGI multiprocessor chassis each of which can hold two, four, six, or eight R-3000 processors. Eight processors, which have a maximum processing capability of 286 MIPS, are used for the Logan demonstration; two processors in a chassis and the Indigo workstations are retained at Lincoln Laboratory for additional software development, debugging, and non real-time data processing. We have four HP computers, one is at Logan; the other three are at Lincoln Laboratory. The SGI and HP computers at each of the two locations are linked by Ethernet. The two locations communicate via a high-speed modem link that can transfer software between computers, thereby allowing tight configuration control and remote control of the processors.

The Logan-airport system can be operated to run either in real time or to use recorded data. We have recorded data when the airport was in different runway configurations, when there were periods of light, medium, and heavy traffic, and when it was raining and snowing to assess system performance in these conditions.

Data is recorded or real-time tests are performed as required. Working with the Raytheon radar has been complicated because of its use at Logan as a replacement for the ASDE-2 until an ASDE-3 is installed and working. This use was started in April 1992 and was originally expected to end in October 1992. It appears that this mode will be used until the summer of 1993 due to delays in delivering the ASDE-3 radar and making it operational. An agreement was worked out between the FAA and Lincoln Laboratory that allows us to record data from the radar. Neither our data recording nor the operation of our processing system in real time affects the operation of the Raytheon radar as an ASDE. Any changes

we want to make to the radar have to be approved by FAA and Raytheon personnel and must be done at times when the radar is not being used by the controllers. Raytheon has a maintenance contract with the FAA to perform preventive maintenance and to repair the radar when it fails. After some initial difficulties, this arrangement has not caused an undue burden on the RSLS program.

To operate in real time, the surveillance code has to run on eight processors, which required careful attention to interprocessor communications. An existing interprocessor communications protocol, which had been developed for other projects, was modified to transfer information between the various processors.

As shown in Figure 3-1 there are three sets of processors to apply the surveillance algorithms to the radar data. When data is transferred from one set of processors to another set, it must first be converted from the internal object-oriented format to a serial format. This process, which can be computationally expensive, is called flattening. When data is received by the succeeding processors, it undergoes an unflattening operation that puts it back into an object-oriented format.

An existing interprocessor communications package was chosen to control the transfer of data between processors. It was modified for this application and it comprises a protocol, server-client, and C++ layers. At run time, the user specifies the channel and ports for data transfer. The package can use shared memory, Ethernet, files, etc. It efficiently handles all details of the transfer in a real-time environment. It operates on both the SGI and Sun computer systems. The C++ layer does the flattening and unflattening operations.

3.3 SURVEILLANCE PROCESSES

The radar data undergoes a series of processes to produce tracks and object features for the safety system. This section describes these processes

3.3.1 Clutter Rejection

This initial stage of processing rejects most of the clutter in the radar return and forms a binary image. Figure 2-7 shows the censored raw radar data that is to be processed. The return amplitude for each pixel is compared to a stored threshold value for that pixel; the value in the binary image is set to zero if the radar return does not exceed the threshold and to unity if the threshold is exceeded. The thresholds are set fairly low so that many pixels exceed it.

The algorithmic functions that are performed by the clutter-rejection process are as follows:

1) Initialize the values in the clutter map during the initial scans when the system is first turned on.

2) Determine the threshold level for each pixel from the stored clutter values. The threshold is set to the mean plus some constant times the standard deviation.

3) Compare the radar data that is returned from a transmitted pulse to these levels and record the radial position both at which the data first exceeds the threshold and when it drops below the threshold. These two values constitute a run. The list of the positions of the threshold crossings is called a run-length table.

4) Examine the runs in adjacent PRIs. If there is not an adjacent run that has some of the same range values, then discard the run.

5) Pass the remaining threshold crossings in the form of a run-length table to the next processing stage.

6) Update the statistics in the clutter map at the airport positions where a run does not exist.

The motivation and details of the implementation of this clutter-rejection algorithm are now described. The implementation in this system is different from that of the ASDE-3. The clutter has been found to vary greatly with position on the airport surface. In the ASDE-3 there are constant threshold values for different regions of the airport surface, where each region comprises many pixels. If the clutter is non-uniform in a region, not only does some clutter break through, but also the probability of detection is reduced in some areas. Since the ASDE-3 was designed, memory and computational costs have decreased significantly to the point that one can reduce the size of the clutter region that has the same stored clutter level to a single pixel without incurring a significant cost. That has been done in the RSLS system.

When the system is turned on, the clutter means and mean sum squares are determined by using a decaying memory filter. The threshold is initially set at the maximum value for non-movement areas and at a level well above the clutter and noise values on the movement areas but well below the maximum value. Training of the clutter statistics takes place for those pixels that do not exceed the threshold. The decaying memory filter has a weight of 1/6 for new information and 5/6 for old information. After 35 scans, which takes about a minute, the clutter statistics have converged to the correct values for areas in which no objects are present. The clutter statistics of those values are updated for a long enough period of time so that most vehicles have moved from the spot they occupied at the start of the initiation process. This is done to ensure that correct clutter values are obtained for the runway movement areas and for regions that may have been shadowed by these objects.

If an object does not move until after the training period, then at the time of movement the clutter statistics will train to the correct value. It should be noted that if an object stops on the airport surface, then it will always exceed the threshold. Training is suppressed for areas that are above the threshold; therefore, the track of an object will not be dropped no matter how long the object remains stationary.

From the stored mean and variance in the clutter map a threshold value is calculated for each pixel. The threshold value is,

$$Threshold = Mean + K\sigma,$$

where σ is the standard deviation, and K is a constant, which is presently set to 5. There are maximum and minimum values for $K\sigma$ used in this formula that are stored parameters. The parameters to determine the threshold value has yet to be fully optimized.

The incoming data is compared to the threshold value and if it is exceeded, then a run is started. The run ends when the threshold is no longer exceeded. Only the starting and ending positions of the runs are stored. Runs are used because they are a compact representation of the data, and more importantly, because many of the algorithms that operate on the data are very efficiently implemented with this data representation. The thresholds are set fairly low, which results in a large number of detections per scan. This low threshold ensures us of a high probability of detecting a real object. The many false objects detected by this process are eliminated in the subsequent processing.

To operate in real time one must be concerned with load balancing and processing latency. The processing described above is the same for each pixel and repeats every scan. Four processors are assigned to clutter rejection. Because of the repeatability, specific parts of the airport are assigned to a given processor. The data is fed into each processor in a data block of 128 contiguous PRIs. Each of these blocks corresponds to a surface wedge on the runway surface. Because for part of the scan the radar looks at areas not associated with the airport surface and approaches, only 22 of these radial surface wedges are required to perform surveillance for the airport. Each surface wedge is time stamped with the time the radar illuminated that surface wedge.

The next processing step rejects isolated noise spikes and interference from other radars whose PRI is typically different from our 415 microseconds PRI. The azimuthal beamwidth, scan rate, and PRI are such that a point object that exceeds the threshold by at least three dB is detected in five contiguous PRIs. If a run does not have a run from an adjacent PRI on either side that overlaps in range, it is discarded since it does not have the characteristic of a real object. Figure 3-2 shows an image of the data after this processing stage. This process is performed here for computational efficiency since the first stage in the morphological processing would also reject these pixels. The remaining runs are sent to the connected-components processors.

The clutter means and mean sum squares are updated with data from the most recent scan by using a decaying memory filter. The time constant to update the clutter values is short and currently is six scans. For the data examined so far, the time constant is short enough so that when it starts raining the stored clutter values can rise fast enough to keep the detection statistics almost the same, even with the raised clutter values.

3.3.2 Connected Components

The runs remaining after interference rejection are passed to the connected-components processors. The images now have a binary amplitude. Some morphological processing is used in this module since it is an efficient way to eliminate small objects and to control processor loading. The algorithmic functions that are performed during the connected-components process are as follows:

1) Read the runs from the clutter-rejection module.
2) Perform the morphological opening operation.
3) Form a chain code of the outline of a component from the clustered adjacent runs.
4) Subtract the runs corresponding to that component from the global set of runs.
5) Repeat the process until all runs are assigned to individual components.
6) Calculate features such as the area and centroid.

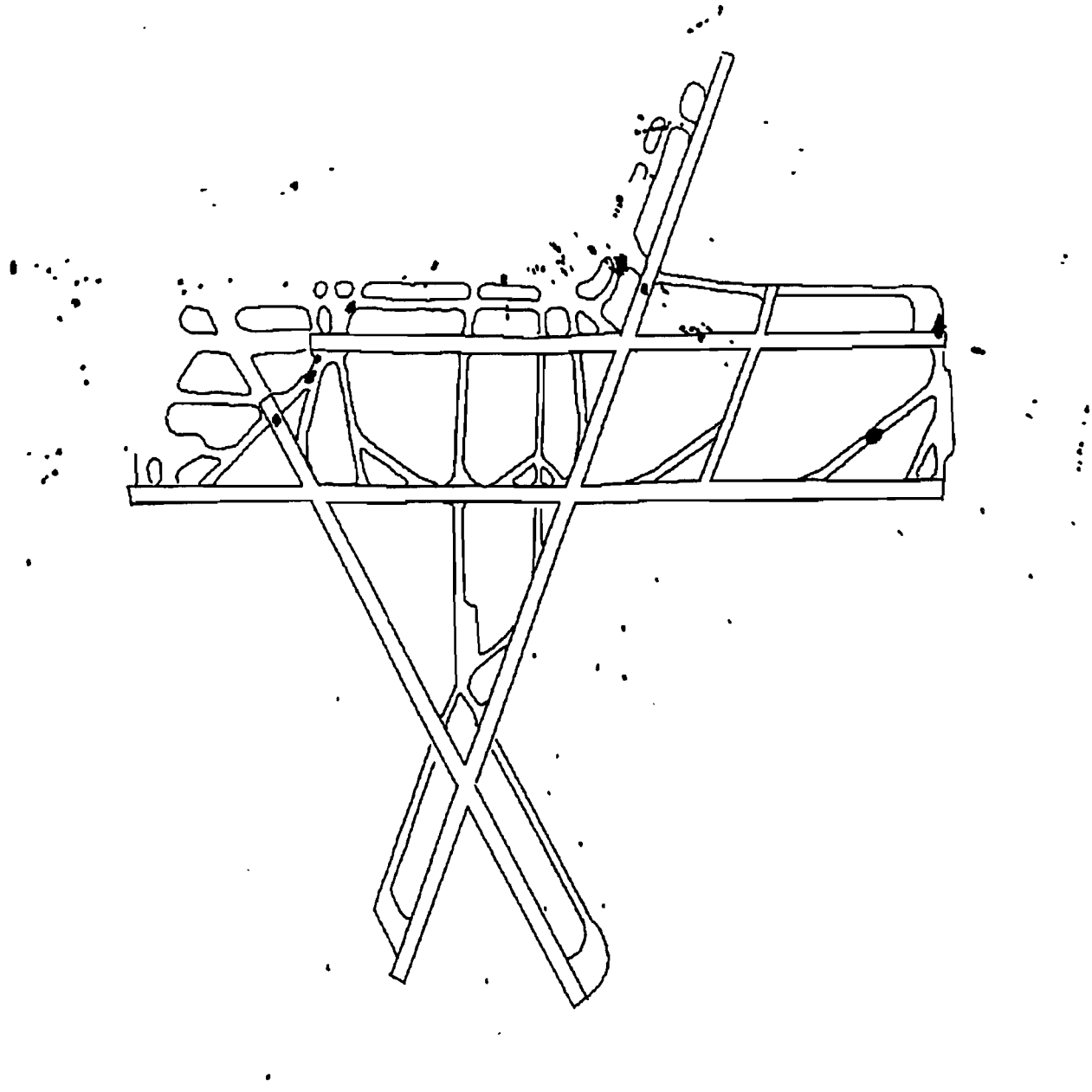


Figure 3-2. Radar data after clutter and interference rejection.

Having the data above threshold in the form of runs lowers the data transfer requirements between processors. In addition, there are efficient algorithms available to perform morphological operations on the data.

Small components are eliminated by performing an opening. An opening is a morphological operation composed of an erosion followed by a dilation. An opening on an image is illustrated in Figure 3-3 in which pixels with amplitude unity are shown in black, and pixels with amplitude zero are shown in

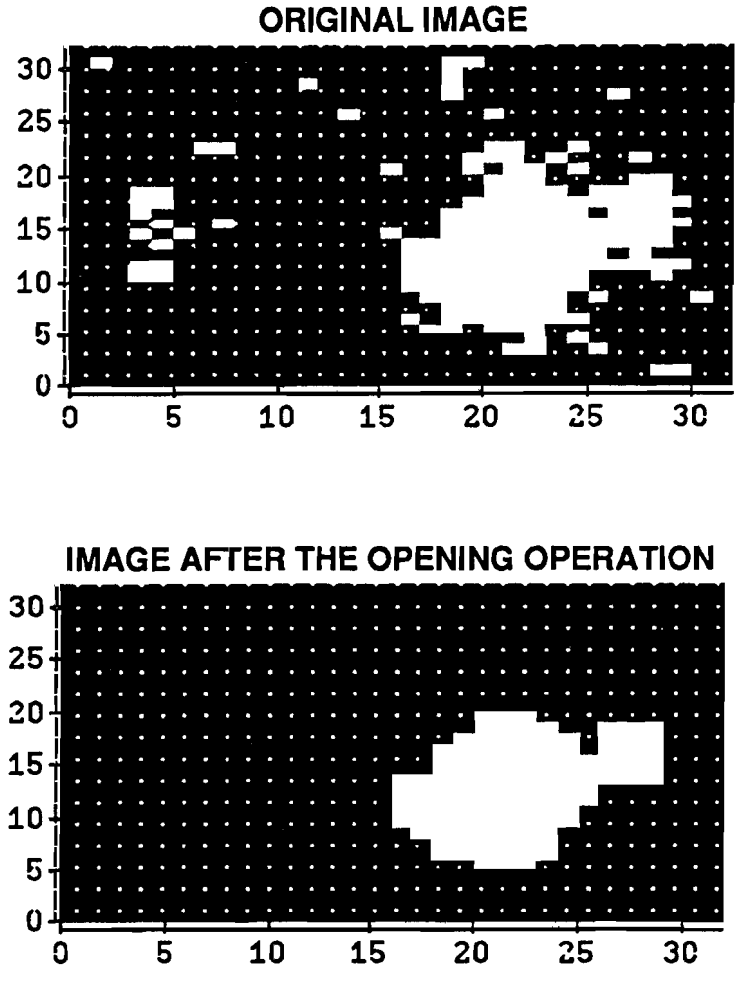


Figure 3-3. The effect of the morphological opening process on an image.

white. Each non-boundary pixel is surrounded by eight pixels. In an erosion, any black pixel that touches a white pixel even at a corner is turned white. This decreases the size of the object, smoothes the contour, and eliminates small objects. In a one-pixel dilation any white pixel that has an edge or corner in common with a black pixel is turned black. These operations are not inverses of each other. For instance, if the component was a two-by-two array of pixels, then the erosion would eliminate the object. The dilation does not bring it back into existence. Aside from eliminating small components, this operation smoothes the boundaries of components. Larger objects can be eliminated by doing an opening in which there are several erosions followed by the same number of dilations.

The number of objects to be processed in a given part of the airport can vary from scan to scan and with changes in runway use. For this reason fixed parts of the airport cannot be assigned to individual processors without incurring excessive latency in certain situations. To overcome this problem, a dynamic load balancing scheme is used. Data from the 22 surface wedges that has been processed by the clutter-rejection processors is put in a cue. When a connected-component process is free, then the time stamp is examined. The data in the oldest surface wedge is assigned to that processor.

The number of runs that needs to be processed for a scan is determined. If this number gets too high, then the processors can fall behind. To prevent this from occurring, when the number exceeds a stored value, then the opening operation is changed to one in which two erosions are performed followed by two dilations. This will eliminate components that are larger than those with a single-pixel opening, thus reducing the number of components that must be processed.

After this operation, adjacent runs are combined to form a component. In this process, a chain code, which contains the ordered position of pixels on the periphery of the component, is formed. The chain code can be used to calculate the perimeter of the component; however, perimeter is not calculated in the current RSLS implementation.

After a component is extracted from the total set of runs, the runs associated with it are subtracted from the data in the surface wedge. The process of finding a component is repeated until there is no more data present. The result of the processes in this section is a table of associated runs corresponding to individual components whose sizes are more than a certain area.

3.3.3 Components to Objects

Several individual components may be associated with the same object. In this stage of processing, close components are grouped to form an object. In addition, components that are at the surface wedge boundary are marked for additional processing in the merge processor. The algorithmic functions that are performed in this process are as follows:

1) Combine close components to form objects.
2) Determine the features of the objects.
3) Mark components that are at the surface wedge boundary.
4) Send the objects to the next processing stage.

We want to use a distance criterion to associate components with each other. It is computationally expensive to calculate the distance between components, and a different method is used to approximate the desired process.

Each component is surrounded by an enclosing wedge that is padded in distance so that there is at least a five-meter separation between the enclosing wedge and the component. Enclosing wedges that overlap are noted. The components in two enclosing wedges that overlap are dilated by a given distance radially. The same number of pixels are dilated in the azimuth direction. The number of objects in the overlapping enclosing wedges is now examined. If it is one, then the components are considered to be part of the same object, and if it is two, they are declared separate objects. If they are part of the same object, then an enclosing wedge is generated for the new object. This process is performed for each pair of overlapping wedges. Figure 3-4 shows an example in which three components are identified as parts of a single object.

Next, one determines if the objects are close enough to the surface wedge boundary so that one must see if objects in the adjacent surface wedge are part of the same object. This is done by enclosing the object in an enclosing wedge with padding. If this enclosing wedge overlaps the surface wedge boundary, then it is considered to be touching the surface wedge boundary. If none of the components

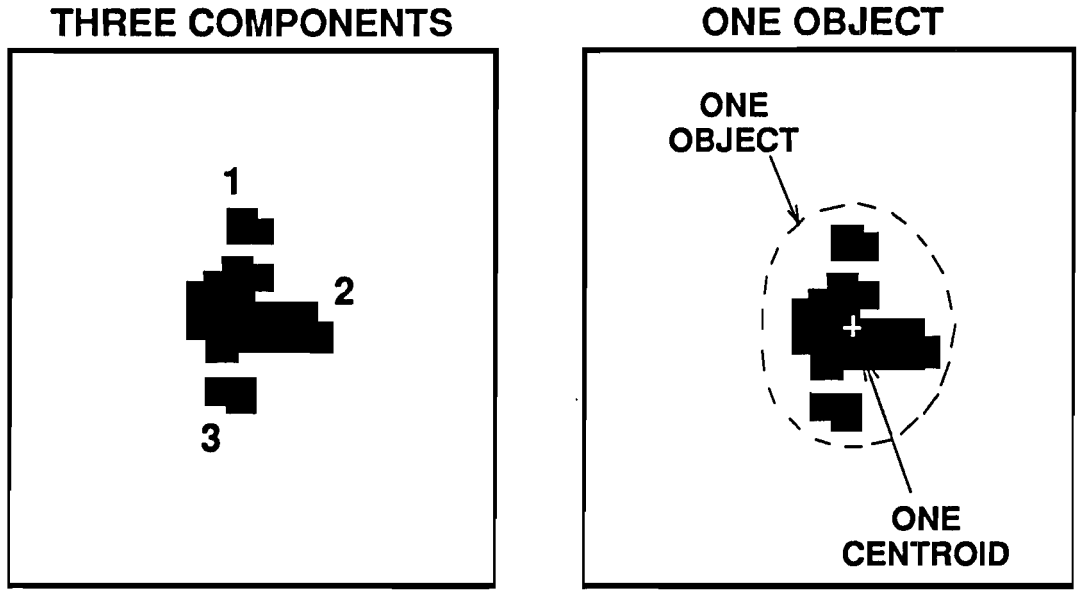


Figure 3-4. Depiction of the process that converts close components into objects.

that correspond to an object touch the surface wedge boundary, then the object is put into a table listing completed objects, and its centroid and area are calculated. The centroid is the center of gravity of the object. If a component touches a wedge boundary, it is put onto a list of incomplete objects. This processing is performed for each of the 22 surface wedges. The amount of data at this stage has been greatly reduced from that in the raw radar image.

3.3.4 Merging

The tables of complete and incomplete objects and the calculated features are passed to the merge processor. Here the incomplete objects are compared to incomplete objects in adjoining surface wedges, and they are combined into single objects when appropriate. The algorithmic functions that this process performs are as follows:

1) Read in the data from the connected components processors

2) Examine incomplete objects that are in adjoining surface wedges whose enclosing wedges overlap.

3) Declare the combined object to be a single object if dilations result in a single object and separate objects otherwise.

4) Put this combined object in the complete table if the resulting object does not touch a surface wedge boundary.

5) Repeat steps 2 and 3 if the combined object touches a surface wedge boundary.

6) Calculate features of the complete objects.

The two lists of complete and incomplete objects from the last processing stage are unflattened. The distances of incomplete objects are compared to objects in adjoining surface wedges using enclosing wedges with padding. As before, if the enclosing wedges overlap, then dilations are performed to see if they should be considered a single object. If they are not, then the same process is performed for other objects whose enclosing wedges overlap. If the object remains a complete object in all the tests, then it is put on the list of complete objects. If the components are part of a single object, then it is determined if this object is close to another surface wedge boundary. If it is not, then it is added to the list of complete objects. If it is close to a surface wedge boundary, then the above process is repeated again and again with successive surface wedges until it is declared a complete object. Figure 3-5 shows some data that straddles a surface wedge boundary and the effect of the merge process on it. The result of this process is a list of objects, that includes their centroid, area, and enclosing wedge.

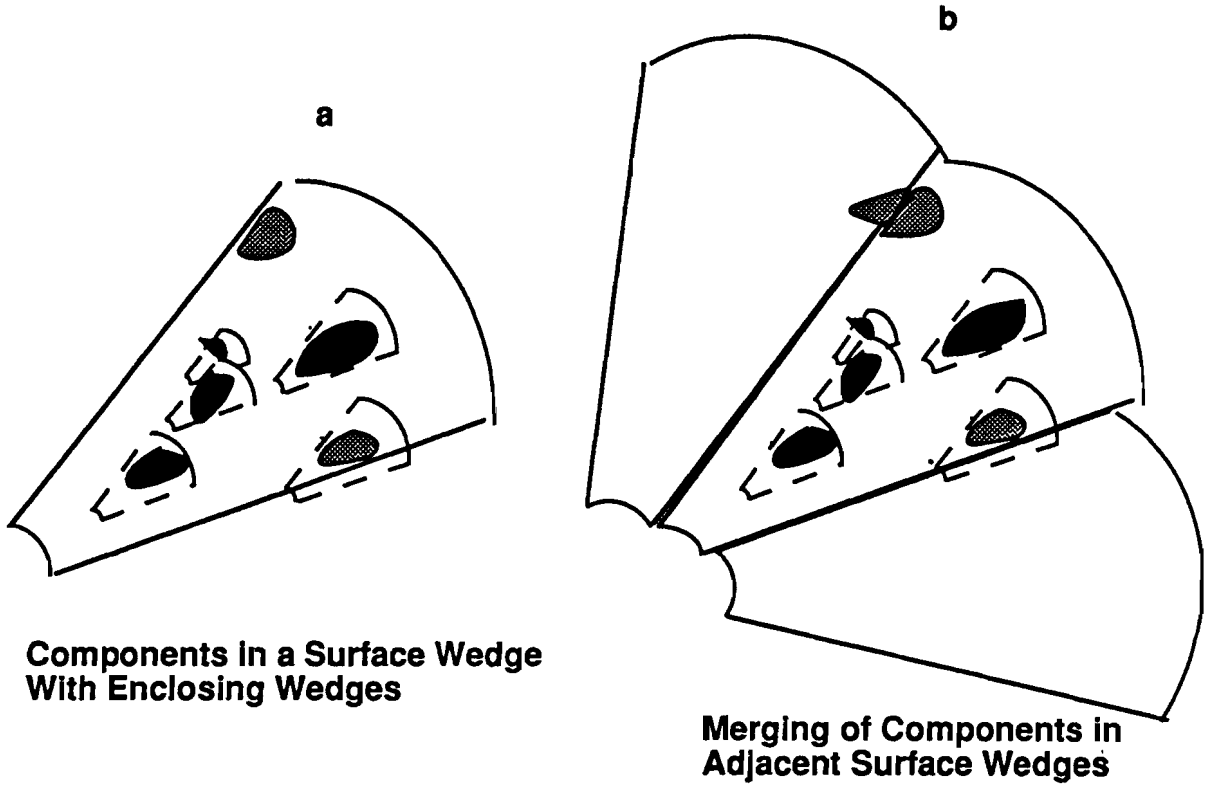


Figure 3-5. a) Components surrounded by enclosing wedges in a surface wedge. b) the merging of components in two adjoining surface wedges into one component.

3.3.5 Scan-to-Scan Correlation

In this last stage of surveillance processing, tracks are formed and updated, and this information is sent to the safety system. The logic is complicated because one needs to achieve both a low probability that tracks will be dropped on real objects, and a low probability that they will be initiated on false objects. This is accomplished by projecting tracks from a previous scan to the present scan. A best-fit correlation based on the distance of the objects near the projected track is made. If a correlation is found, then the track is updated. If there is no projected track close to a detected object, or if another object is even closer to the projected track, then a new track is started. Tracks are classified as being of low and high confidence. Because of the presence of multipath and object fading, a track can be temporarily lost. Special software has been implemented to reacquire tracks of objects rapidly that we have high confidence should still be on the movement area. Such a track loss is referred to as a "bad track drop."

Track association and lost-track recovery require additional information about the object. An aspect graph contains this information, which is unique for each type of object. Currently, it contains the area of the object as a function of both the range and the angle between a vector from the radar to the object and the main axis of the object as determined by its velocity vector. This is calculated in real time. The algorithmic functions the scan-to-scan correlation process performs are as follows:

1) Correlate the centroid position of objects from this scan to the projected positions of tracks from previous scans. Select the object to correlate with based on a priority list.

2) Initiate tracks on uncorrelated objects.

3) If track is lost on a high-confidence object, then use the rapid reacquisition algorithm to reacquire it.

4) Update the tracks based on the positions of the objects in this scan.

5) For objects in track, pass to the sensor fusion the position, estimated error in the position, area, object extent about the centroid, time, and confidence that the object is real.

6) For objects in track calculate entries for the aspect graph.

The centroid, velocity, and the area of each object are stored in the tracks. The positions of objects from previous scans are projected to this scan and compared to the positions of the centroid of objects. The correlation is first performed for high-confidence tracks, then for "bad track drops," then for low-confidence tracks, and then for new tracks. If a projection is within a user-set distance of an object in the present scan, then the same track number is assigned to the new detection. If the present positions of several objects are within this user-selected distance of the projected track, then the track number is assigned to the object that is closest to the projected track position. Conversely, if several tracks project to with the user-selected distance from an object, then the track that is associated with the object is the one that projects closest to the object. The position, velocity, and the aspect graph of the objects in track are updated.

To minimize latency in transferring tracks to the sensor fusion, the best-fit match of objects is accomplished in stages. In doing the correlation, if an object is within the user-set distance when the correlation is performed, then that information is immediately used to update the track, and it is sent to the

sensor fusion. If a subsequent object has a smaller difference in position between the projected track, then the track is corrected based on this new position, and this information is sent to the sensor fusion.

The track number can be transferred to another object if the return from the real object fades and there are other returns nearby that don't associate with the correct object, or there is a significant error in projecting the position of the object. The centroid jitter has a value of 1.3 m rms for objects moving in straight line. Because the image area of a turning object can rapidly change, the projection error is sometimes larger in that case. Because the centroid error is so small, the present system has a small probability of dropping or transferring track.

If an object in the present scan does not correlate with an object from previous scans, then a new track is initiated. The track has a confidence level associated with it that expresses one's confidence that the object is indeed a real object and not clutter or multipath. The confidence level is determined by how far it has traveled since it was first observed. Presently the confidence level is binary, and its value is zero for a new track.

Multipath can lead to false track initiation. The most common cause of multipath is a specular reflection between two objects. Typically, a small relative movement between the objects greatly reduces the amplitude of the return. We have found that requiring an apparent track to move a certain distance (presently 200 meters), called the lead-in distance, before declaring it to be a high-confidence track is a very effective filter to reduce greatly the potential of declaring a high-confidence track on multipath returns. Before it moves this distance, it is declared a low-confidence track; after it has moved this distance it is declared a high-confidence track.

This lead-in distance does not significantly affect the ability to have an initial track on an object when needed since we have a guard zone around most of the surveillance region and secondary surveillance for aircraft on approach. This zone has a width corresponding to the distance required for a track to move before it is reported as a high-confidence track. Therefore, most real objects are in high-confidence track before they enter the required surveillance region. The lead-in algorithm does, however, have a significant detrimental impact on the display of traffic on the inner taxiway, ramp, and gate areas.

Even though the use of a lead-in does not adversely affect initial track acquisition, it can adversely affect reacquisition of tracks that are lost for one or several scans. Tracks can be lost because of shadowing, object breakup, or because the track is associated with a nearby multipath object. In a majority of cases this multipath object disappears after a scan or two, in which case the track does not associate with any nearby object. To recover from this situation, a special algorithm has been implemented. If an object is in high-confidence track and is moving less than 50 m/sec, and was seen in the movement area in two out of the last three scans, and is not correlated with any object from the present scan, then a "bad track drop" is declared. When this occurs the following steps are taken to reacquire the object:

1) The user-set distance over which a correlation is sought is changed to the distance the target at its last reported velocity would move in 15 scans.

2) The object within this correlation distance that most closely matches the last cell observed of the object's aspect graph and does not associate with any high-confidence track is chosen as the new track position.

3) A correlation is sought for 20 scans, after which the track is dropped.

At the ends of several runways there are piers that support the approach lights. The combination of piers and lights has a high radar cross section and either partially or totally obscure airplanes that pass over them To reduce the probability of track loss in these areas, the tracking filter is modified so that the object coasts in the direction of the runway centerline if it is not detected. For some scans this essentially allows the track to coast through the difficult areas.

All these operations help to reduce the false-detection rate while having a very minor effect on the probability of detection of real objects. The tracks, containing size and position are passed to the sensor fusion with a binary estimate of the confidence that the object is real. Figure 3-6 shows the positions of several objects on the runway surface for a series of radar scans.

3.4 POSSIBLE SURVEILLANCE PROCESSING IMPROVEMENTS

This chapter has described the algorithms as currently implemented in the RSLS system. We have identified the following additional enhancements to improve performance:

- Many system parameters have not been fully optimized. A suite of analytical and graphical tools would greatly aid this optimization and allow us to debug more easily any software changes or additional algorithms.
- Amplitude information is currently not passed to the connected-components process. The clutter-rejection module passes only binary amplitude information. Amplitude information would help to eliminate false objects since many multipath objects have amplitudes that are significantly less than those of airplanes.
- ARTS information including aircraft type is used by sensor fusion and is presently not sent to the surveillance logic. If available to the surveillance logic, one could use the ARTS tracks and airplane identification to help prevent the initiation of false tracks and to help prevent the dropping of real tracks.
- The physical distances corresponding to a pixel are different in range and in azimuth. The distance extent of an azimuth pixel is a function of range. The performance of the components to object morphological operation in the connected-components processors would improve if it took that into account.
- The aspect graph currently only has object area. It would produce a more accurate identification if it contained perimeter and gray-scale information. Also, the current graph has values uniformly distributed in angle. A non-uniform distribution with more values in regions in which the values in the aspect graph table are rapidly changing would be better.

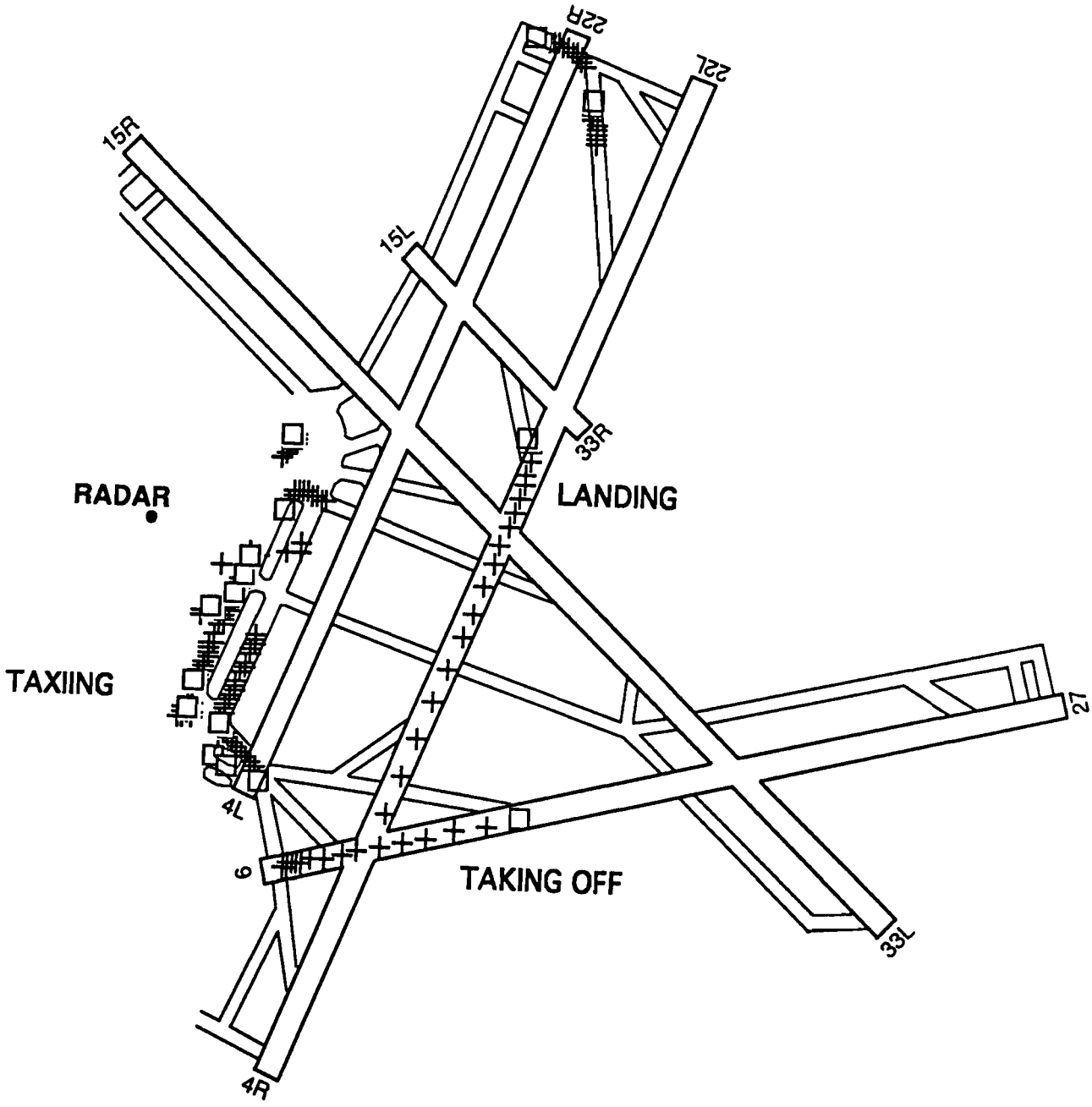


Figure 3-6. Tracks at Logan airport. The square box indicates the last position of the track.

- The aspect graph is currently generated in real time for a object on the airport surface. When the object is turning, which is the time that track is most likely to be lost, values in the aspect graph at these angles may not be available. If one knew the aircraft type and had a stored high-resolution aspect graph for that object, then one would do a better job of maintaining and reestablishing track.

- We know the position of several fixed objects and often the position of moving objects that contribute to multipath. Currently, we do not use this information. With known multipath reflectors one can predict where multipath objects will appear, which would allow them to be discarded.
- By using the details of the amplitude distribution statistics of clutter and various airplanes, which are radically different, one could do a better job of association from scan to scan.
- The clutter map is not stored from one session to the next. Doing this would allow one to initialize the system more rapidly.

4. ARTS INTERFACE

The RSLS Logan Demonstration System requires data from the Automated Radar Terminal System (ARTS-IIIA) to provide surveillance information and aircraft data tags for aircraft on approach. The purpose of this chapter is to outline the Automated Radar Terminal System (ARTS) interface requirements, to present the interface design, and to describe the interface hardware implementation. Details of the software implementation can be found in the system software chapter of this report.

4.1 REQUIREMENTS

The ARTS data required by the Logan Demonstration System include aircraft position (range, azimuth, altitude) and the information stored in data tags (including aircraft identification or ACID, and equipment type). Not all the aircraft visible to the ARTS system are important to RSLS; only data from those aircraft which are on or near the airport surface (including departures, arrivals, and other operations such as missed approaches, go-arounds, or touch-and-goes) will be required. To minimize the potential coverage gap or maximize the coverage overlap between the airport surface radar (ASR) terminal area radar data and X-band surface radar data, the ARTS data should provide information down to the lowest ranges and altitudes available, without being subjected to any unnecessary surveillance mask. All data should be provided in a timely fashion; for aircraft on final approach, any unnecessary delay in the availability of surveillance data is not acceptable.

Further requirements on the ARTS interface were mandated by the development schedule for the Logan Demonstration System. Because of the short schedule, it was not possible to make any basic changes to the operational ARTS system at Logan. No ARTS program changes were allowed, no dedicated input or output ports were available, and no interference with operations could be accepted. There was insufficient time for any significant hardware development effort. Thus the ARTS interface was required to be a passive, listen-only interface that could derive all needed data from ports already in use, and would use commercial off-the-shelf equipment.

There is no single ARTS port or data stream which carries surveillance data which are both timely and close-in, and are tagged. The surveillance data are made available to the ARTS input-output processor (IOP) via the serial communications interface processor (SCIP) port from the ASR-9. The SCIP surveillance data do not include data tags; only position, altitude, transponder code, and secondary information such as validity flags. The SCIP tap provides timely and unmasked surveillance, but does not provide the needed aircraft data tags.

The flight plan information, which is the source of the ACID and equipment type, is provided separately to the ARTS IOP via the interfacility communications (IFC) port. The ARTS computer matches the surveillance information with the flight plan information to produce, after some processing delay and a range/azimuth filter, track reports which include aircraft identification, position, and velocity. Estimates of the ARTS processing delay range from 0.7 to 2.4 seconds. The ARTS range/azimuth filter presently used at Boston is shown in Figure 4-1. The ARTS display data is available in two places: at the ARTS-MDBM (multiple display buffer memory) interface, or at the MDBM-DEDS (digital electronic display system) interface. The function of the MDBM is to allow connecting four DEDS to a single ARTS output port, while reducing the 30 Hz data refresh requirement of the DEDS to a requirement that the data be updated only when modified. The ARTS-MDBM interface thus offers the simultaneous advantages of providing access to data going to four DEDS at once and reducing the amount of data that

must be processed by the ARTS tap. It has the minor disadvantage that the information at this interface does not represent complete display messages, but only changes to display buffer memory. Thus the IOP-MDBM interface was chosen as the data source. The ARTS output data available on the IOP-MDBM port provides the needed aircraft data tags, but is neither timely nor unmasked.

The need for timely and unmasked data, including surveillance and data tags, thus requires a combination of the SCIP and MDBM data. Both the SCIP-IOP and the IOP-MDBM communications links are doubly redundant. The ARTS tap must take this redundancy into account. The information required by the RSLS Logan Demonstration System must be made available in a convenient format.

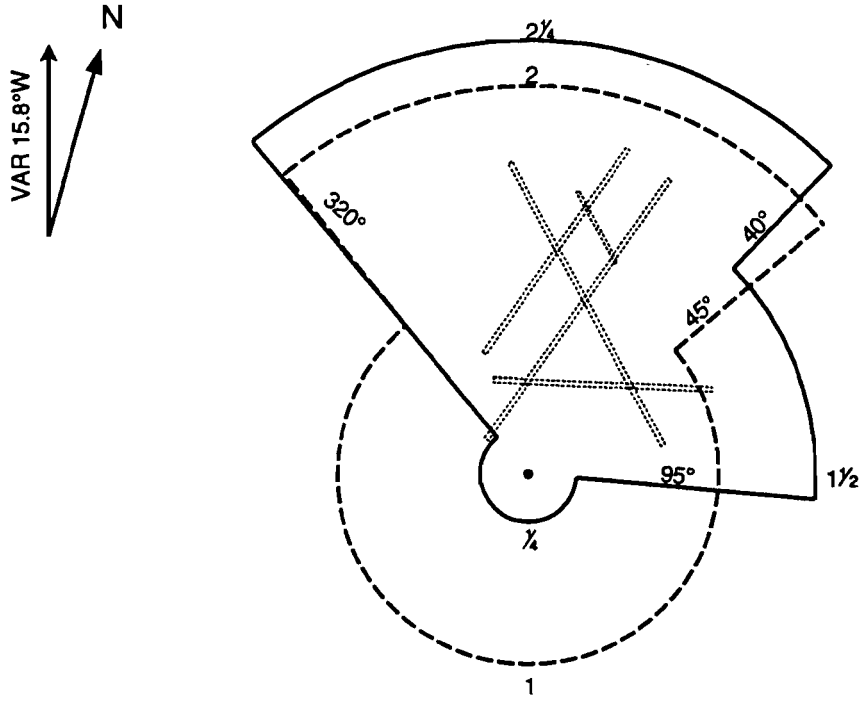


Figure 4-1. Logan ARTS departure auto-acquire (solid) and arrival auto-drop (dashed) boundaries. Magnetic azimuths and ranges in nautical miles (1" = 1 nm) are indicated. Approximate runway positions are shown (dotted).

4.2 ADIDS DESIGN

The ARTS interface which meets the above requirements is termed the ARTS Data Interface and Distribution System (ADIDS). Figure 4-2 shows the ADIDS architecture. ADIDS consists of two independent subsystems, ADIDS-SCIP and ADIDS-MDBM. These two subsystems are electronically very similar, and use different software to produce the different functionality required at the two ports. Each listens to the data available on its ARTS data stream, reformats and filters the data as necessary, and retransmits the data on an external interface. No data recording capability is built into ADIDS.

The SCIP-IOP and IOP-MDBM data paths are 30-bit wide parallel connections. To make the data easily accessible to a variety of computers, ADIDS reformats the data into an 8-bit wide external format. To control the amount of data to be reformatted and distributed, area filtering of the data is performed by

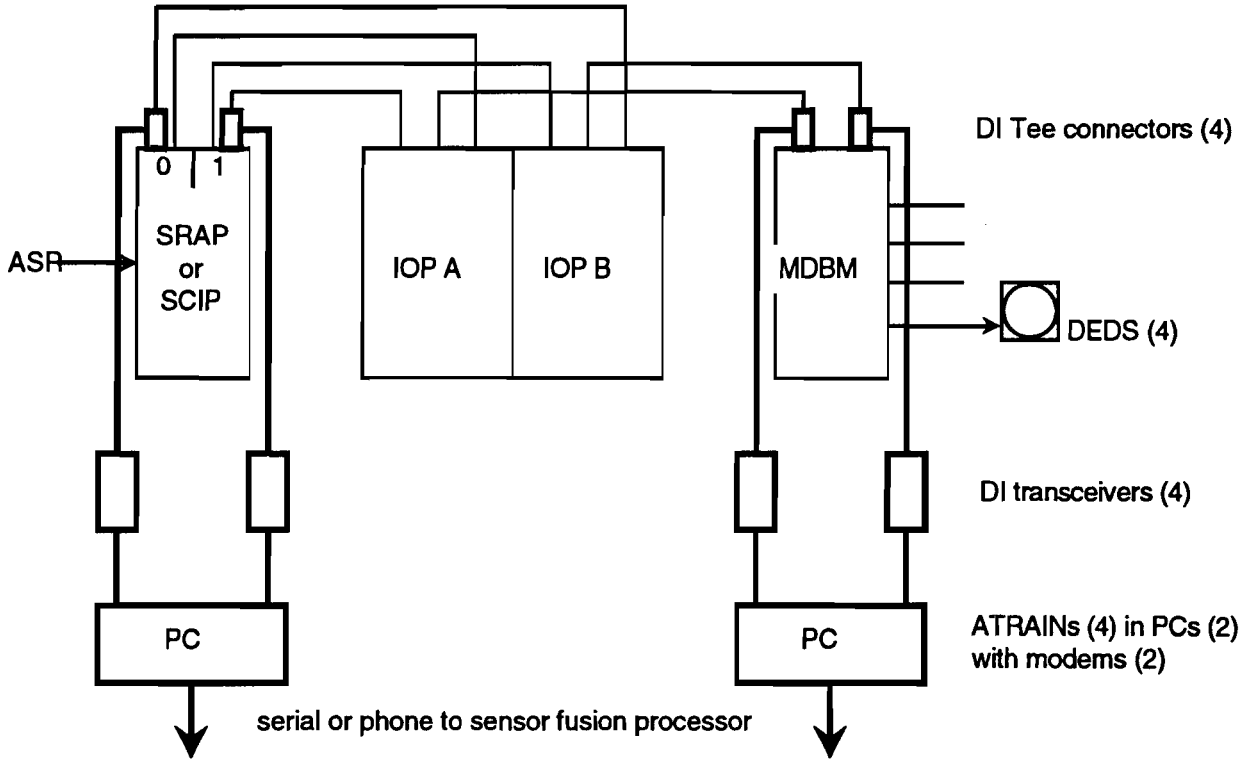


Figure 4-2. ADIDS block diagram. ARTS hardware is shown in normal line thickness, ADIDS hardware in heavy lines.

ADIDS. The design of the area filter allows for the possibility of an ASR displaced from the region of interest. Area filter parameter upload and some program control can be effected via the external interface.

4.2.1 ADIDS-SCIP

A summary of the data available at the SCIP input of the ARTS is shown in Tables 4.1a–d. Because the surveillance data at this interface are in cylindrical coordinates, the ADIDS-SCIP volume filter is also expressed in cylindrical coordinates, as shown in Figure 4-3a. The data messages from the SCIP interface are essentially self-contained, so no particular sophistication is required to perform the distribution of these data. The ADIDS-SCIP subsystem will work without modification for an ARTS-IIIA site using an ASR-7 with a sensor receiver and processor (SRAP) interface.

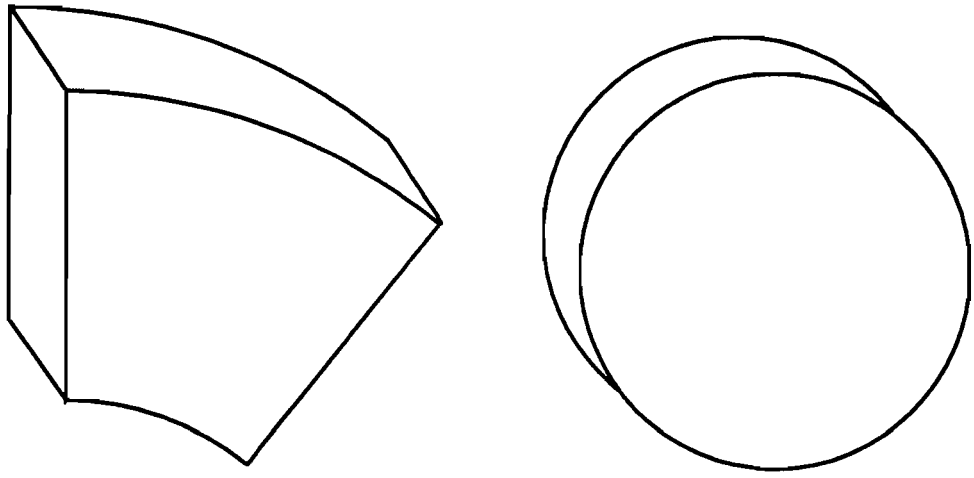


Figure 4-3a. ADIDS-SCIP volume filter geometry. For off-site ASRs, the annular wedge geometry would be more efficient. For on-site ASRs, its degenerate circular case would be appropriate.

4.2.2 ADIDS-MDBM

A summary of the data provided by the ADIDS-MDBM subsystem is shown in Tables 4.2a–d. The messages at the IOP-MDBM interface reflect the use of the MDBM as a display buffer memory. These messages are display memory update messages, rather than complete surveillance messages. Thus making the display information available in a simple format requires the emulation of the MDBM, to reconstitute the display memory update information into complete display item messages. This is done in the ADIDS-MDBM subsystem.

The MDBM emulation performed by the ADIDS-MDBM subsystem consists of two parts. The first part is to maintain the MDBM memory in accordance with the memory update messages received from the IOP. When the ADIDS-MDBM subsystem is initialized, its emulated memory map will not properly reflect the state of the MDBM internal memory map. But as memory update messages are received, more and more of the emulated memory map will contain the correct information, and after a period of time the entire emulated memory map will be correct. The normal operating procedure of the ADIDS-MDBM subsystem is to leave it turned on, constantly maintaining its emulated memory map, so initialization is not normally an issue.

The second part of the MDBM emulation is the interpretation and retransmission of the types of display data presented in Tables 4.2a-d. Because the positions of the various data types in the MDBM memory can vary between ARTS sites, and also between ARTS system software revision levels, the interpretation of the MDBM display data is specific to a particular site and software revision. There are five parameters used to describe the positions of the various data types which must be modified to account for this specificity. The surveillance data at this interface are in Cartesian coordinates, so the ADIDS-MDBM area filter is also expressed in cartesian coordinates, as shown in Figure 4-3b.

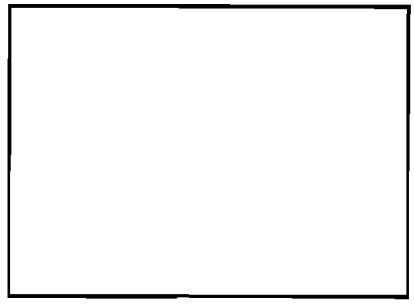


Figure 4-3b. ADIDS-MDBM area filter geometry.
The rectangle is not constrained to contain the ASR position.

4.3 HARDWARE IMPLEMENTATION

ADIDS uses ATRAIN (ARTS IIIA/TRACS Interface) hardware available from Dimensions International for its electrical connection to the ARTS system [Ref 1]. This hardware was originally developed by Dimensions International (DI) to allow the recording of SRAP or SCIP data in Transportable Radar Analysis Computer System (TRACS) format. The ATRAIN system consists of three components: a tee device, a signal repeater assembly, and a personal-computer- (PC-) resident parallel interface board. The ATRAIN system allows sufficient isolation and fault tolerance for use on an operational ARTS system. The ATRAIN interface is a passive, listen-only device. The ARTS channels monitored by the ATRAIN interface are not relayed; they continue operation in their normal manner. The ARTS data undergoes no modification, diminution, augmentation, or delay on either of the monitored

interfaces. There is no impact on the controller displays or interfaces. Test NCP approval was required and obtained for installation on the Boston ARTS-IIIA system.

There are two ATRAIN systems for each ADIDS-SCIP and ADIDS-MDBM subsystem. This allows the monitoring of either of the doubly redundant SCIP-IOP and IOP-MDBM interfaces. In normal operation, both of the SCIP-IOP interfaces are active at the same time, transmitting identical data, so either can be monitored. In case one of the SCIP-IOP interfaces becomes inactive, the ADIDS-SCIP subsystem automatically switches over to the other interface. In normal operation, only one of the IOP-MDBM interfaces is active at a time, but the chosen interface can switch back and forth aperiodically. Thus the ADIDS-MDBM subsystem monitors both IOP-MDBM interfaces at all times.

The data received by the ATRAIN boards is reformatted by the two ADIDS PCs and made available for transmission across an RS-232 interface, or via a v.32bis v.42bis modem and telephone lines to the RSLS ADIDS data client.

TABLE 4.1A

SCIP Sector Message Data

Datum	Bits	Range	MSB	LSB
parity	2	0–3	upper parity	lower parity
ID	6	0x0D		
sector	5	0–31	16 sectors	1 sector

TABLE 4.1B

SCIP Alarm Message Data

Datum	Bits	Range	MSB	LSB
parity	2	0–3	upper parity	lower parity
ID	6	0x36		
SCIP alarm	1	0-1		

TABLE 4.1C

SCIP Primary Radar Message Data

Datum	Bits	Range	MSB	LSB
parity	2	0–3	upper parity	lower parity
ID	6	0x0A		
range	14	0–255.984 nmi	128 nmi	1/64 nmi
azimuth	12	0–4095 ACPs	2048 ACPs	1 ACP
RTQC/Test	1	0–1		
ARTS quality	3	0–7		

TABLE 4.1D

SCIP Beacon Radar Message Data

Datum	Bits	Range	MSB	LSB
parity	2	0–3	upper parity	lower parity
ID	6	0x16		
range	14	0–255.984 nmi	128 nmi	1/64 nmi
azimuth	12	0–4095 ACPs	2048 ACPs	1 ACP
RTQC/Test	1	0–1		
ARTS quality	3	0–7		
Mode 3/A code	12	0–4095		
radar reinforce	1	0–1		
Mode C altitude	10+sign	-102300–102300 ft	51200 ft	100 ft
beacon hit count	5	0–31		
ring indicator	1	0–1		
3x	1	0–1		
ident indicator	1	0–1		
Mode C validity	2	0–3		
Mode 3/A validity	2	0–3		

TABLE 4.2A

ADIDS-MDBM A-Word Data

Datum	Bits	Range	MSB	LSB
parity	2	0–3	upper parity	lower parity
line type	2	solid, dashed, dotted, dash-dot		
refresh	1	0–1		
display/radar coord	1	0–1		
brightness	2	0–3		
blink	1	0–1		
force	1	0–1		
leader direction	2	N/NE, E/SE, S/SW, W/NW		
mode	2	single symbol, data block, tabular data, vector mode		
no process	1	0–1		

TABLE 4.2B

ADIDS-MDBM B-Word Data

Datum	Bits	Range	MSB	LSB
parity	2	0–3	upper parity	lower parity
Y coordinate	10+sign	-63.94–63.94 nmi	32 nmi	1/16 nmi
X coordinate	10+sign	-63.94–63.94 nmi	32 nmi	1/16 nmi
blink indicator	1	0–1		
symbol TI code	6/char	0–63		

TABLE 4.2C

ADIDS-MDBM C-Word Data

Datum	Bits	Range	MSB	LSB
parity	2	0–3	upper parity	lower parity
blink indicator	1	0–1		
symbol 1 TI code	6	0–63		
symbol 2 TI code	6	0–63		
symbol 3 TI code	6	0–63		
symbol 4 TI code	6	0–63		
symbol 5 TI code	6	0–63		

TABLE 4.2D

ADIDS-MDBM Data Block Format

Full Data Block	A B C C C C
Altitude (Limited) Data Block	A B C C
MSAW Data Block	A B C C C C C C
Single Symbol Data Block	B

REFERENCES

1. *ARTS IIIA/TRACS Interface System Instruction Book*, prepared for the Department of Transportation/Federal Aviation Administration by Dimensions International under contract DTFA03-89C-00053.

5. SENSOR FUSION

5.1 SENSOR FUSION OVERVIEW

For the RSLS Logan Demonstration, surveillance data is derived from both an X-band airport surface detection equipment (ASDE-X) radar, and the ASR-9/ATCBI (Air Traffic Control Beacon Interrogation) airport surveillance radar via a dual tap to the automated radar terminal system (ARTS IIIA) computer. The two surveillance radars produce data independently, yet feed into a common safety and display logic. The surveillance information from these two radars must be processed to form a single coherent picture of the traffic on and near the airport surface. Forming this single view of airport traffic is the responsibility of sensor fusion.

The major tasks of sensor fusion include

- Data input from two radars via three channels (ASDE, ARTS-MDBM, and ARTS-SCIP) and from a target simulator (sprites)
- Coordinate transformation to a common coordinate system
- Time translation to common clock
- Track parameter estimation (velocity, acceleration)
- Altitude pressure correction
- Beacon multipath rejection (SCIP data)
- ASDE multipath rejection
- ASDE-ARTS track fusion and unfusion
- Aircraft data tag transfer
- Unreliable track rejection
- Track coasting
- Artificial target handling
- Data output to safety logic and any other client

The design of sensor fusion had to allow some measure of generality, in particular for the tasks of track fusion and rejecting unreliable tracks. The tasks are shown schematically in Figure 5-1. Each of the functional components is discussed in detail below.

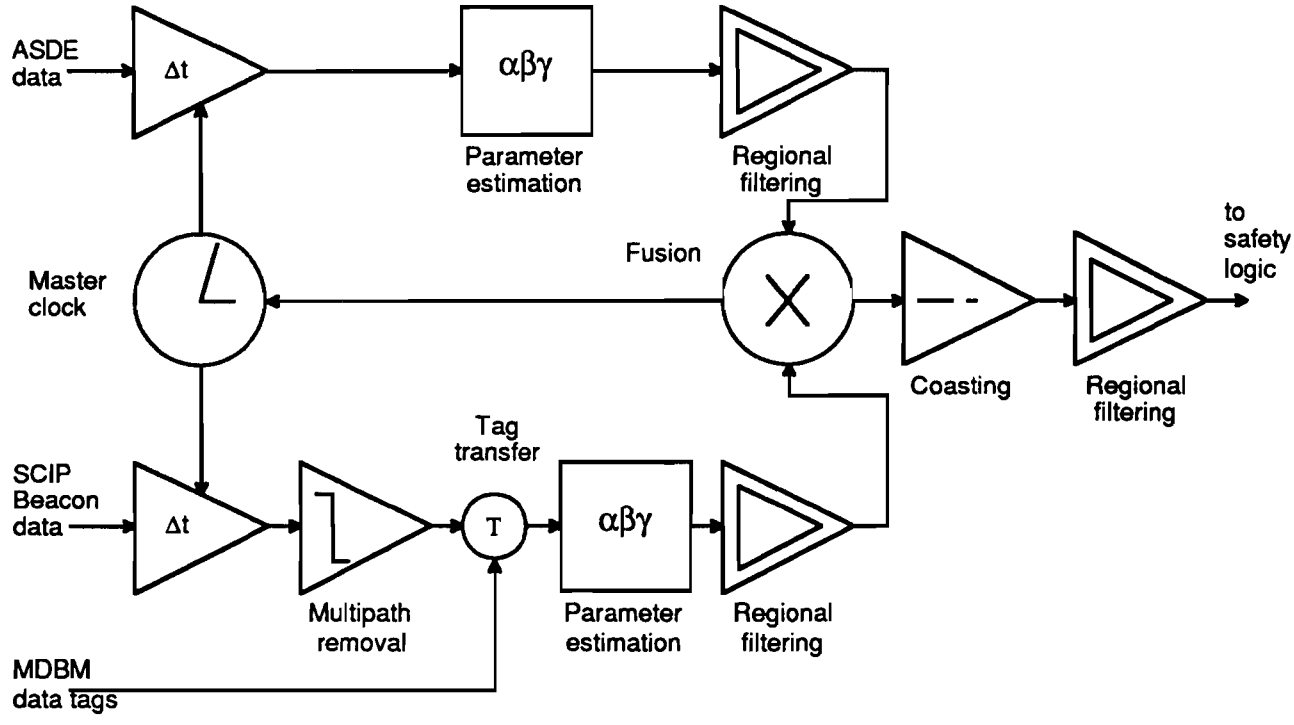


Figure 5-1. Sensor fusion block diagram.

5.2 ARTS DEMULTIPATHING

Beacon reports from the ASR-9/ATCBI airport surveillance radar derived via the ADIDS-SCIP tap are subject to multipath. The term "multipath" is used loosely to describe any beacon return which either contains false position or false squawk code information, and includes real multipath, target splits, and garbled squawk codes. Because the beacon reports contain target identification in the form of a transponder code, the algorithm for detecting and eliminating the false reports can be made to be quite effective. Because the ADIDS-MDBM tap is used only to provide aircraft identification (ACID) and not surveillance per se, and because the ARTS computer itself does some multipath rejection, the ARTS multipath removal algorithm needs to be applied only to the SCIP data.

5.2.1 ARTS Multipath Removal Algorithm Requirements

The multipath removal algorithm operates in real time. Thus it must not significantly delay targets which may be genuine. It discards data which can be immediately proven false, and it tries to replace data which prove later to be false. Finally, even when multipath leaks through for any reason, subsequent real

targets must not be suppressed; that is, the algorithm has to be able to recover from pathological conditions.

Multipath removal from the ARTS SCIP data stream is greatly aided by use of the assumption that all aircraft in the Boston TCA (Terminal Control Area) are equipped with Mode A transponders. Furthermore, all aircraft arriving to or departing from Logan airport are required to use a discrete beacon code. Compliance with these two requirements are assumed in this algorithm. The use of discrete beacon codes makes scan-to-scan association trivial. It leaves open the possibility, however, that an aircraft with no transponder, or one with an incorrectly operating or incorrectly set transponder, may become unnecessarily invisible to ASTA sensor fusion. This limitation will have to be addressed before a fully reliable ASTA system can be fielded.

The multipath removal algorithm makes use of a few basic features of a multipath return. The first is that multipath returns are necessarily further away than the true position of the aircraft. (Unfortunately this true position is in general not known.) Multipath returns often, but not always, coexist in the same radar scan with true returns. Multipath returns are often far away from the expected position of the aircraft, which is based on the previous history of that aircraft. Multipath is often due to reflections off fixed objects (like buildings), and appears in areas outside normal aircraft operating areas.

The multipath removal algorithm must accommodate several annoying features of a multipath return. The first is that a multipath return is not necessarily further way than either the previous or the next true return. Sometimes several multipath returns over several scans are received while no true returns are. Sometimes multipath returns appear close to where the true return is or should be. Sometimes multipath returns appear in areas where aircraft normally operate.

5.2.2 ARTS Multipath Suppression Algorithm Specification

There are six basic components to the ARTS multipath removal algorithm. These components are designed to work in concert, each correcting deficiencies but not interfering with the correct operation of the others. Roughly in order of data flow they are: beacon validity requirement, regional suppression, report time/range checking, track projection, track length enforcement, and pathological recovery. Each is discussed below in its own section.

5.2.2.1 Beacon validity enforcement

Only SCIP beacon reports are used by sensor fusion. SCIP primary-only reports are ignored. In addition, the beacon reports are required to have a Mode A validity code of 3 (on a 0–3 scale). Furthermore, nondiscrete or erroneous beacon codes, such as 1200 or 0, are ignored.

5.2.2.2 Regional suppression

Because certain regions may be relied upon to uniformly contain multipath target reports in great excess over true reports, these regions are mapped out of the surveillance space of the ASR radar. Any ARTS target report occurring in a suppressed region are not made available for further processing.

5.2.2.3 Report time/range checking

The basic concept behind the report time/range algorithm is simple. If more than one report is received in one radar scan for a single discrete beacon code, at least one of the reports must be multipath. As a multipath return cannot be in-range from the true aircraft position, the further of the two reports must be the multipath, and can be removed. The obvious advantages of this algorithm are that it requires little knowledge of the target's previous behavior, and that it is computationally quite efficient. The most important advantage is that it is self-starting, i.e. that it can reject multipath given only two hits on a target.

This simple picture must be modified, however, to take into account the effects of aircraft motion, surveillance error, and missing true reports. (Radar scan time variations are small compared to the time perturbations induced by these other effects.) When an aircraft is moving, its range from the sensor becomes a function of time, and the time between hits can differ from the radar scan time. Surveillance error can make a multipath return appear closer than it should be. When true reports are missing for whatever reason, the time difference between received true reports is a multiple of the radar scan time, in addition to any contribution due to aircraft motion. All of these effects could under certain circumstances induce the simple time/range algorithm to reject true reports while accepting multipath reports.

Thus the modified time/range algorithm is as follows: The time between the present report and the most recent previous report is compared to the ranges given in Table 5.1. (The previous report must have passed the whole multipath removal algorithm, otherwise it would have been removed and not stored for subsequent processing.) If this time difference lies in one of these ranges, then either the present or the previous hit is multipath. If the difference between the ranges exceeds the range difference threshold given in Table 5.1, then the nearer report is taken as the true one, and the further report is removed. If the further report is the later (current) one, then it is merely discarded. If the further report is the earlier (previous) one, then it must be removed from the track, and its effect on the track's derived parameters be eliminated. Sensor fusion presently does not cleanly reverse any fusion or unfusion decisions made from the previous report; that reversal would happen via the normal fusion and unfusion procedure. Then the nearer and later report can be appended to the track as normal. If the range difference does not exceed the threshold, then track projection (see below) is used to remove one of the reports.

TABLE 5.1

Time/Range Parameter Values

# of Full Scans	Time Range (sec)	Range Difference Threshold (nmi)
0	0.0 – 4.2	0.1
1	5.0 – 8.7	0.2
2	9.7 – 13.1	0.2
3	14.6 – 17.5	0.2
4	19.5 – 21.8	0.2
5	24.3 – 26.2	0.2
6	29.2 – 30.6	0.2
7	34.1 – 34.9	0.2
8	38.9 – 39.3	0.2

The Table 5.1 values assume a radar rotation rate of 13 revolutions/minute. Both the entries and the number of lines in Table 5.1 are subject to change when more multipath data become available. This is especially true for the last column, where there is at present really insufficient data to make adequate threshold recommendations. The entries in Table 5.1 are the result of the examination of about four hours of ARTS CDR (Continuous Data Recording) data from 7 May 1992. The last rows of Table 5.1 may never be exercised because, depending on the setting of the coast parameters, sensor fusion may have already dropped the track.

5.2.2.4 Track projection

Projection is only used if the ARTS track has a valid velocity. Track projection is used in two ways to remove ARTS multipath. First, if and only if the time/range algorithm indicates that either the present or the previous radar report is multipath, but cannot discern which is the offender, track projection removes the report which has the larger distance to the track projection. Thus if

$$d = \left\| (x,y)_{measured} - (x,y)_{projected} \right\| \tag{5.1}$$

is the distance between the reported position and the ARTS track's position projected to the reported time, then if $d_{present} < d_{previous}$ then the previous report should be removed and the present one used, otherwise the present report should be discarded. It is important that the projection operation used to determine $(x,y)_{projected}$ in each case not use either $(x,y)_{present}$ or $(x,y)_{previous}$; it must be a projection based on information independent of the two target reports being weighed.

The second way that track projection is used is to always remove reports whose distance to the track projection exceeds a threshold. That threshold is slightly range dependent, and is given by

$$d_{ARTS\_multipath} = d_0 + kr \tag{5.2}$$

Thus if $d > d_{ARTS\_multipath}$, the present radar report is discarded.

5.2.2.5 Track length enforcement

ARTS tracks are not considered reliable (blessed, in sensor fusion parlance) until N_{ARTS_bless} (presently 3) target reports are on the track. Because target reports which correspond to multipath returns generally do not get put on the track, or are removed from the track after they are put there, the ARTS track length may be smaller than the number of track reports that were provided by the SCIP data source. Only blessed tracks are sent on to safety logic for analysis.

5.2.2.6 Pathological recovery

There are conceivable situations where the above multipath removal procedures will lock on to a multipath track instead of a true one. In this case, all subsequent true target reports could be rejected. A simple recovery procedure is required to take care of such pathological situations. This pathological recovery is effected by the coast/drop mechanism already existing in sensor fusion. If an ARTS track gets no reports which are considered valid for a certain amount of time, then the track gets dropped. Subsequent reports would initiate a new track, allowing recovery.

5.3 PARAMETER ESTIMATION

5.3.1 Alpha-beta-gamma filter

Target centroids estimated from ASDE and ARTS data must be processed to reduce random noise and constant biases and to provide accurate estimates of the target positions, velocities, and accelerations. Sensor fusion uses a fixed-gain, variable time interval filter in Cartesian coordinates, the $\alpha-\beta-\gamma$ filter.

The $\alpha-\beta-\gamma$ filter can be expressed as

$$\begin{aligned}
\hat{\bar{r}} &= \bar{r}_p + \alpha\Delta\bar{r} \\
\hat{\bar{v}} &= \bar{v}_p + \beta\,\Delta\bar{r}/\Delta t \\
\hat{\bar{a}} &= \bar{a}_p + \gamma\,\Delta\bar{r}/\Delta t^2,
\end{aligned} \tag{5.3}$$

where $\hat{\bar{r}}$, $\hat{\bar{v}}$, and $\hat{\bar{a}}$ are the filtered position, velocity, and acceleration estimates, respectively, Δt is the time interval between measurements, $\Delta\bar{r} = \bar{r} - \bar{r}_p$ is the residual between the present measurement $\bar{r}$ and the predicted position $\bar{r}_p$, and $\bar{r}_p$, $\bar{v}_p$, and $\bar{a}_p$ are the predicted position, velocity, and acceleration based on the filtered estimates from the previous iteration and are given by

$$\vec{r}_p = \hat{\vec{r}} + \hat{\vec{v}}\Delta t + \tfrac{1}{2}\hat{\vec{a}}\Delta t^2$$
$$\vec{v}_p = \hat{\vec{v}} + \hat{\vec{a}}\Delta t$$
$$\vec{a}_p = \hat{\vec{a}} \qquad (5.4)$$

The temporal behavior of this filter is governed by the parameters α, β, and γ, which are fixed in advance according to an optimization scheme. The optimized filter coefficients are produced by training the filter on a data set representative of the actual data to be used, and using an evaluation function of the accuracy of simple position projection to produce optimized gains.

The implementation of the $\alpha-\beta-\gamma$ filter must take into account the possibility that, for either the ASDE or the ARTS data source, corrected updates may occur. These occur for ASDE tracks when a new target is found that is a better match to the track after a target has already been reported for that track in the same scan. Corrected updates occur for ARTS tracks when the multipath removal logic detects that a target report previously appended to a track was in fact a false report, and a new report is to be inserted in its place. These corrected reports are always one-deep; that is, they never remove more than one previous report. Thus the $\alpha-\beta-\gamma$ filter implementation retains a one-deep history of its estimates so that corrected reports can be incorporated properly.

5.3.2 Creep velocity

The RSLS safety logic uses target heading for target state determination, position prediction, and light transition and alert logic. The normal definition of heading ϕ is

$$\phi = \tan^{-1}\vec{v}, \qquad (5.5)$$

where $\vec{v}$ is the target velocity and the four-quadrant arctangent has the range $(-\pi, \pi)$.

For targets with velocities small or comparable to the velocity uncertainty, however, such an estimate of the heading is too noisy to be used. The velocity uncertainty is determined by the surveillance noise and by the coefficients of the $\alpha-\beta-\gamma$ filter used to estimate velocity. There is a tradeoff between velocity estimate noise and response time to a change in the target velocity. The coefficients used in sensor fusion were optimized for estimating future position, not for estimating present heading of slowly-moving targets.

A more useful method of estimating the velocity of slowly-moving targets, the creep velocity, is defined as follows. Let $(\vec{r}, t)$ be the latest target report position and time, and $(\vec{r}_d, t_d)$ be the most recent previous target report position which satisfies the condition

$$\|\vec{r} - \vec{r}_d\| > d_{creep}, \qquad (5.6)$$

where $\|\ \|$ denotes the Euclidean norm, and d_{creep} is a fixed creep distance parameter. Then the creep velocity $\vec{v}_{creep}$ is given by

$$\vec{v}_{creep} = (\vec{r} - \vec{r}_d)/(t - t_d), \tag{5.7}$$

and the creep heading is given in the normal way by

$$\phi_{creep} = \tan^{-1}(\vec{v}_{creep}). \tag{5.8}$$

A simple implementation would be to keep the target reports in a list or queue, appending new ones to the end and taking off old ones from the beginning to satisfy Equation 5.6. This can be problematic, however, for very nearly stationary targets, where Equation 5.6 is satisfied only for a very large number of target reports (if at all). The simple implementation could then appear as a memory leak for stationary targets. This problem can be mitigated with only slight loss of accuracy by appending a new target report $(\vec{r}', t')$ to the list or queue only if

$$\|\vec{r}' - \vec{r}\| > d_{new}, \tag{5.9}$$

where $d_{new} < d_{creep}$ is a distance parameter for novelty.

The implementation of the algorithm for creep velocity must also take into account the occurrence of corrected reports. The algorithm presently used in sensor fusion does this by keeping more target reports on the creep list than would be required by Equation 5.6. Instead, the oldest target reports $(\vec{r}'', t'')$ are only removed from the creep list when

$$\|\vec{r}'' - \vec{r}\| > d_{creep\_remove}. \tag{5.10}$$

5.3.3 Altitude correction

Altitude information is maintained by sensor fusion and consists of the altitude above ground level, reported in feet and accurate to 100s of feet; and an altitude validity bit. The transponder-reported altitude from the SCIP is modified to do a simple pressure correction to produce an altitude above ground level, and coasted to allow for dropped Mode C altitudes. This is done as follows:

A time-stamped list of the $N_{altitude-average}$ recent valid transponder-reported pressure altitudes z_{PA} is maintained. New SCIP reports have their pressure altitudes added to the list if

(a) the velocity $v < v_{on-ground}$ and
(b) the pressure altitude z_{PA} satisfies $|z_{PA} - z_{PAGL}| < \Delta z_{PAGL}$

Old pressure altitudes are removed from the list when

(a) there are more than $N_{altitude-average}$ altitudes on the list, or
(b) they are older than $t_{old-altitude}$ seconds before the present ARTS report.

The pressure altitude for ground level z_{PAGL} is computed as the average of the altitudes on the list. If there are no such altitudes, then z_{PAGL} and all altitudes reported to the safety logic are invalid, and any new pressure altitude automatically passes the Δz_{PAGL} test.

For SCIP reports with a valid transponder-reported altitude, the pressure-corrected altitude reported to the safety logic is given by $z = z_{PA} - z_{PAGL}$ if $v > v_{on-ground}$, and zero otherwise. For SCIP reports with an invalid transponder-reported altitude, the previously reported altitude is coasted without modification, up to a maximum of $t_{altitude-coast}$ seconds, with the altitude validity flag set. For reports corresponding to tracks without valid altitude data for more than $t_{altitude-coast}$ seconds, the altitude validity flag is cleared.

A fused track reports its SCIP subtrack's altitude to the safety logic. An ASDE-only track has its altitude validity flag cleared. On unfusion, the resulting ASDE-only track has its altitude validity cleared, while the ARTS-only track retains its altitude validity.

The default values for the various parameters are given in Table 5.2.

TABLE 5.2

Parameter Descriptions and Default Values for Altitude Correction

Parameter	Description	Default Value
$N_{altitude-average}$	Number of on-ground altitudes to keep	100
$t_{old-altitude}$	Age of oldest on-ground altitude to keep	600 sec
$v_{on-ground}$	Threshold velocity defining on-ground	40 kt
Δz_{PAGL}	Pressure altitude outlier limit	500 ft
$t_{altitude-coast}$	Altitude coasting time	15 sec

5.4 MDBM-SCIP SUBFUSION

The dual tap to the ARTS provides surveillance with the greater accuracy, coverage, and promptness from the ADIDS-SCIP tap, and surveillance with aircraft data tags and equipment types from the ADIDS-MDBM tap. As they both use the same source of surveillance, the ASR-9, fusing these two data streams can be done with a simple algorithm. This task is called MDBM-SCIP subfusion.

The algorithm presently used for MDBM-SCIP subfusion is a preliminary one chosen for expediency. For each MDBM target report, if there is a SCIP subtrack with no flight ID information, and if the latest target report for that SCIP subtrack is close enough to the present MDBM target report, then the MDBM target report and that SCIP subtrack are matched.

This algorithm is not the correct one to use, and does produce errors on occasion. It works most of the time, but occasionally results in two SCIP subtracks with the same flight ID. Better subfusion algorithms have been devised, but have not yet been implemented or tested.

5.5 GEOGRAPHIC FEATURE FILTER

In the design of sensor fusion, it is necessary to have a general-purpose feature-based target filter. This filter is used in at least three places, one each for the two input data streams to sensor fusion, and one for the output of sensor fusion. The purpose of the filter is to limit the target reports which can be involved in sensor fusion or which can be passed on to the safety logic.

The incorporation of a feature filter is motivated by several concerns. One is that there may be regions of degraded surveillance, where false targets due to multipath or other causes may create problems for sensor fusion (causing false association or tag swaps) and for safety logic (causing false alarms). Another concern is that in regions of overlapping ARTS and ASDE coverage, if sensor fusion fails (due to faulty surveillance), a coherent picture of the traffic can be maintained, and an extra target report from the unfused data avoided, if the extra target report is filtered out. These concerns require that the filter be definable over specific geographic regions, and be sensitive in some way to track reliability and data source. A combination of prefusion ARTS and ASDE and postfusion filters built on this general capability has been devised which achieves the following goals:

- Elimination of false ASDE tracks
- Part of the elimination of false SCIP tracks
- Definition of allowed coverage areas for the two radars
- Fallback performance in case of track fusion failure
- Elimination of slowly-moving targets in approach areas (boat filter)
- Elimination of overflight traffic
- Elimination of coasted tracks in nonsensical areas

The general requirements of the geographic feature filter are outlined below.

5.5.1 Geographic feature filter requirements

The feature filter is composed logically of three parts: a region definition (where targets have to be for the filter to do anything), a criterion definition (what nature the targets or tracks have for the filter to be applied), and a functional definition (what action the filter performs if it is applied). A filter may have zero or more regions, each with its own target criterion, and its own function. A list of the parameters associated with each feature filter is shown in Table 5.3.

5.5.2 Region definition

A filter is defined over zero or more geographic regions. Each geographic region is determined by zero or more polygons, and allows disjoint, inside-out, and topologically zeroth- and first-order regions (see Figure 5-2). A region with zero polygons is considered to contain every point. Each target report is compared to each region, and if the target report is contained in that region, then the target criterion is tested to see if the region's function is to be exercised. The target may fall within more than region, and if the target is not suppressed by the target function of the first region, then the next region is applied. If the target does not fall within any region, then there is no action of the feature filter on that target. If the filter

has no geographic regions, then it is considered to apply everywhere. It is up to the filter designer to make sure that the order of region testing is unimportant.

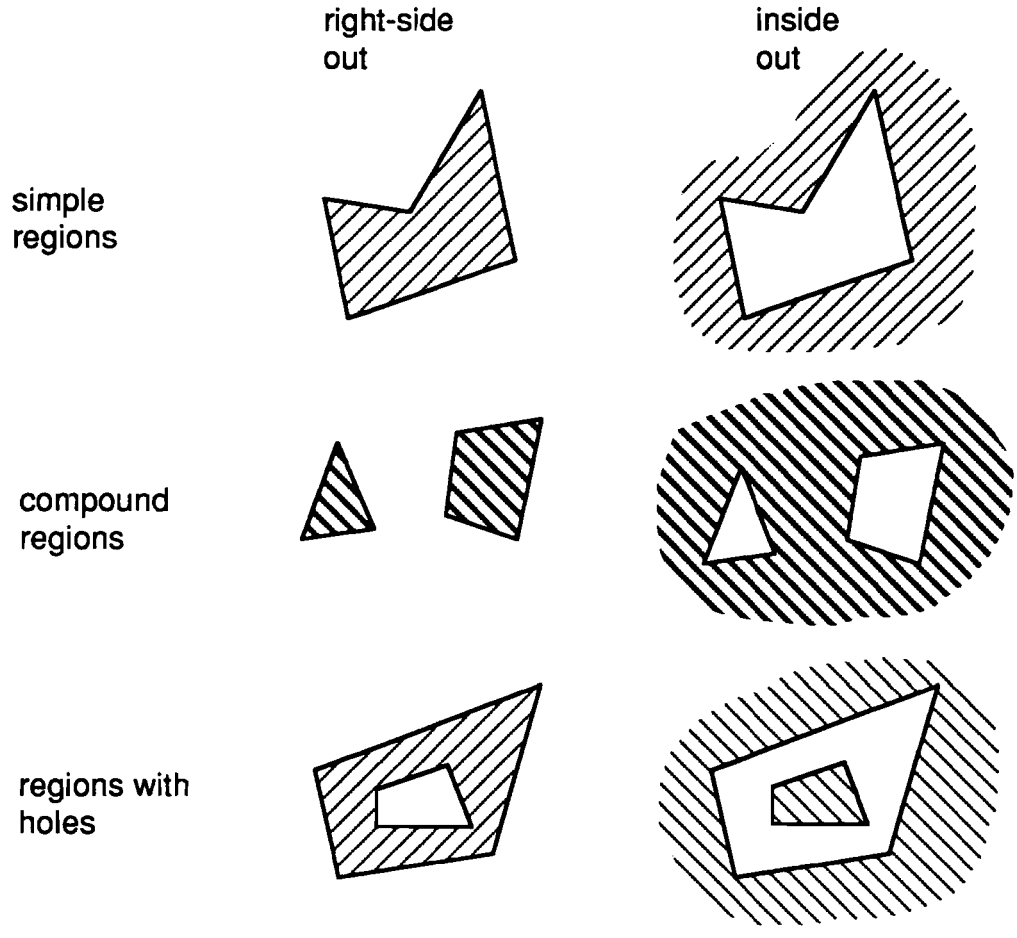


Figure 5-2. Region definitions for the geographical feature filters.

5.5.3 Criterion definition

The criterion definition allows one or more compound boolean tests to be applied to any target which is within the region. A target must pass all of the component boolean tests to satisfy the target criterion definition for a region. Hence each component boolean test must have a "don't care" parameter available to disable that test. For convenience, a boolean switch can be used to turn off the region

entirely; this is most efficiently tested before the geographic test mentioned above. The target/track features that the tests are able to examine are surveillance source, track length (in radar hits), target area, track travel distance, target altitude, and track blessedness.

5.5.4 Function definition

There are three sorts of functions which the feature filter may perform: target suppression, track blessing, and ASDE confidence following. Target suppression is just that: targets which fall a the geographic region and satisfy its target criterion are not passed on for further processing. In the case of the feature filters which happen before sensor fusion proper, this means that these target reports will not be available for sensor fusion or safety logic. In the case of the feature filter which happens after sensor fusion, this means that these target reports will not be passed on to the safety logic.

Track blessing sets a flag in the track that it is blessed. This flag may be set, but may not become cleared once set. Blessing is performed independently for ARTS and ASDE subtracks by the feature filters before track fusion, and also for the fused track by the feature filter after track fusion. Fusing two subtracks causes the new track to be initialized as unblessed. Unfusing of a track causes the two daughter tracks to become unblessed. Subtrack blessedness is unchanged by fusion or unfusion. If a track or either of its subtracks is blessed, then its target reports are passed on by sensor fusion to the safety logic.

ASDE confidence following performs a combination of target suppression and track blessing to force sensor fusion to use track confidence information passed to it by the Merge task in the ASDE processing software. This capability is used mostly for debugging purposes.

5.5.5 Geographic feature filter use

A listing of the available parameters used by a feature filter is given in Table 5.3.

TABLE 5.3

Parameter List for Geographic Feature Filters

Parameter	Values
Enabled/disabled switch	enabled or disabled
Region definition	latitude-longitude polygon list, inside-out switch
Allowed track surveillance source	any combination of ARTS only, ASDE only, fused track, or coasted
Minimum track length	hits, 0 = don't care
Maximum track length (useful mostly for blessing)	hits, infinity = don't care
Minimum target area	m^2, 0 = don't care
Minimum net travel distance	m, 0 = don't care
Minimum sum travel distance	m, 0 = don't care
Minimum altitude	ft, -5000 ft = don't care
Maximum altitude	ft, 50000 ft = don't care
Invalid altitude treatment switch	FALSE = don't apply, TRUE = apply to targets with invalid altitudes
Track blessedness	FALSE = don't care, TRUE = track must be blessed
Function selection	Suppress/bless/ASDE confidence following

A key functionality of the geographic feature filter incorporated in sensor fusion is the reduction of false tracks due to multipath. The initial specification of the geographic feature filter included a technique to reduce false tracks by requiring tracks in certain regions to have more than a minimum number of radar hits to be marked for passing on to the ASTA safety logic. (This marking is called blessing the track.) This specification was augmented as a result of an algorithm proposed by Dan Wyschogrod.

The travel distance criteria are used for lead-in filters. In such a filter, a tracked target is required to have traveled at least a minimum distance since track creation before it is blessed. This works because multipath is of two heuristic types: stationary, which would never travel far enough to satisfy this criterion; and moving, which disappears before it moves very far. This filter works very well indeed.

The travel distance for a track can be calculated in several ways. It can be calculated as the distance r_{net} between the latest position and the initial position, but this errs on the short side for curved or cornered trajectories. The travel distance can be calculated as the simple sum r_{sum} of the distances

between successive target reports, but this errs on the long side because the centroids have some noise component. The estimate r_{sum} would eventually exceed any threshold distance. These simple travel distances are inadequate by themselves.

The technique due to Dan Wyschogrod is to combine the use of the two simple calculations of the travel distance. The criterion actually used is the combination $(r_{net} > d_{net})$ && $(r_{sum} > d_{sum})$, where the parameter distances d_{net} and d_{sum} may differ from filter region to filter region. This technique makes all the errors of the individual calculations of the travel distance, but combines them so that the errors don't matter. In fact the travel distance filters used so far combine both travel distances and a track length requirement.

5.6 FUSION ALGORITHM REQUIREMENTS

The track fusion algorithm must fuse tracks which correspond to the same aircraft, but not those which do not. The fusion should use ground position, velocity, and altitude information when available to perform the fusion. Fused tracks should be checked for correctness of fusion when appropriate. In cases where the fusion would be ambiguous, any tentative fusion should be reversible.

The target fusion algorithm must be able to handle both surveillance gaps and surveillance overlaps. Gaps are produced when one sensor is physically incapable of sensing targets in a particular area (blind spots), or when the surveillance data is processed in such a way as to eliminate target reports in an area (masks). Surveillance overlaps occur when both sensors can detect a target in a particular area. At Logan, we must anticipate having both surveillance gaps and surveillance overlaps.

The target fusion algorithm must also be able to handle the temporal non-coperiodic and asynchronous nature of the two surveillance sources. The ASR-9/ATCBI spins at one revolution every 4.6 seconds, while the modified Raytheon Pathfinder now spins at one revolution every 1.74 seconds. Each surveillance source also has an incompletely specified and variable internal data delay. The two clocks from the two surveillance sources are asynchronous. This lack of synchronism in the timestamps is not constant; the two clocks drift with respect to each other.

To balance the conflicting demands of timely and accurate fusion, the target fusion algorithm must be able to perform tentative fusion and corrective unfusion. The safety logic requires immediate fusion to avoid the erroneous identification of two targets simultaneously on final approach, for example, when in fact the two targets are merely different views of the same aircraft from two surveillance sources (dual targets). This immediate fusion should be provided on the first radar hit. Such an immediate fusion can only be decided on position information, and thus is not as reliable as position-velocity fusion. Hence the initial fusion should be marked as tentative. The tentative fusion would prevent dual targets from coming to the safety logic, but would not require the display logic to put up the data tag for that target. A tentative fusion would normally be reinforced by subsequent corroborative surveillance, and would then be marked as a firm fusion. At present, tentatively and firmly fused tracks are treated identically in sensor fusion.

The hazard in fusion is that it might be performed incorrectly. Thus fusion must be reversible, and the impact of erroneous fusion must be minimized. Corrective unfusion would occur when fused tracks

prove to be independent targets. The unfusion should not produce any counter-intuitive behavior on the controller displays, nor generate any inappropriate safety system lights or alarms (though in such a case, an alarm quite likely would be appropriate). An error message should be generated if a firmly-fused track becomes unfused.

The central target fusion algorithm is composed of a simple track maintenance rules to reduce the number of tracks considered for fusion; a position- and velocity-dependent fusion criterion to identify fused target pairs; and a time-synchronization scheme to maintain an accurate estimate of the clock offset between the two surveillance sources. Each of these three components is discussed in its own section.

5.6.1 Fusion Track Maintenance

Several types of target tracks are maintained to account for the fusion process. Tracks that are not fused but are suitable for fusion are of two types: ARTS tracks (ATBC) and X-band tracks (XTBC). Tracks that have been fused recently are called tentatively fused tracks (TC). Tracks that have been fused a while are called firmly fused tracks (FC). Other tracks are either ARTS unfused tracks (AUC) and X-band unfused tracks (XUC). Tracks that are to be dropped are called NIL.

Several geometric concepts must be introduced to specify the algorithm. The surveillance maps specify the areas where targets are allowed to be fused (see Figure 5-3). ASDE surveillance data is available for targets in region X, whereas ARTS information is available for targets in region A. For simplicity of algorithm development, generate circles centered at the surveillance origin with the following properties: Circle L contains the whole ASDE surveillance area, plus a little pad for surveillance error. Circle F contains at least all the targets which could possibly enter Circle L within N_{Acoast} ASR rotations, assuming larger-than-normal airborne aircraft velocities. The corresponding radii are denoted by R_L and R_F. (Typically $R_F - R_L$ is at least a statute mile.) Let $Z_{max\pm}$ be the maximum altitude that an ARTS target may have for fusion to take place, with the $\pm$ denoting two parameters providing hysteresis. Let the range of the target from the surveillance origin be denoted by R, and the target's ARTS-reported altitude by Z.

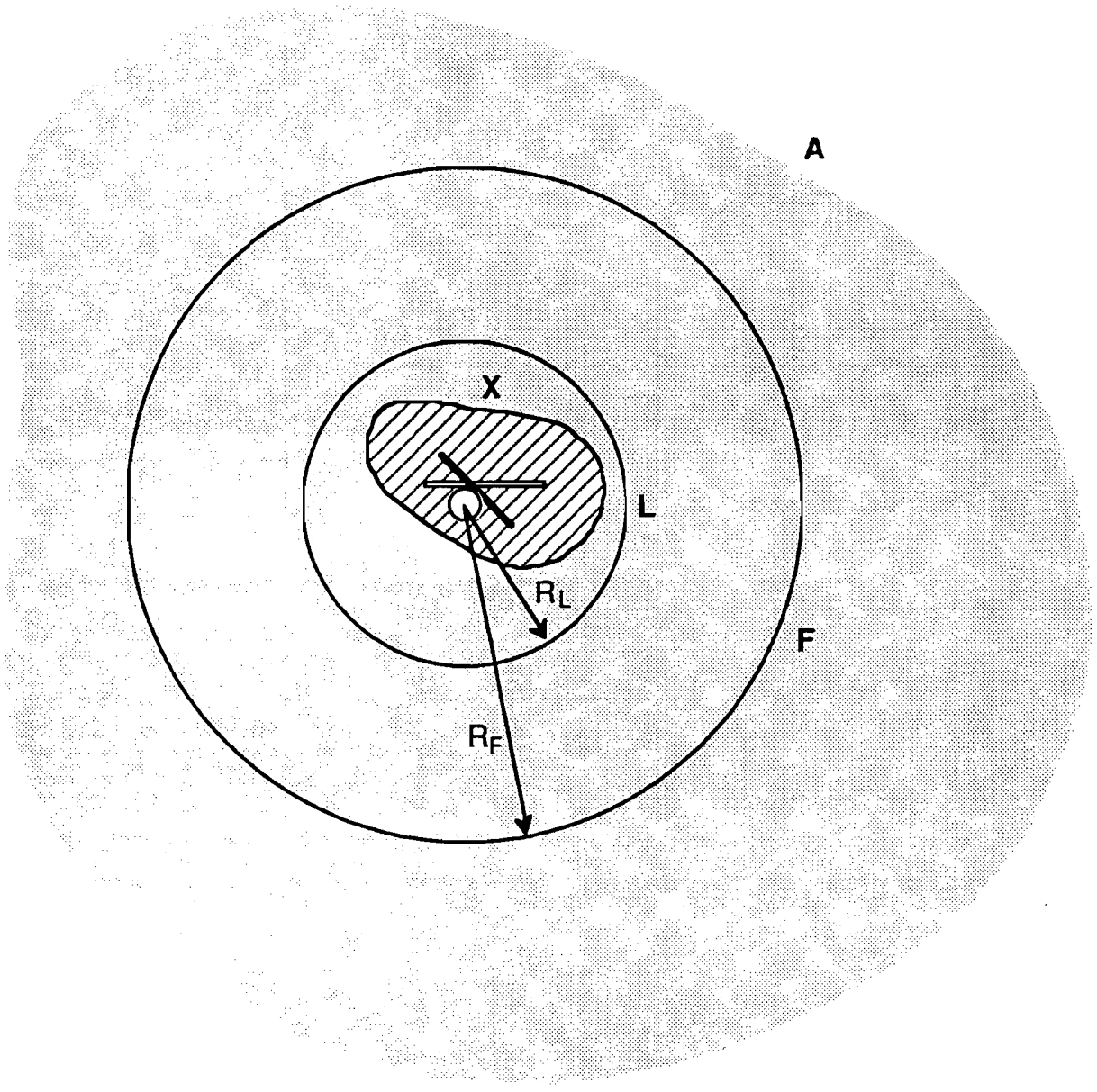


Figure 5-3. Fusion geometrical concepts. ASDE coverage area (hatched), ARTS coverage area (dotted), and defined circles and radii.

Now the tracks maintenance is performed according to the rules in Table 5.4. The altitude-dependent tests are made only if valid altitude is available. The fusion criterion defined in the following section.

TABLE 5.4

Fusion Track Maintenance Rules

Transition	Condition
ARTS tracks	
AUC → ATBC	$\{[(R < R_F) \&\& (R < R_{previous})] \| (R < R_L)\} \&\& (Z < Z_{max-})$
AUC → NIL	track is dropped
ATBC → AUC	$(R > R_F) \| (Z > Z_{max+})$
ATBC → NIL	track is dropped
ASDE tracks	
XUC → XTBC	always
XTBC → TC	track satisfies fusion criterion with one ARTS track
XTBC → NIL	track is dropped
unfused track pairs	
ATBC+XTBC → TC	(fusion) track pair uniquely satisfies fusion criterion
fused tracks	
TC → FC	track satisfies fusion criterion > N_{Fc} times
TC or FC → ATBC+XTBC	(unfusion) track pair satisfies unfusion criterion
TC or FC → ATBC	ASDE subtrack is dropped
TC or FC → XTBC	ARTS subtrack is dropped

5.6.2 Fusion Criterion

Fusion is established by comparing any updated track in the ATBC list with all tracks in the XTBC list, and also vice versa (any updated track in the XTBC list with all tracks in the ATBC list). Fusion is checked for all updated track pairs in the TC and FC lists. Fusion is performed on time-synchronized position information. (Time synchronization is discussed in the following section.) Let $(\vec{r}_A, \vec{v}_A)$ be the two-dimensional position and velocity of an ARTS target, and $(\vec{r}_X, \vec{v}_X)$ be those of an ASDE target, the older one projected to the more recent one's time. Then let

$$\Delta r^2 = \|\vec{r}_A - \vec{r}_X\|^2 + \tau_c^2 \|\vec{v}_A - \vec{v}_X\|^2 \tag{5.11}$$

be the square of the distance between the two targets, where τ_c^2 is a constant. If either surveillance source does not provide velocity, then the velocity difference defaults to a constant value $\Delta v_{default}$. The fusion criterion is then that $\Delta r^2 < r_{fuse}^2$, where r_{fuse}^2 depends on whether either subtrack is undergoing high

acceleration or deceleration. In certain cases, an ASDE-only track may have previously been fused with an ARTS track, and the ARTS track may drop, causing the track to enter the XTBC list. In this case, this track would, barring any possible ASDE track swap, retain the ARTS data tags matched to it by the previous fusion.

Unfusion is possible for fused tracks where surveillance requires it. Unfusion occurs when $\Delta r^2 > \mu r^2_{fuse}$, where the hysteresis multiplier $\mu > 1$. Unfusion further requires that surveillance from both sources be sufficiently recent.

The word "uniquely" in the rule for the transition ATBC+XTBC -> TC has the following explanation. The latest track report (whether it be ASDE or ARTS) must fuse with one and only one track from the other surveillance source. (One should note that fusion is not precisely commutative. That is an unfortunate necessity, due to the extrapolation of the position of only one target in the fused pair.) If the uniqueness criterion is not satisfied, then no fusion is performed.

One problem with the direct implementation of the above algorithm is that each new target report for a track on either TBC list requires a fusion check with all targets on the other TBC list. A certain amount of optimization is obtained by allowing fusion only between old ATBC tracks (those with more than N_{Aold} target reports) and new XTBC tracks (those with less than N_{Xold} target reports), and vice versa. This would effectively allow fusion only between ARTS arrivals and new ASDE tracks, and between ARTS departures and old ASDE tracks. This optimization must allow exceptions in certain cases discussed in the section for fusion history lists.

5.6.3 Fusion history lists

Normally ARTS and ASDE subtracks may only seek fusion (be on a fuse candidate list) if they are young, presently defined as having less than N_{Aold} or N_{Xold} hits, respectively. Old subtracks may not fuse with old subtracks. This produces the correct fusion behavior for both arrivals and departures, and for discontinuous subtracks. It disallows the fusion of subtracks which should have fused previously.

This procedure ignores the possibility that an unfusion event occurs (due to surveillance error), but the subtracks stay continuous. As an X-band surveillance error sufficient to cause unfusion would likely also cause a subtrack drop as well, the more likely cause for this situation would be a surveillance error in the SCIP subtrack. The result is that the two subtracks are possibly both old, and would then not allowed to fuse together again. The side effect is that the persistent unfused SCIP subtracks can fuse incorrectly with false tracks. As we do not want to eliminate the prohibition of old-on-old fusion, a solution in the form of fusion history lists is implemented.

A fusion history list is maintained individually for each subtrack, both X-band and SCIP, and contains a list of other subtracks that this subtrack has ever been fused with. When a subtrack is initialized, its fusion history list is initialized to be empty. When a fusion event occurs, a pointer to each subtrack's fusion counterpart is appended to the fusion history list. Thus there is a pointer from each fused subtrack to the other. Unfusion has no effect on the fusion history lists (that would be historical revisionism). When a subtrack is deleted, references to it are deleted from all other fusion history lists (no

necrological list entries are allowed). This can be done efficiently because the fusion history list in the subtrack being dropped points to all the other subtracks that refer to it in their fusion history lists.

In addition to the normal procedure of young subtracks checking for fusion, old subtracks with nonempty fusion history lists are also checked for fusion when they update, but only for fusion with unfused subtracks in their fusion history lists. This is the only effect of the fusion history lists on fusion itself.

The fusion history lists associated with each subtrack allow unfusion to be easily reversed if later warranted, even in the case that both subtracks are old.

5.6.4 Fusion track number assignment

To mesh the treatment of fusion track numbers with the requirements of safety logic, sensor fusion must assign fusion track numbers carefully. Fusion track numbers are assigned by fusion to be unique identifiers of tracks which may or may not be fused, and which may or may not be sent on to safety logic.

To provide safety logic with the most continuous picture of track behavior, it is important that track numbers appear as consistent as possible on fusion and unfusion. To this end, sensor fusion uses the ASDE subtrack blessed flag to determine which fusion number to retain on fusion and which subtrack to give the old fusion track number on unfusion.

Let S be the ARTS (SCIP) subtrack number, X be the ASDE (X-band) subtrack number, F_S be the pre-fusion or post-unfusion ARTS-only fusion track number, F_X be the that of the ASDE-only fusion track, and F be the post-fusion or pre-unfusion fused track number. The fusion and unfusion processes are depicted in Figure 5-4.

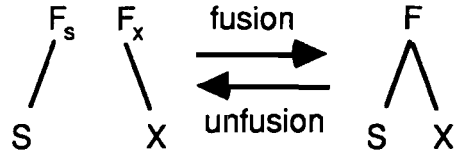


Figure 5-4. Fusion and unfusion processes.

On fusion, if the ASDE subtrack is blessed, then $F = F_x$, otherwise $F = F_s$. On unfusion, if the ASDE subtrack is blessed, then $F_s = F$ and F_x is a new fusion track number, otherwise $F_x = F$ and F_s is a new fusion track number.

5.6.5 Fusion Drop and Suppress Messages

Two sorts of end-of-track messages sent to safety logic, drop messages and suppression messages. Drop messages are to be sent only for those tracks which have been sent to safety logic and are now being purged from the sensor fusion track store. Suppression messages are to be sent for those tracks which are no longer to be sent to safety logic and which are not to have any effect on safety logic, but remain in the sensor fusion track store and may be sent to safety logic again in the future.

The reason two such message types are used is that safety logic purges its track store when it receives the drop message, and any state history for that track would have been deleted. This affords suboptimal performance, as track history is used for several of the safety logic algorithms.

Track fusion always results in a drop message. Unfusion may result in a suppression message, if neither unfused track is to be transmitted to safety logic.

5.6.6 Time Synchronization

Sensor fusion must synchronize the clocks of the two surveillance sources, and use extrapolation to bring the asynchronous target reports into effective synchrony. Sensor fusion uses simple velocity extrapolation to do the latter task. Thus to extrapolate an X-band track to check for fusion with an ARTS target report, one would merely calculate

$$\bar{r}_X(t_A) = \bar{r}_X(t_X) + \bar{v}_{sX}(t_A - t_X - t_{AX}) \tag{5.12}$$

where t_A is the surveillance time from the ARTS target report, t_X is that of the latest X-band target report, and t_{AX} is the time offset between the two sources. No extrapolation for velocity would be required (constant velocity assumption). This is sufficient for the regions where fusion is important, i.e. for targets on short final and targets right after departure. A similar expression for Equation 5.12 can be written for extrapolating ARTS tracks, with $t_{XA} = -t_{AX}$.

To synchronize the clocks of the two surveillance sources, t_{AX} is initialized to the smallest difference between the arrival times of the first hundred ASDE reports and the system time of the computer running sensor fusion, plus a constant fudge factor for average ASDE processing delays. ADIDS-SCIP derives its clock indirectly from the same system time and thus has no effect on the initialization of t_{AX}. The initialization can be done to within an accuracy of less than a second. After initialization, the value of t_{AX} is adjusted when proper fusions are being made, to allow for drift. This adjustment is performed by a simple exponential filter

$$t_{AX}(new) = (1-a)t_{AX}(old) + at_{AXopt} \tag{5.13}$$

where t_{AXopt} is the optimal offset determined by the minimization of the fusion error with respect to t_{AX}.

$$t_{AXopt} = t_A - t_X - \bar{v}_X \cdot (\bar{r}_A - \bar{r}_X)/\|\bar{v}_X\|^2 \tag{5.14}$$

A similar equation can be written for synchronization update based on the velocity estimated from ARTS data. The most recently estimated velocity is used. The steady state gain a in the exponential filter is a small number, typically 0.01. In view of the appearance of the velocity magnitude squared in the denominator of Equation 5.14, the adjustment to t_{AX} are not performed when the velocity drops below a certain velocity, about 5 knots, used in the target state transition determination.

5.7 TRACK COASTING

Sensor fusion allows two different types of coasting. The first type produces coasted tracks, which are allowed to be treated differently for the purposes of track fusion. The second type produces extrapolated target updates, with the tracks fusion characteristics otherwise unmodified. At present, the capability to produce coasted tracks is not used in the Logan demo, and will not be discussed here.

Track extrapolation allows target reports to be generated for tracks which have not been reported to fusion for a while. Three time parameters define the characteristics of extrapolation: the time before the first extrapolation is made, the time between extrapolations, and the time after which no more extrapolations are made. After this last time, a track is put into the coast state, which presently causes it to be dropped immediately.

Track extrapolation thus allows tracks to be coasted through surveillance gaps or regions of degraded surveillance. By setting the time to first extrapolation less than a normal radar scan time, scan interpolation can be performed. This allows the simulation of a radar with faster scan rates.

5.8 ARTIFICIAL TARGET HANDLING

Sensor fusion allows the injection of artificial targets or sprites. These are useful for demonstrating and testing the RSLS safety logic and displays. Artificial tracks are treated exactly as normal tracks, with the exception that fusion is allowed only between real tracks and real tracks, or between artificial tracks and artificial tracks, and never between real tracks and artificial tracks.

5.9 POTENTIAL SENSOR FUSION IMPROVEMENTS

Sensor fusion presently only allows tracks from two radar sources to be fused. A real fusion system should be able to accommodate multiple surveillance sources. There is an immediate need to incorporate ASR-9 primary radar reports into sensor fusion as well as the beacon reports. The incorporation of information from further advanced surface surveillance systems, such as the ADS-Mode S, will require a multisensor fusion capability.

Sensor fusion presently does not attempt to combine position reports from different surveillance sources to produce more accurate positions. In the present case, with the vast disparity in centroid accuracy between the ASDE-X and the ASR-9/BI surveillance, little accuracy is to be gained. Future systems may incorporate radars with more nearly equal accuracies, whereupon such a capability would be advantageous.

The timestamps provided by ADIDS-SCIP reflect the time of receipt of the surveillance, and not the time of the surveillance itself. For the most part, the difference between the two can be ignored, but an occasional beacon report is delayed anomalously before it gets to ADIDS-SCIP, resulting in an incorrect timestamp. Sensor fusion should be able to correct such anomalous timestamps. This can be done by synchronizing the normally correct timestamps to a calculated radar rotation model. Timestamps that don't fit the model would be corrected by using the model prediction instead of the reported time of receipt.

The MDBM-SCIP subfusion algorithm needs to be upgraded to handle ambiguous subfusion. This would prevent the occasional incorrectly tagged target that presently occurs.

When a track first becomes blessed, sensor fusion presently merely sends the latest target report for that track to safety logic. A better approach would be to send a history of the pre-blessed behavior of that track, so that any track state information can be properly initialized by safety logic.

Longer-term improvements to the entire RSLS system may involve architectural changes. One such possibility is the integration of sensor fusion with ASDE scan-to-scan association. This would allow better ASDE track initiation for arrival aircraft, where the expected position, velocity, and other aircraft features can be derived from ARTS data. It may also help disambiguate tracks that may have been confused due to merged targets.

Another longer-term improvement would be to use the arriving runway assignment produced by the surface monitor to aid in coasting arriving aircraft through large surveillance gaps. This would increase the reliability of fusion for arrivals, and allow the more frequent retention of tag data for arriving aircraft.

6. SAFETY LOGIC

6.1 OVERVIEW OF THE SAFETY-LOGIC ARCHITECTURE

The safety-logic algorithms determine when the runway-status lights should be on (red) or off. They also determine when alerts should be issued, although alerts have been added to the RSLS for demonstration purposes only to show how a complete safety system would function. Figure 6-1 illustrates the safety-logic architecture. The sensor-fusion-and-tracking algorithms, described in an earlier chapter, provide track reports to the safety logic. The track reports contain the following information for each tracked target: 1) track number (both ASDE track number and sensor-fusion track number), 2) position, 3) velocity, 4) acceleration, 5) extent (a measure of the size of the target), and 6) other data, such as altitude (when it is available from ARTS data).

The safety-logic algorithms can be divided into four modules, as shown in Figure 6-1: 1) target state machine, 2) prediction engine, 3) light-control logic, and 4) alert logic. The target state machine determines the target's "state," for example, whether it is departing, landing, or taxiing. The prediction engine estimates where the target could be in the future by predicting its path. The details of the prediction engine are in Section 7.7. The light-control logic determines which runway-status lights should be red and which should be off, and indicates this to the radar displays and airport model board by issuing light commands. The alert logic sends alert messages to the radar displays and the audible alerting system when it has determined that two targets are in conflict and that conflict warrants an alert. The rest of this chapter will give a high-level description of these modules. A detailed outline of the safety-logic algorithms can be found in Appendix A.

Other inputs to the safety logic are parameters from two databases, as shown in Figure 6-1. The safety-logic database contains the parameters used directly in the safety-logic modules. Although the safety logic is airport-independent, it still requires the airport-surface database, which provides specific information about the airport such as the location of runways and taxiways. Appendix B lists the default values for the safety-logic parameters referenced in this chapter and in Appendix A. Appendix C lists the runway settings that depend on the particular runway configuration in use at the airport. These settings are used to override the default values of some of the safety-logic parameters.

6.2 TARGET STATE MACHINE

6.2.1 Target States

Every target is classified as being in one and only one target state. Table 6.1 is a list of the target states and the acronyms that appear in the safety logic. The "landing abort" state refers to a go-around, a missed approach, a touch-and-go, or not landing safely (e.g., going off the end of the runway).

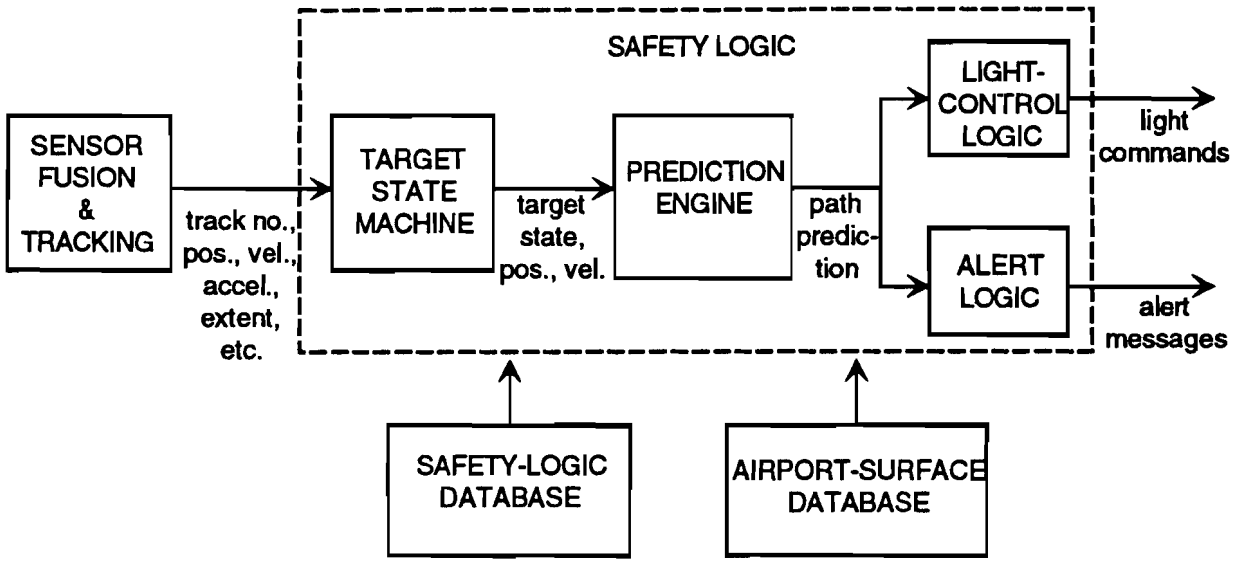


Figure 6-1. Safety-logic architecture.

TABLE 6.1
TARGET STATES

Target-State Acronym	Target State
STP	stopped
TAX	taxi
ARR	arrival
LDG	landing
DEP	departure
DBT	departure abort
LBT	landing abort
UNK	unknown intent

6.2.2 Target-State Transitions

The target state machine specifies the transitions *from* one target state *to* another, as opposed to specifying the conditions for being *in* a target state. Figure 6-2 illustrates the possible transitions between states. The state "NONE" is a pseudostate that represents "no track," i.e., it is used as a source and sink of tracks. Although some of the states are not appropriate for ground vehicles (e.g., DEP, ARR, and LDG), the RSLS surveillance is not adequate to determine whether a target is an aircraft. Thus, a ground vehicle might be put into an inappropriate state, such as in a transition from TAX to DEP.

Appendix A contains a detailed description of the state transitions and provides the transition rules. For example, Figure 6-3 illustrates the transition from ARR to LDG. Namely, the target enters the airport region, wherein the airport region is defined as a convex polygon around the airport that includes the entire movement area but does not extend too far beyond the movement area.

There are two advantages to the state-machine approach. First, it avoids the possibility that a target might "fall between the cracks," that is, end up in a situation for which no target state has been defined. With a state machine, a target will always have a defined target state. Second, the target state machine incorporates hysteresis, a technique for avoiding the problem of jumping back and forth between states due to surveillance errors. Figure 6-4 illustrates a target-state transition that is an example of hysteresis. Suppose that the parameters v_tax- and v_stp+ are velocities[1] and that v_tax- < v_stp+. The transition from STP to TAX occurs when the velocity of a target in the stopped state exceeds v_stp+. However, the transition from TAX to STP occurs when the velocity drops below v_tax-. By having two different velocity parameters separated by an amount greater than the expected surveillance error, small surveillance errors would not cause numerous back-and-forth state transitions.

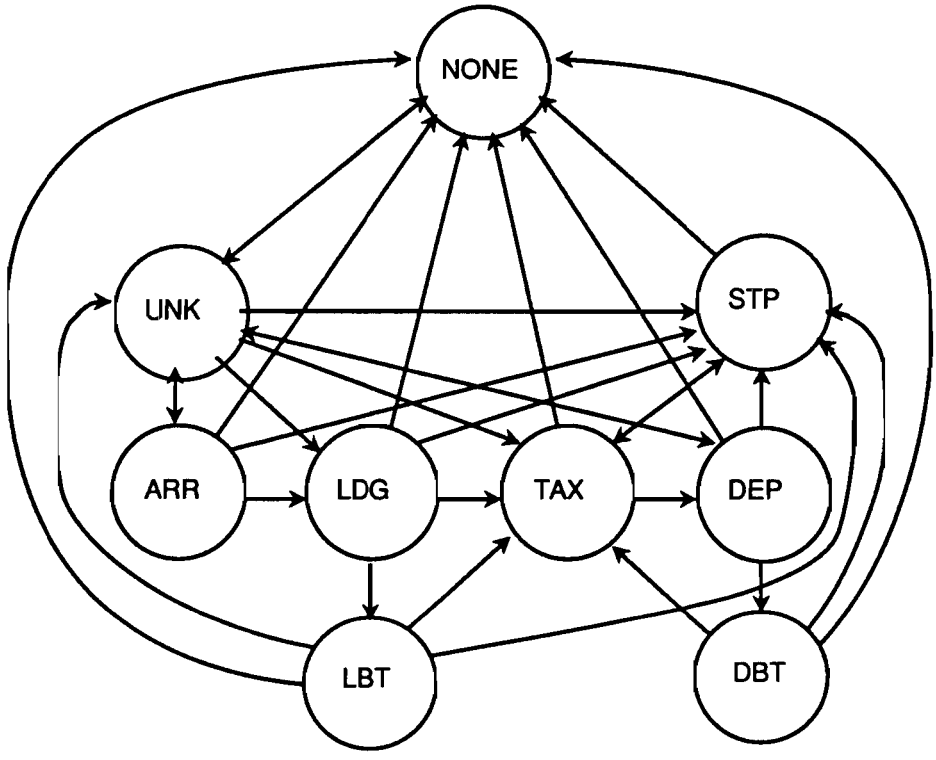


Figure 6-2. Target state machine.

[1] In this chapter, unless otherwise stated, the term "velocity" refers to the magnitude of the velocity, i.e., speed. Also, v_tax- is the minimum velocity for the TAX state, and v_stp+ is the maximum velocity for the STP state. Appendix B contains the values of these parameters.

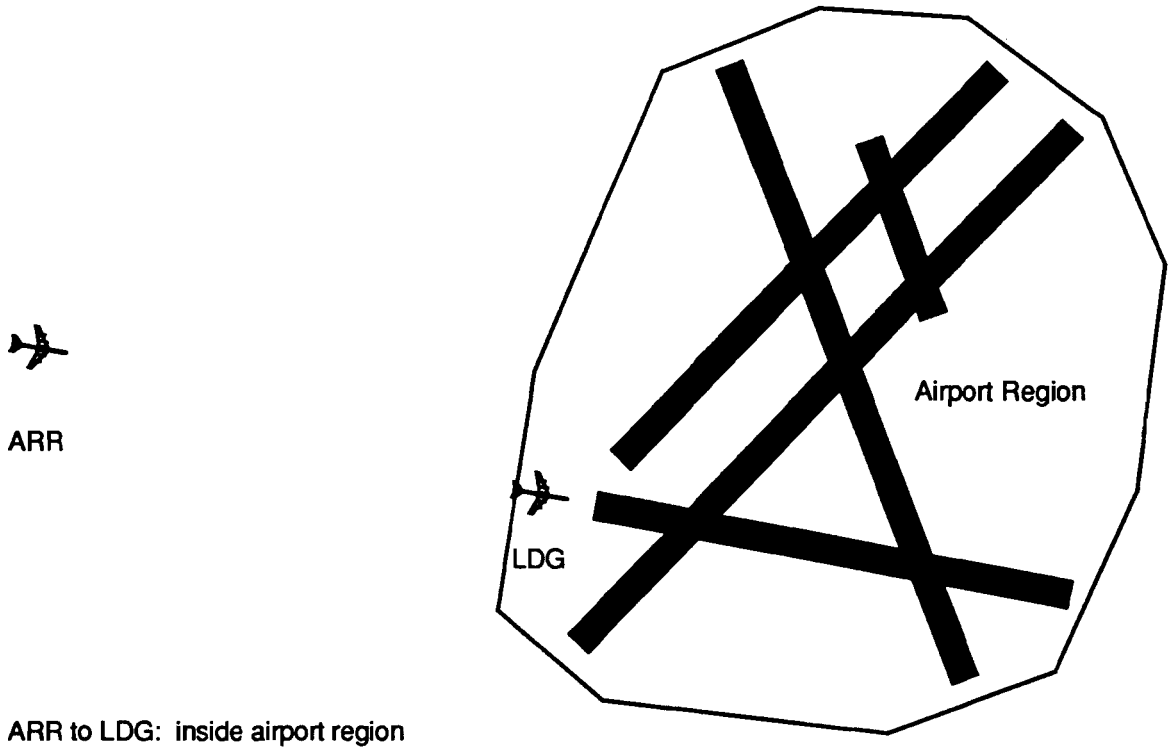


Figure 6-3. Target-state transition from arrival state (ARR) to landing state (LDG).

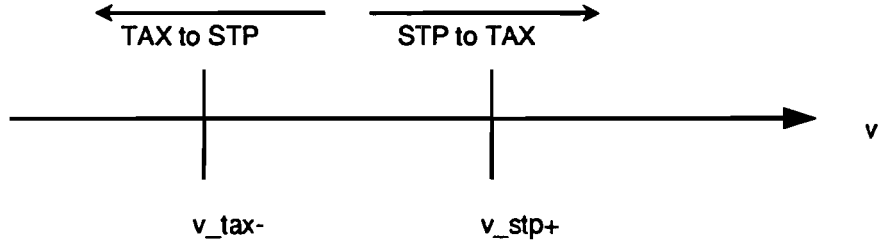


Figure 6-4. Target-state transitions between stopped (STP) and taxi (TAX) states.

6.3 PREDICTION ENGINE

6.3.1 Overview of the Prediction Engine

To control the runway-status lights and generate alerts, the RSLS safety logic requires knowledge of the predicted positions of all tracked targets. The set of algorithms that determines where a target could be in the future is called the "prediction engine." Since the prediction engine takes into account that targets can make turns at intersections of runways and taxiways, a target can have more than one predicted path. Rather than come up with a single set of predicted paths, the prediction engine computes *two* sets of predicted paths as illustrated in Figure 6-5. One set, shown as dotted lines, assumes the target will travel as far as reasonable within a given look-ahead time. The other set, shown as a solid line, assumes the target is attempting to stop (but not a panic stop) within a given look-ahead time. Thus, the two sets of predicted paths, or "trees," compute reasonable bounds on future target position. In other words, the RSLS path-prediction algorithm does not answer the question "where *will* the target be in a given t seconds?" but rather the question "where *could* the target be in a given t seconds?"

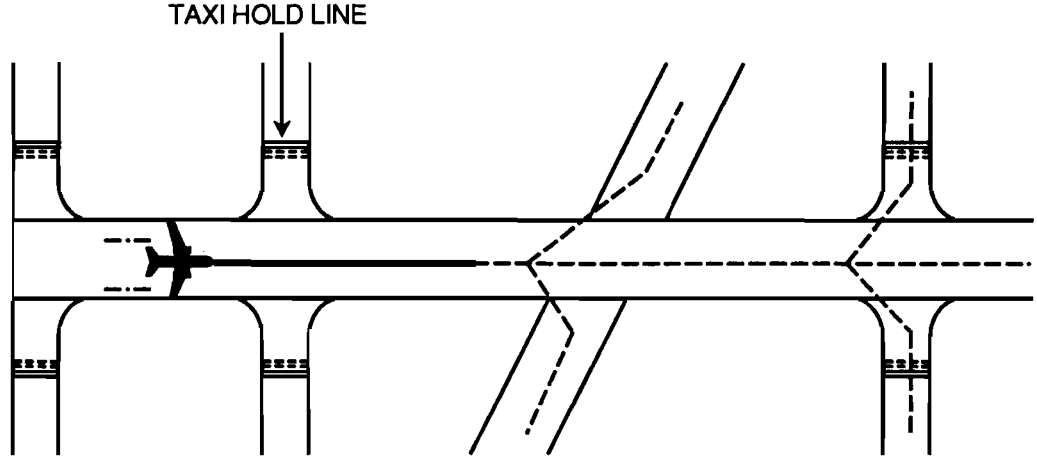


Figure 6-5. Path-prediction trees.

To determine these path-prediction trees, the prediction engine requires prediction models of target motion. There are two models for each target state. The first one, called the "acceleration model," is used to determine the first set of predicted paths (i.e., the furthest the target could be). The second one, called the "deceleration model," is used to determine the second set of predicted paths (i.e., the nearest the target could be). It should be noted that these models by themselves do not determine future position, because

they do not take into account the airport geometry (i.e., which paths can be taken) and models of how targets would turn from one path onto another path at any intersection.

6.3.2 Prediction Models of Target Motion

Table 6.2 describes the prediction models as a function of target state. All predictions start with the estimates of current target position and velocity from the sensor-fusion and tracking algorithms. In general, the standard equations of motion for a given look-ahead time are used to predict future position; for example, the look-ahead time is 60 seconds for targets in the high-speed states DEP, ARR, and LDG. These predictions use either a nominal acceleration rate (accel_nom) or nominal deceleration rate (decel_nom), which are functions of target state. These rates could be functions of target type once target type is reliably known to the safety logic. The question might be asked, "Why not use the estimates of *actual* acceleration instead of assumed nominal acceleration and deceleration?" The answer is that estimates of actual acceleration are relatively noisier, and thus less accurate, than those of velocity. Also, actual accelerations are not constant; since they vary unpredictably with time, estimates of current acceleration are not useful for predicting future behavior on the 30-to-60-second time scale.

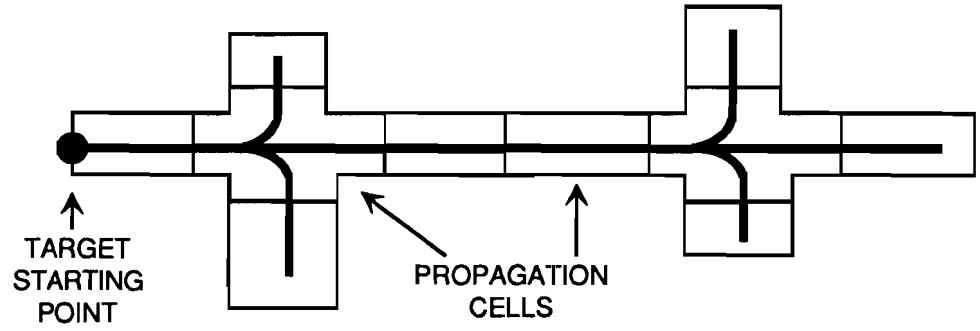
TABLE 6.2
PREDICTION MODELS FOR EACH TARGET STATE

Target State	Acceleration Model	Deceleration Model
stopped (STP)	NA	NA
taxi (TAX)	accelerate at accel_nom(tax) up to a specified maximum taxi velocity v_max_tax, and then continue at that velocity	decelerate at decel_nom(tax)
departure (DEP)	accelerate at accel_nom(dep) to a maximum departure velocity v_max_dep	decelerate at decel_nom(dep), i.e., becomes a departure abort (see below)
landing (LDG)	continue at current velocity (i.e., accel_nom(ldg) = 0)	a) if before threshold, then same as acceleration model b) if after threshold, then decelerate at decel_nom(ldg)
arrival (ARR)	continue at current velocity (i.e., accel_nom(arr) = 0)	a) if before threshold, then same as acceleration model b) if after threshold, then decelerate at decel_nom(ldg), i.e., becomes a landing
departure abort (DBT)	continue at current velocity (i.e., accel_nom(dbt) = 0)	decelerate at decel_nom(dbt)
landing abort (LBT)	continue at current velocity (i.e., accel_nom(lbt) = 0)	NA
unknown (UNK)	continue at current velocity (i.e., accel_nom(unk) = 0)	decelerate at decel_nom(unk)

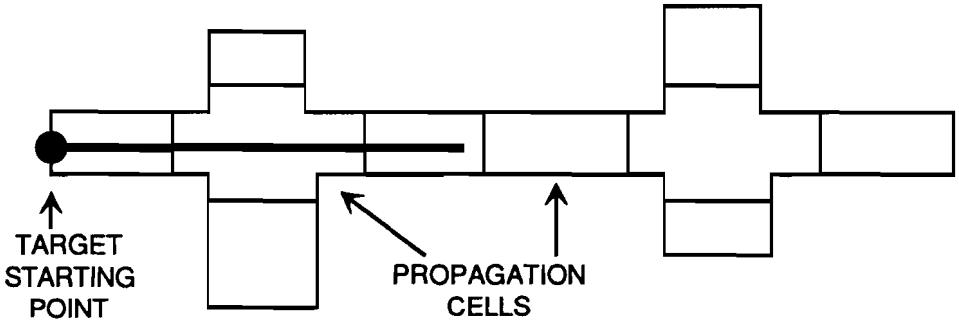
NA = not applicable

6.3.3 Path-Prediction Algorithms

To predict target paths, the prediction engine first divides the airport surface into a network of "propagation cells" that completely cover the movement area and do not overlap. For example, a set of cells are drawn end-to-end along the entire length of a runway. The same is done for taxiways (see Figures 6-6a and 6-6b for examples of propagation cells). Thus, the cell structure depends on the airport geometry. The important point to understand about path prediction is that it is done locally within each cell, then all the cells are stitched together to generate the entire path.



a) acceleration model



b) deceleration model

Figure 6-6. Prediction trees for models of target motion.

The procedure to generate the paths is as follows. First, the prediction engine determines in what cell a new target is located and what the possible exit points are. Second, using the prediction models of target motion described above, standard Newtonian equations of motion, and a turning model that assumes a maximum transverse acceleration (described in more detail in Section 7.7), the prediction engine determines which exit points the target can reach (if any) in the look-ahead time. Third, for each

exit point the target can reach within the look-ahead time, the prediction engine draws a path to that point and hands off the target to the next propagation cell, thus creating a local "tree" with side branches as shown in Figure 6-6a. After each path has been drawn as far as it can go within the look-ahead time, the final result is a set of two global trees, one for the acceleration model of target motion (as shown in Figures 6-5 and 6-6a) and one for the deceleration model (as shown in Figures 6-5 and 6-6b). These trees are the unions of the individual propagation-cell trees.

6.3.4 Approach Logic and Approach Bars

One of the tasks of the prediction engine is to implement the approach logic, which determines whether an airborne target is on approach to a runway and, if so, which runway. As part of this logic, the prediction engine projects an arriving target and determines whether it could land on a particular runway. To make this determination, the prediction engine uses a turning model that allows for S-curves as well as straight-in approaches and single-curved approaches.[2]

Another task of the prediction engine is to calculate distances for approach bars. As mentioned in Chapter 1, approach bars are ASDE-display enhancements that indicate to the controller the position of the aircraft on final approach to the airport. As illustrated in Figure 6-7, an approach bar appears on the ASDE display near the approach end of a runway; its length represents the distance between the outer marker and the runway threshold. A small diamond moving along the bar represents the actual aircraft on final approach and indicates its position relative to the outer marker and threshold as determined by the prediction engine. The RSLS gives the controller the following four display options for the approach bars: 1) show no approach bars, 2) show approach bars only when a target on approach is involved in an alert, 3) show approach bars only when there is a target on approach, and 4) always show approach bars, even when there is no target on approach.

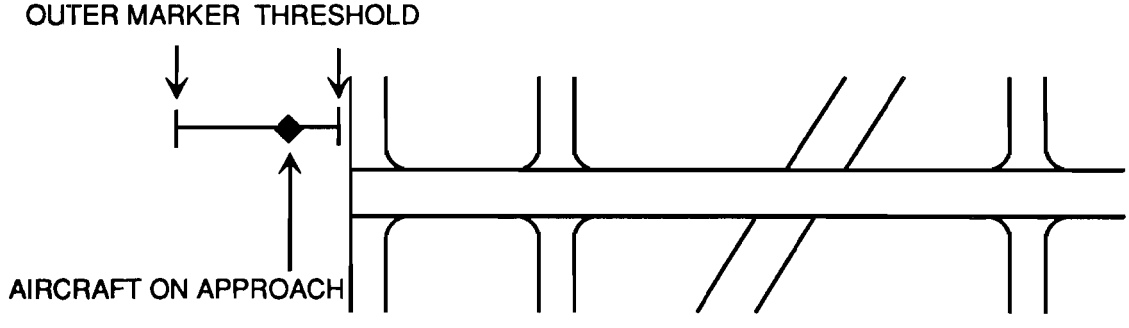


Figure 6-7. Indication of arriving target on approach bar.

If it is possible for an aircraft on approach to land on more than one runway (i.e., the prediction engine projects it onto more than one runway), then the aircraft is called an *ambiguous* target. However,

[2] The details of the approach logic are described in Appendix A and in Section 7.7.

ambiguous targets still appear on only one approach bar and are associated with only one runway for turning on runway-status lights and issuing alerts. The logic rule for selecting the approach bar is to indicate the target on the approach bar of the runway with the lowest required transverse acceleration, i.e., the runway corresponding to the straightest of the approaches. The logic rule for choosing the runway for turning on lights and issuing alerts is to assign the target to the runway in use for landings in the current configuration. For example, at Logan Airport aircraft arriving to runway 33R often follow the ILS approach to runway 27 and then veer off to 33R. The safety logic at first places the aircraft on the 27 approach bar and then the 33R approach bar. If runway 27 is being used only for departures in the current configuration, then the safety logic assumes the aircraft will land on 33R and not land on 27. If both runways 27 and 33R are being used for landings, then a parameter can be set to indicate to the logic which runway should be assumed to be the landing runway. However, if the wrong runway is selected, the lights and alerts will not be operating correctly. A better way to handle this problem is to use the part of the ARTS data that specifies the aircraft's landing runway. This capability has not yet been implemented.

6.4 LOGIC FOR RUNWAY-ENTRANCE LIGHTS

6.4.1 Overview of the REL Logic

As discussed in Section 1, red runway-entrance lights (RELs) at a runway/taxiway or runway/runway intersection indicate that it is unsafe to enter the runway at that intersection. It is unsafe because there is a high-speed target (e.g., DEP or LDG) moving along the runway toward that intersection; controllers call the runway "hot" under these circumstances. As illustrated in Figure 6-8, the logic for the runway-entrance lights is based on three concepts. The first concept is the *target hot zone*. The hot zone is an area ahead of a high-speed target, measured from the front of the target's extent (i.e., from the nose), that should be free of other targets; thus, entry into the hot zone by other aircraft, ground vehicles, or personnel would be unsafe. The second concept is the *REL activation region*, which is associated with each set of independently operated RELs. The basic idea behind the logic rule for RELs is as follows: the lights are red if a hot zone overlaps their associated REL activation region, and off otherwise. In particular, the lights behind the target are always off. An exception to this rule is when RELs inside a hot zone are off due to the application of a third concept, called *anticipated separation*, which will be described in Section 6.4.3.

In Figure 6-8 the RELs are represented by rectangles at runway/taxiway intersections (although this is not the way they are represented on the RSLS displays). At the runway/runway intersection, the question marks indicate that the logic for runway/runway intersections is still under development (Section 6.4.4 will discuss runway/runway intersections).

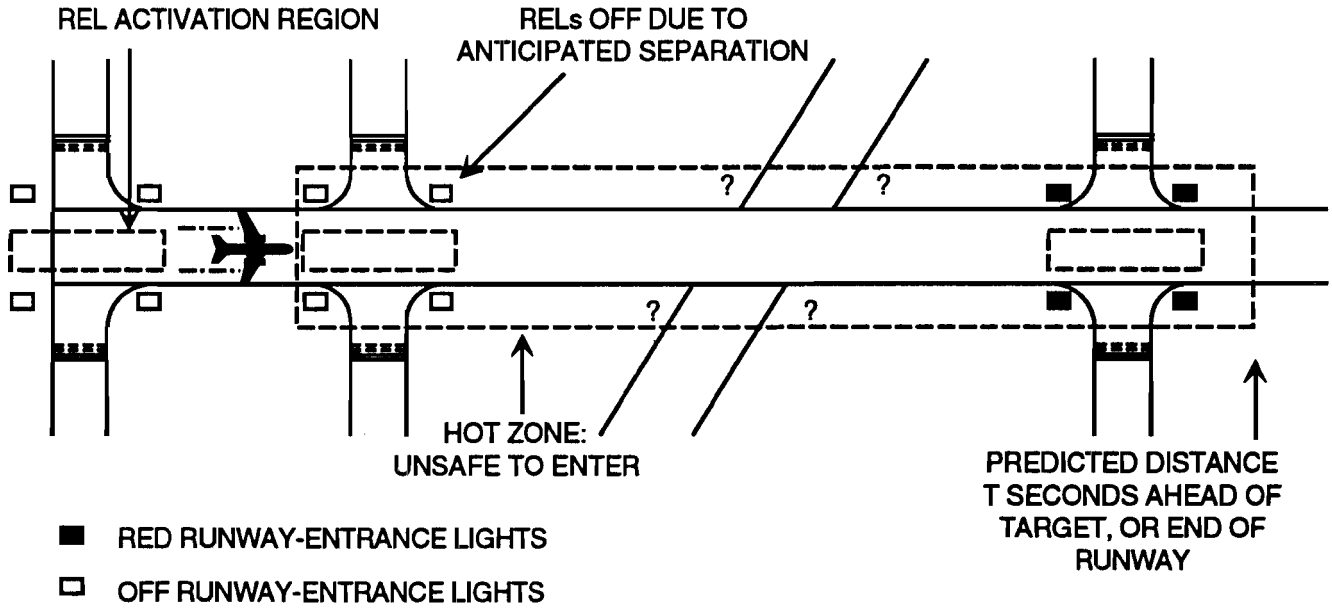


Figure 6-8. Hot zone and REL activation regions for runway-entrance lights.

6.4.2 Target Hot Zones

There are two types of hot zones each with a different type of length: 1) t-second zones, whose length is the distance corresponding to *t* seconds ahead of the target (e.g., t = 30 seconds), where *t* is a function of target state and a particular set of RELs, and 2) whole-runway zones, whose length is the whole runway ahead of the target. The length of a t-second zone is calculated by the prediction engine using the target's acceleration model of target motion.

The length of the hot zone depends on the target's state. Table 6.3[3] shows the hot zones being used for the target states in the current version of the safety logic. Arrivals (ARR) have a t-second zone, where *t* is called t_arr_rel in the parameter list in Appendix B. Departures (DEP) "own" the whole runway, and thus have whole-runway zones, unless they are airborne (i.e., altitude is above the parameter altitude alt_dep_rel), in which case their hot-zone lengths are zero. Targets that are DBT or LBT also own the whole runway, because they are in unusual and potentially dangerous states. Targets that are TAX, STP, or low-speed UNK, where "low-speed" means velocity less than v_tax+, have zero hot-zone lengths at this time, because their velocities are so low that it is often safe for other targets to cross the runway ahead of them. However, UNK targets with velocity greater than v_tax+ have t-second zones.

Landing targets (LDG) are divided into three cases. If they are on approach and have not yet crossed the threshold, they have t-second zones. Once they cross the threshold, they "own" the whole runway, and thus have a whole-runway zone until they are "under control," that is, until their velocity has dropped below some parameter velocity called v_ldg_rollout. When they are under control they are considered to be in a landing rollout and then have a t-second zone again.

---

[3] See Appendix B for the values of the parameters, e.g., t_ldg_rel.

TABLE 6.3
HOT-ZONE LENGTHS

Target State	Hot-Zone Length Ahead of Target
ARR	t_arr_rel seconds
LDG	1) before the runway threshold: t_ldg_rel seconds 2) across the runway threshold but before landing rollout: whole runway 3) during landing rollout: t_ldg_rel seconds
DEP	whole runway (if not airborne)
DBT	whole runway
LBT	whole runway
TAX	zero
STP	zero
UNK	1) high-speed: t_unk_rel seconds 2) low-speed: zero

6.4.3 Lights at Runway/Taxiway Intersections

This section will describe the logic for RELs at runway/taxiway intersections along active and inactive runways. A runway is *active* if it is being used primarily for takeoffs and/or landings in the current airport configuration. A runway is *inactive* if it is not active but may be used temporarily for takeoffs and landings at any time.

As mentioned in Section 6.4.1, the safety logic uses the concept of a *REL activation region*, an area on the runway at an intersection, as shown in Figure 6-9. Each activation region is associated with a set of RELs (i.e., a set consists of a pair of lights on either side of a taxiway or runway at the entrance to a runway). No set of RELs can respond to more than one REL activation region, but a single REL activation region may control more than one set of RELs. The light-control logic turns on a set of RELs (i.e., sends the command to turn them red) if a target's hot zone overlaps the REL activation region associated with the RELs, except when anticipated separation applies, as will be described below.

There are two ways to determine whether a hot zone overlaps a REL activation region, depending on the type of hot zone. If the target has a whole-runway zone, then the hot zone overlaps all REL activation regions ahead of it. If the target has a t-second zone, the safety logic first considers the REL activation regions that the target could reach within its look-ahead time. It then compares t to the time it would take the target to reach those REL activation regions. Expressed mathematically, the hot zone overlaps a REL activation region if the following is true:

$$t\_min\_enter < t\_state\_rel, \tag{6-1}$$

where t_min_enter is the minimum time for the target to enter the REL activation region, as calculated by the prediction engine using the target's *acceleration* model, and t_state_rel is the parameter from Table 6-3 with "state" representing the target state (state = arr, ldg, or unk).

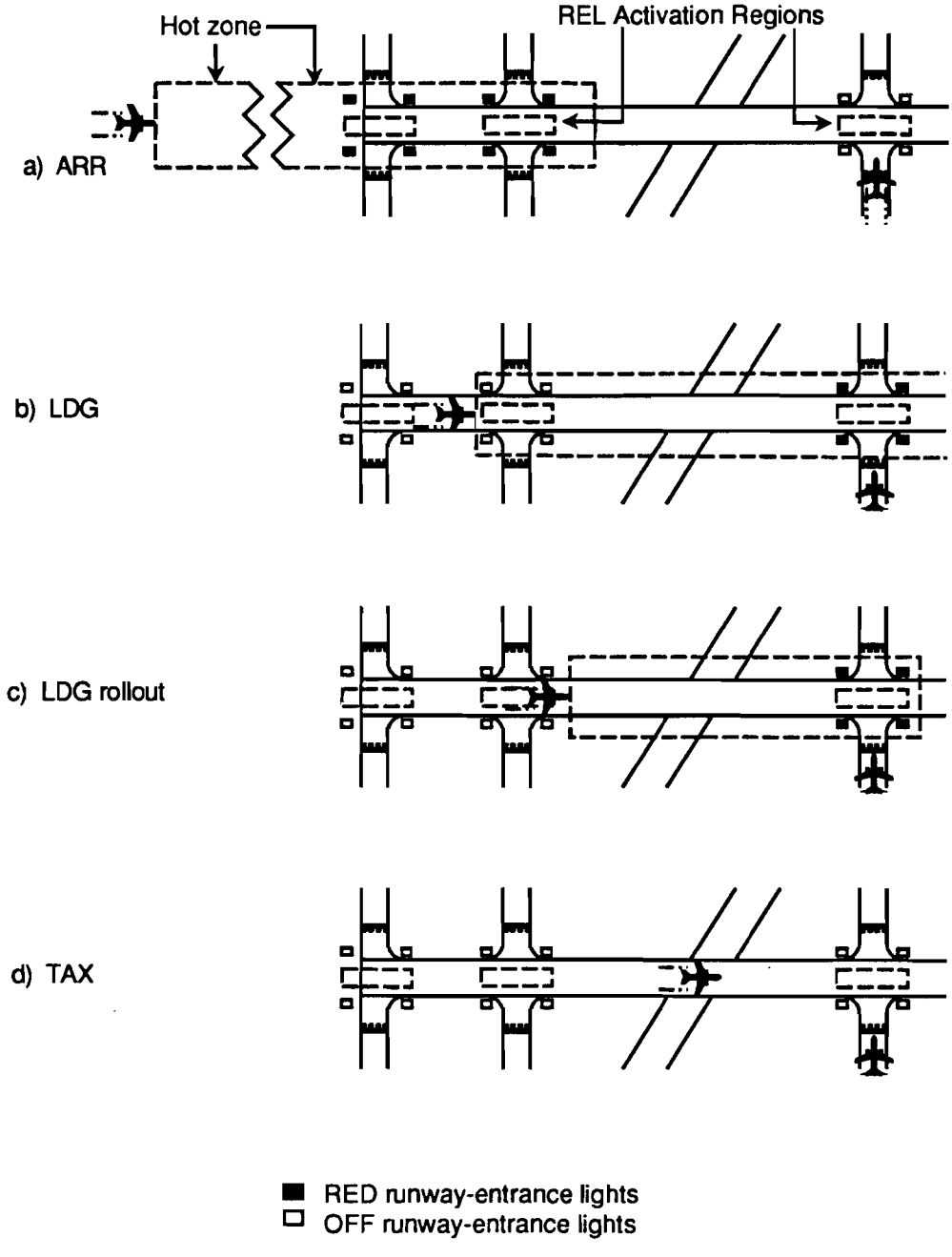


Figure 6-9. Examples of logic for runway-entrance lights.

There are several ways to turn RELs off. One of the ways uses the concept of *anticipated separation.* This term refers to the fact that controllers can issue clearances and instructions to aircraft in anticipation that legal separation between aircraft will exist when required, even though legal separation does not currently exist [1]. For example, it is not legal for an aircraft to cross a runway in front of a departure. However, a controller can issue "taxi across" instructions to an aircraft waiting to cross a runway at a taxiway hold line, even though a departure on that runway has not yet passed the taxiway intersection, in anticipation that, by the time the waiting aircraft starts across, the departure *will* be past the intersection. The delay in crossing is due to the time it would take the waiting aircraft to spool up its engines before it started across.

To avoid interference with the controllers and to allow for surveillance delays, the light-control logic uses anticipated separation to turn off the RELs before a target's hot zone has passed through the REL activation region at approximately the time when a controller would issue the "taxi across" instruction. Thus, the logic sends the command to turn off a set of RELs if the target that turned the RELs red is a time t_antic_sep_rel seconds from exiting the REL activation region. This condition can be expressed mathematically as the following:

$$t\_max\_exit < t\_antic\_sep\_rel, \tag{6-2}$$

where t_max_exit is the maximum time for the target to exit the REL activation region, as calculated by the prediction engine using the target's *deceleration* model, and t_antic_sep_rel reflects the anticipated-separation time (e.g., 4 seconds, representing the spool-up time of a waiting aircraft). In Figure 6-8 the first set of RELs inside the hot zone is off due to anticipated separation.

Other ways to turn RELs off, besides anticipated separation, are as follows: 1) the target track is dropped, 2) the target is a DEP and has altitude greater than a parameter altitude alt_dep_rel (if there is a valid altitude), 3) in the case of t-second zones, the hysteresis threshold is exceeded (i.e., t_min_enter ≥ t_state_rel + delta_t_state_rel), and 4) in the case of whole-runway zones, the target changed to a state where the hot zone is no longer a whole-runway zone. If the RELs were turned off due to the second condition, then they can be turned on again only if the target has transitioned out of the DEP state.

6.4.4 Lights at Runway/Runway Intersections

As mentioned in Section 6.4.1, the safety logic for RELs at runway/runway intersections is handled differently than at runway/taxiway intersections. At first, one might ask, "Why not use the same logic rules as for runway/taxiway intersections?" However, there is a difference: at runway/runway intersections, a target observing red RELs is likely to be traveling at high speed, such as a departure, arrival, or landing on an intersecting runway. This is particularly true for active runways but would also be true for inactive runways that are *temporarily* being used for takeoffs and/or landings. The concern is that a pilot traveling at high speed might take an unsafe action upon seeing red lights.

Consequently, the decision was made in the RSLS to deactivate the RELs at runway/runway intersections except for those cases when a runway is consistently being used only as a taxiway in the current configuration. For example, at Logan Airport in Boston landing aircraft on runway 4R generally exit onto runway 33R. Since all aircraft on 33R must cross runway 4L to go to the terminal, and 4L is usually an active runway whenever 4R is active, the RSLS *does* activate the RELs facing targets on 33R/15L at the intersection with 4L for those configurations where 4R is active and 33R/15L is always

inactive.[4] As will be discussed in Section 6.7, a topic for further research is to determine logic rules for RELs at runway/runway intersections when both runways are active.

6.4.5 Examples of Runway-Entrance Lights

Figures 6-9(a - d) illustrate the operation of the runway-entrance lights through a series of examples involving an arriving aircraft. The black rectangles at runway/taxiway intersections represent the RELs that are red; the white rectangles represent RELs that are off. The hot-zone lengths are listed in Table 6-3.

Figure 6-9(a) shows an aircraft arriving to the runway (ARR state). The dotted rectangle represents the aircraft's hot zone, which extends ahead t_arr_rel seconds. For example, if t_arr_rel is 30 seconds and the aircraft's velocity is 130 knots along the runway, the target acceleration model for ARR computes the length of the hot zone to be (t_arr_rel) * (velocity of the target) = 6630 feet (about 1.1 nautical miles). All RELs whose activation region overlaps the hot zone are red; none are off due to anticipated separation, because ARR is too far away from the activation regions to satisfy the anticipated separation condition. The RELs are off at the last intersection, because their activation region does not overlap the hot zone.

Figure 6-9(b) shows the same aircraft after it has entered the airport region, so that it is now in the landing state (LDG). Assuming the aircraft is across the threshold and at high-speed, i.e., its velocity is greater than v_ldg_rollout (e.g., 55 knots), the hot zone extends to the end of the runway. The RELs at the first intersection are off, because the hot zone no longer overlaps the activation region at that intersection. The RELs at the second intersection are off because of anticipated separation. However, the RELs at the last intersection are red, because the hot zone overlaps the activation region there, and the target is too far away from the activation region to satisfy the anticipated separation condition. The aircraft on the taxiway is waiting behind the hold line and is facing red lights. If a controller accidentally were to issue instructions for it to cross the runway, the pilot would know that the runway was not safe to enter and would question the controller before proceeding across.

Figure 6-9(c) shows the aircraft in a landing rollout, i.e., it is in the LDG state, but its velocity is less than v_ldg_rollout. The hot zone is now t_ldg_rel seconds ahead. The RELs are off at the second intersection, because the hot zone no longer overlaps the activation region.

Figure 6-9(d) shows the aircraft after it has slowed down to TAX. Its hot-zone length is zero (i.e., t_tax_rel = 0), so that no RELs are red.

6.5 LOGIC FOR TAKEOFF-HOLD LIGHTS

6.5.1 Overview of the THL Logic

The light-control logic determines whether takeoff-hold lights (THLs) next to active or inactive runways should be red or off. Although inactive runways are not generally used for takeoffs, they may be used for takeoffs at any time, so that the logic for takeoff-hold lights should apply to inactive runways as

---

[4] Appendix C lists the runway/runway intersections where RELs are activated for each configuration.

well as active runways. Also, THLs are at locations used for intersection takeoffs as well as full-length takeoffs.

Takeoff-hold lights at a given location are red if two conditions are satisfied: 1) a target is in position for takeoff or starting its takeoff at this location, and 2) the runway is not safe for takeoff at this location. The first condition implies that there must be a target in a position to see the THLs in order for them to be red. This is not true for RELs, which turn red regardless of whether there is a target in a position to see them. The second condition depends on whether there is another target that could come in conflict with a departure. This condition is also different from the logic for RELs, in that THLs turn red based on the actions of *two* targets, whereas RELs turn red based on the action of a *single* target. These conditions are described in more detail in the next three sections.

6.5.2 Target In Position for Takeoff or Starting Its Takeoff

As shown in Figure 6-10, the logic to determine whether a target is in position for takeoff or starting its takeoff has been divided into two cases. In Case 1, the target is defined to be "in position for takeoff," because it satisfies the following conditions: 1) it is in the STP state, 2) its centroid is in a specified area on the runway, called the *arming region* (shown in Figure 6-10 as a dotted rectangle), and 3) it was not previously in the LDG state. The last condition, which is optional and can be set to apply to only certain runways for each configuration, was added to prevent THLs from turning red when a previously-landed aircraft temporarily stops in an arming region at a location on the runway used for intersection takeoffs. Since this aircraft is not likely to take off (because it just landed), a red THL would be a nuisance light. Also, as shown in Figure 6-10, the THLs are placed ahead of the departure location, and the arming region extends all the way to the THLs.

In Case 2, the target is "starting its takeoff," i.e., it is in a rolling takeoff or in the early stage of a takeoff. There are five conditions a target must satisfy to be considered starting its takeoff. First, it could be in either the TAX or DEP state. Second, as for Case 1, its centroid must be in the arming region. Third, to rule out the situation where the target is crossing the runway, the target must be sufficiently lined up with the runway, i.e., the difference between the target's heading and runway heading must be within a specified tolerance. Fourth, to rule out the situation where the target is passing through the arming region at high speed, the target's velocity must be below a given parameter. The fifth condition is that the target was not previously LDG for the same reasons stated for Case 1 above.

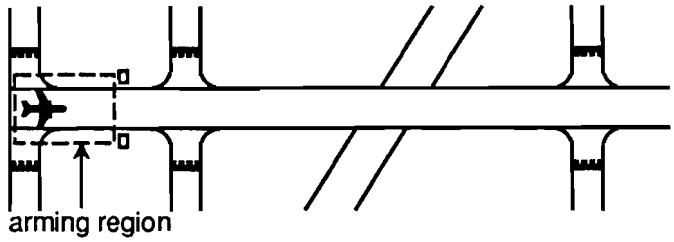
6.5.3 Targets on the Same Runway

As mentioned in Section 6.5.1, the second condition for THLs to be red is that the runway must be unsafe for takeoff. This condition can be divided into two cases depending on whether there is another target on the *same* runway as the target about to take off or there is another target on an *intersecting* runway. This section will describe the former case; the next section will describe the latter.

In the former case, the THL logic rule states that the runway is unsafe for takeoff if there is a target inside the *THL activation region*, an area that includes the runway ahead of the lights as well as an extension on either side of the runway. As in the description of the target hot zone, the THL activation region is an area ahead of a target in position for takeoff, or starting its takeoff roll, that should be free of other targets before takeoff commences. The logic rule is illustrated in Figure 6-11, which shows that the THLs are red, because one target is in position for takeoff and another target is inside the THL activation

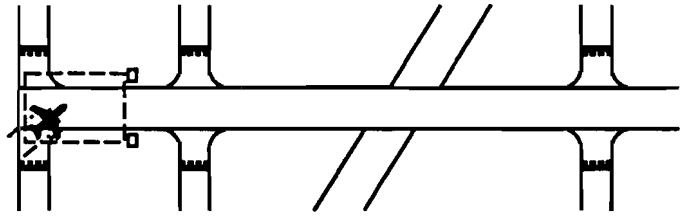
region. If there had been no target in the arming region the THLs would be off, even though the runway is unsafe for takeoff.

Case 1: Target in Position for Takeoff



1. STP
2. centroid in arming region
3. not previously LDG

Case 2: Target Starting Its Takeoff



1. TAX or DEP
2. centroid in arming region
3. target heading within specified tolerance of runway heading
4. target velocity below specified threshold value
5. not previously LDG

□ Takeoff-Hold Lights

Figure 6-10. Illustration of the first condition for takeoff-hold lights to be red.

There are two exceptions to the logic rule that the runway is unsafe for takeoff if there is a target in the THL activation region. The first exception states that the runway *is* safe for takeoff if the target in the THL activation region is predicted to exit "soon." This is illustrated in Figure 6-12, which shows that the THLs are off. Exiting "soon" means that target B is predicted to exit the THL activation region before target A could reach target B if A started to take off. The mathematical expression for this condition is the following:

$$t\_max\_exit < t\_min\_reach, \tag{6-3}$$

where t_max_exit is the maximum time for B to exit the THL activation region, as calculated by the prediction engine using B's *deceleration* model, and t_min_reach is the minimum time for A to reach B using the *acceleration* model for departures (DEP). This rule is motivated by another example of *anticipated separation* (see Section 6.4.3), in which it is legal for a controller to clear a stopped aircraft for takeoff in anticipation that the target on the runway will exit before the aircraft starts its takeoff roll [1]. Although this rule has been designed, it is not yet implemented in the code.

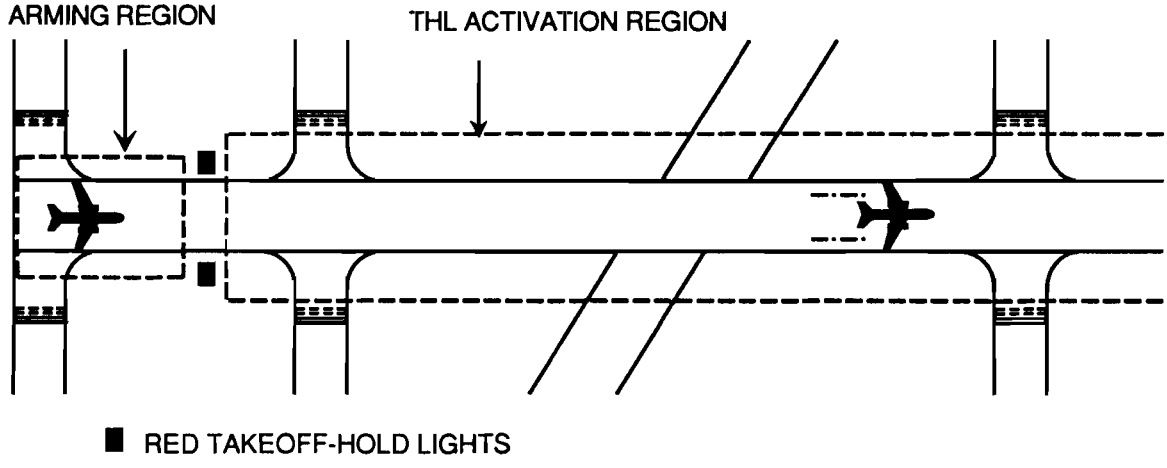


Figure 6-11. Runway is unsafe for takeoff, because a target is in the THL activation region.

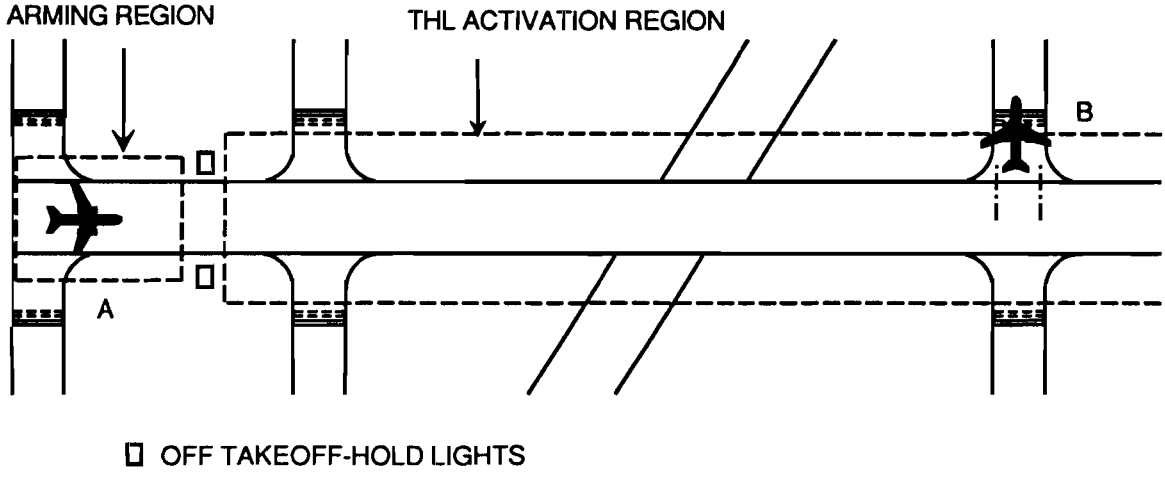


Figure 6-12. Runway is safe for takeoff, because B is predicted to exit the THL activation region before A could reach B if A started to take off (not yet implemented).

The second exception, illustrated in Figure 6-13, involves what is referred to in the safety logic as the "special departure (DEP) case;" if a prior departure is far down the runway and is airborne, then the runway is safe for takeoff even though to the ASDE the departure appears to be in the THL activation

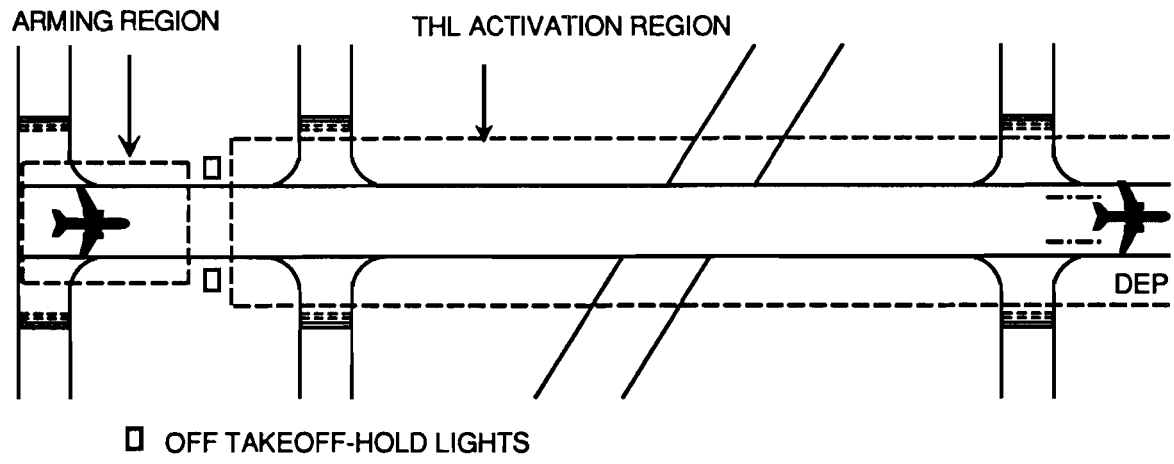


Figure 6-13. Runway is safe for takeoff, because the target inside the THL activation region satisfies the conditions for the special DEP case.

region. This idea is based on controller procedures in which it is legal to have an aircraft take off on a runway when a previous departure is still over the runway as long as certain conditions are met. These conditions are as follows: 1) the distances can be determined by suitable landmarks, 2) the distance between the aircraft is greater than a specified distance that depends on aircraft type, 3) the downstream aircraft is airborne [1].

The light-control logic cannot check these conditions, because it knows neither aircraft type, nor whether distances can be determined by landmarks, nor, in some cases, whether an aircraft is airborne (even though the ARTS data sometimes indicate altitude). Consequently, the light-control logic will check other conditions that approximate these conditions and still provide a margin of safety. They are as follows (refer to Figure 6-13): 1) B is DEP, 2) visual flight rules (VFR) are in effect (this information must be entered manually into the safety logic), 3) B is downstream from A on A's runway, 4) B's distance from A is greater than a parameter distance, and 5) B's altitude is deemed valid by sensor fusion and B is airborne, or B's altitude is not valid and B's velocity is greater than a parameter velocity.

Another condition for the runway to be unsafe for takeoff is for a target to be a TAX and be *predicted to enter* the THL activation region. This condition is illustrated in Figure 6-14, which shows that the THLs are red. The prediction engine determines that a target is predicted to enter a region if all the target's paths based on the deceleration model for TAX enter the THL activation region; in other words, B will definitely enter the region within the TAX look-ahead time even if it starts to brake. This condition is not currently implemented in the safety-logic software.

There is an important point that should be noted here regarding how the prediction engine determines whether there are two targets in potential conflict. Rather than compare all pairs of tracked targets to determine whether A and B might collide, which would be a huge computational problem that grows as N^2, where N is the number of targets, the prediction engine has A and B each "notify" the

runway. Then the prediction engine tells the safety logic to compare only A and B, which is a more-manageable approach than a combinatorial one.[5]

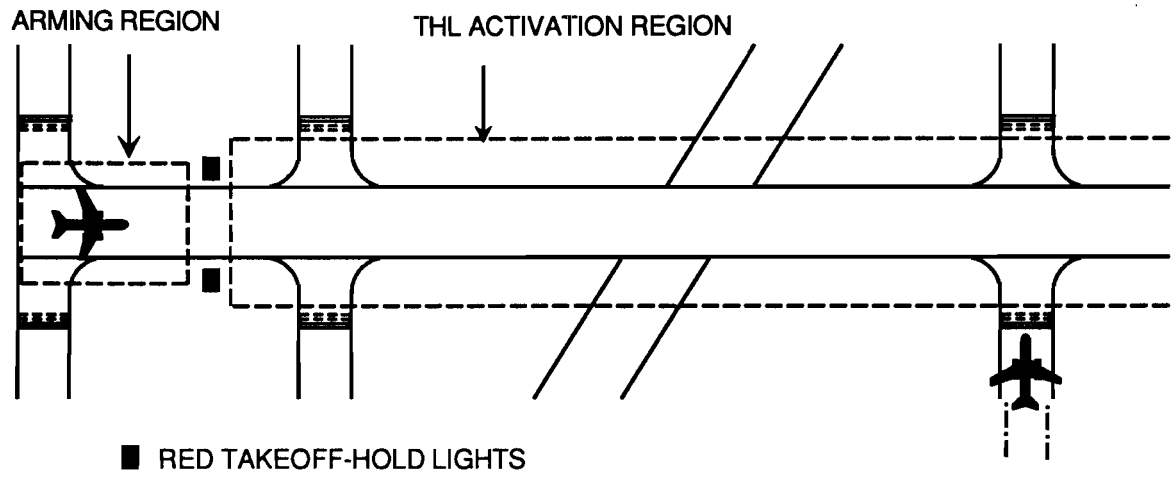


Figure 6-14. Runway is unsafe for takeoff, because the target is predicted to enter the THL activation region (not yet implemented).

6.5.4 Targets on Intersecting Runways

A runway can also be unsafe for takeoff when there is a potential conflict with a high-speed target on an intersecting runway. The logic for this case, which is illustrated in Figure 6-15, is based on the concept of an *intersection window*, an area at the intersection of two runways. The runway is unsafe for takeoff if target A, which is in position for takeoff or starting its takeoff, and target B, which is in any target state except STP and TAX, could be in the intersection window simultaneously if A started to take off. This logic rule again uses the concept of anticipated separation. Rather than wait for B to cross the intersection before turning off the THLs, the logic turns off the lights in anticipation that B will be across the intersection before A could be in conflict with B if A became a departure.

According to the rule described in the previous section, as B crosses A's runway the THLs would turn red while B is in the THL activation region along A's runway. To prevent this from happening, the safety logic has a mechanism that keeps track of whether B is on an intersecting runway or along the same runway as A. If B is on an intersecting runway, the THLs for A will not turn red.

The prediction engine computes the time interval A could be in the intersection window and the time interval for B. The logic rule then becomes: the runway is unsafe for takeoff if the time intervals of A and B overlap with a safety margin time added to the time intervals. The prediction engine uses the acceleration and deceleration models of target motion to compute the time intervals. For target B, the earliest time to enter the window is calculated using its acceleration model, and the latest time to exit the

[5] This is described in more detail in Section 7.7.

window is calculated using its deceleration model. Target A is usually stopped or moving very slowly, so its deceleration path is likely to end before the intersection window. Consequently, both its earliest time to enter the window and latest time to exit the window are calculated using the acceleration model for departures.[6]

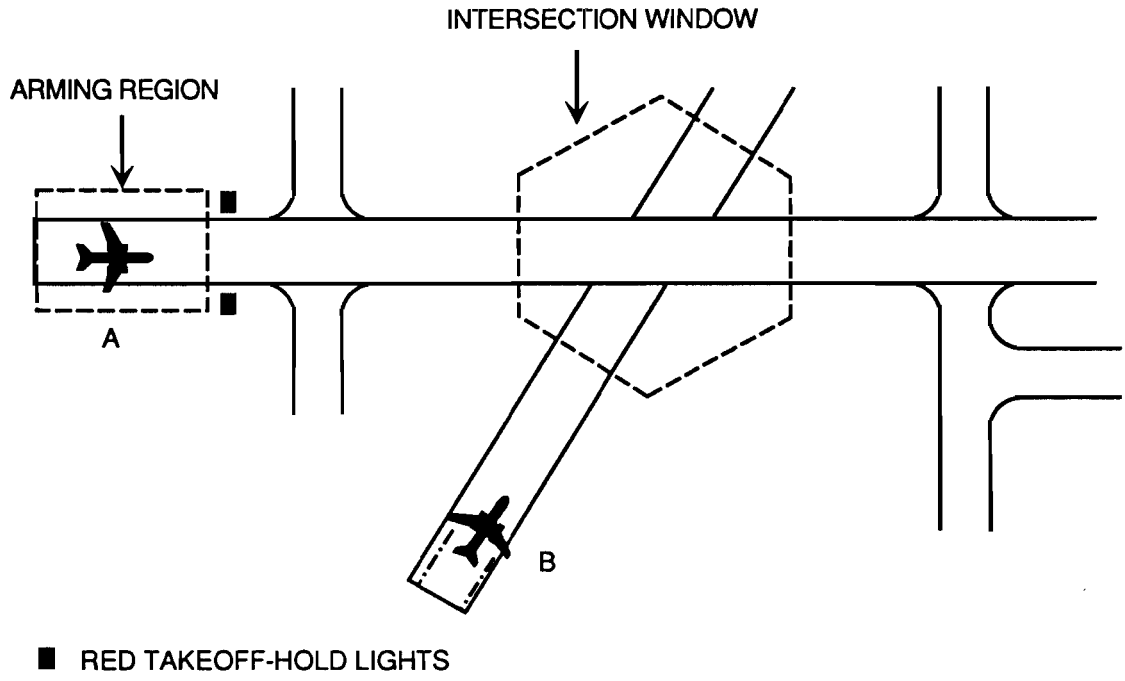


Figure 6-15. Runway is unsafe for takeoff, because A and B could be in the intersection window simultaneously if A started to take off.

As mentioned in Section 6.5.3, the prediction engine does not compare all pairs of targets to determine which pairs are potentially in conflict, thus avoiding a huge computational burden. The prediction engine has A and B each "notify" the intersection window and then tells the THL logic to compare only A and B.

6.6 LOGIC FOR THE ARR/STP AND LDG/STP ALERT

The safety logic currently implements one type of alert, which, as mentioned in Section 6.1, was added for demonstration purposes. This alert, referred to as the ARR/STP or LDG/STP alert and illustrated in Figure 6-16, involves one target (target A) stopped on a runway and another target (target B) on approach to the runway or landing on the runway. More precisely, A is STP, and B is ARR or LDG, or B is UNK with velocity greater than v_tax+. When an alert has been issued, octagons are drawn around the conflicting targets, even if one of the targets is displayed on an approach bar, and an alert

[6] The mathematical expressions are in Appendix A.

message is sent to the audible alerting system. For example, the alert message for an arrival to runway 22L is as follows: "Warning. Two-Two-Left Arrival, Traffic on Runway."

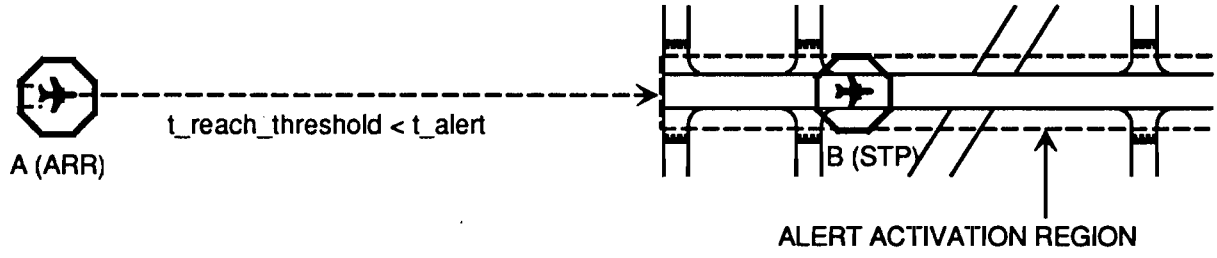


Figure 6-16. Example of ARR/STP alert.

The alert logic is based on the concept of an *alert activation region*, an area around the runway that the prediction engine checks to see if there are any targets that may potentially be in conflict with target B. An alert is issued if two conditions are met: 1) B is "close to" or on the runway, and 2) there is a target A stopped in the alert activation region. If B is airborne, the first condition for issuing an alert can be expressed mathematically as

$$t\_reach\_threshold < t\_alert, \tag{6-4}$$

where t_reach_threshold is the time it would take for B to reach the runway threshold as calculated by the prediction engine using B's acceleration model, and t_alert is a parameter threshold (e.g., 35 seconds). If B is already past the threshold, t_reach_threshold is set equal to zero. The reason the logic does *not* compare t_alert to the time for B to reach *A* is the following: it is important to alert soon enough so that B has time to execute a go-around. This timing is independent of A's location in the alert activation region, so that it is better to measure B's time to a fixed point, such as the threshold or missed-approach point. Since missed-approach points can vary depending on aircraft type, it was decided to use the threshold instead.

Analogous to the discussion in Sections 6.5.3 and 6.5.4, the prediction engine avoids the N^2 computational problem by having A and B "notify" the runway. Thus, the alert logic needs to compare only the two targets A and B instead of all pairs of targets.

6.7 FUTURE TOPICS FOR DEVELOPMENT

This chapter has presented a high-level description of the algorithms currently implemented for the RSLS safety logic. However, there is still a lot of work to be done to have a complete safety system. Some examples of topics that need to be developed in the future are the following:

1. **Logic for stopways:** Boston's Logan Airport has long, crossing stopways at the approach ends of runways 4L and 9. These stopways also intersect with several taxiways. On the model board of Logan Airport, there are RELs at the intersections of taxiways with these stopways. However, there currently is no light-control logic to drive the RELs. Also, the stopways are not included in any activation regions for RELs, THLs, or alerts, so that these stopways are not included in the safety logic. The logic for them has been designed but not yet implemented.

2. **Anticipated separation for takeoff-hold lights:** As mentioned in Section 6.5.3, the THL logic involving anticipated separation for targets on the same runway, in which a target is predicted to exit the THL activation region or enter the THL activation region, has been designed but not yet implemented. However, the THL logic involving anticipated separation for targets on intersecting runways *has* been implemented.

3. **Logic for runway-entrance lights at runway/runway intersections:** As mentioned in Section 6.4.4, the logic for controlling the RELs at runway/runway intersections has not yet been completely resolved. Various logic schemes could be tested with real and simulated data to determine whether lights at these intersections would interfere with normal operations.

4. **Logic using target classification:** Knowledge of target type, e.g., aircraft type or whether the target is a ground vehicle, would greatly improve the safety logic. First, testing of the logic has shown that a ground vehicle is sometimes determined to be a DEP or LDG; knowledge of target type would solve this problem. Second, some of the safety-logic parameters should be a function of target type, e.g., the acceleration and deceleration parameters in the prediction models of target motion, to make the aircraft modeling more accurate. Third, knowledge of target type would help prevent having helicopters triggering RELs to turn red when they should be off.

5. **Controller interface:** A controller interface is needed for the tower supervisor to input information to the safety logic about current conditions, such as meteorological conditions, surface braking conditions, and the current runway configuration. The latter information would indicate which runways are active or inactive, and which areas are closed to any operations.

6. **Incorporating controller/pilot intent:** The current logic is not aware of controller or pilot intent. This causes a problem in certain cases, such as sidesteps and land-and-hold-short operations. In the latter case, the logic does not know when an approaching aircraft has been told by the controller to hold short of an intersecting runway. It assumes that the aircraft has permission to use the whole runway for its landing. This may cause some problems. For example, if there is also a target in position for takeoff on the intersecting runway, then the THLs for that target will turn red, even though the controller may have cleared the target for takeoff (which is legal). This would cause interference with controller operations. On the other hand, if the logic assumes the approaching target *will* hold short of the intersecting runway, but the target is *not* intending to hold short, then the logic does not provide protection between the two targets, i.e., the logic's effectiveness has been reduced. The controller interface just described above would allow the controller to input to the safety logic whether an aircraft is intending to land and hold short.

7. **Smooth transitions between configurations:** Currently, when the airport supervisor changes the runway configuration, software engineers must stop the RSLS surface-monitor software, input the new configuration, and then bring up the system again. Even if a controller interface is added to allow the tower supervisor to specify the current configuration, it would be useful to be able to keep the system running and to transition smoothly during configuration changes.

8. **Other alerts:** As mentioned in Section 6.6, only one alert has been implemented. Many other alerts that have already been designed could be easily added, such as a head-on alert between a DEP and a TAX on the same runway (logic rule: alert immediately). Other alerts need to be designed first and then implemented, such as a tail-chase alert between an ARR to a runway and a TAX downstream on the same runway.

9. **Hot-zone lengths:** One future change in the hot-zone lengths for LDG and ARR targets is to switch to whole-runway zones a fixed time t_threshold seconds before the runway threshold rather than waiting for the target to cross the threshold. This would indicate that an aircraft owns the whole runway a little before it crosses the threshold.

10. **ARTS data for landing aircraft:** As mentioned in Section 6.3.4, the safety logic needs to know the intended landing runway for ambiguous approach targets, especially when both runways that a target can land on are being used for landings in the current configuration. The ARTS data can provide that information, but use of this information by the safety logic needs to be implemented.

REFERENCES

1. U.S. Department of Transportation, Federal Aviation Administration, "Air Traffic Control," FAA Handbook 7110.65F with Changes 1-5 (21 September 1989).

7. RSLS SYSTEM SOFTWARE ARCHITECTURE

7.1 INTRODUCTION

The RSLS system executes primarily on multiple networked workstations that use the UNIX operating system. The ADIDS system, which uses IBM PC-compatible machines that use the DOS operating system, and the ASDE data conversion and distribution system, which is based on a Mercury single board computer, are exceptions. The run-time software consists of approximately 175K lines of C++ code to implement the radar processing, sensor fusion, safety logic, and displays and an additional 15K lines of C code to implement the ASDE radar interface and ADIDS system. This software is organized into a dozen independently executing processes (tasks) that communicate with each other via operating system and network resources.

The multi-process approach minimizes complexity and facilitates the use of multiple processors in areas where more processing horsepower is required. It is based on dividing the major algorithmic tasks into separate data-driven programs. The programs communicate with each other so that the output of one program (algorithmic step) becomes the input to the next program. In some cases, the communication is bi-directional to allow for cases where there is feedback or a query-response structure in the flow of data through the algorithmic steps. The multi-processing nature of the UNIX operating system directly supports this approach. The UNIX kernel also provides direct and flexible support for communication between processes that reside within one computer or within multiple computers connected via a network.

7.2 SYSTEM OVERVIEW

Figure 7-2-1 shows the context data flow diagram for the RSLS system. Figure 7-2-2 shows the top-level data flows and functional elements of the RSLS software itself. In these data flow diagrams, boxes with rounded corners represent sources or sinks of data outside the RSLS system. Rectangles represent functional elements. Ovals indicate primary databases. The arrows reflect the flow of data between elements.

There are three sources of data flowing into the system: digitized radar video from the X-band ASDE, beacon replies from the ASR-9-SCIP-to-ARTS-III IOP interface, and display symbology from the ARTS-III-IOP-to-MDBM interface. The X-band ASDE video is an 8-bit digitization of the log magnitude of the received return for each of the radar's range-azimuth pixels. The beacon replies from the ASR-9 indicate each target's range, azimuth, mode-A-squawk code, mode-C altitude, and various confidence and validity fields. The data extracted from the IOP-to-MDBM interface includes all text and symbol data destined for the controller displays attached to the MDBM. For targets, this text and symbol data includes the target position, the symbol to display at that position, and, for targets with data blocks, the information to display in the data block and the direction of the data block's leader line.

ASDE processing subtracts clutter from the radar image, identifies targets in the clutter-removed image, finds the centroid (position) of the targets, and then matches the target to targets in previous scans to form tracks. Because no single workstation-class processor is capable of handling the high input data

rate and processing demand, the ASDE processing is implemented as three processes: clutter rejection, connected components, and merge. These processes execute on an eight-processor Silicon Graphics workstation. The input data is distributed to multiple instances of the clutter rejection and connected components processes executing in parallel and the intermediate results are recombined in a final merge process.

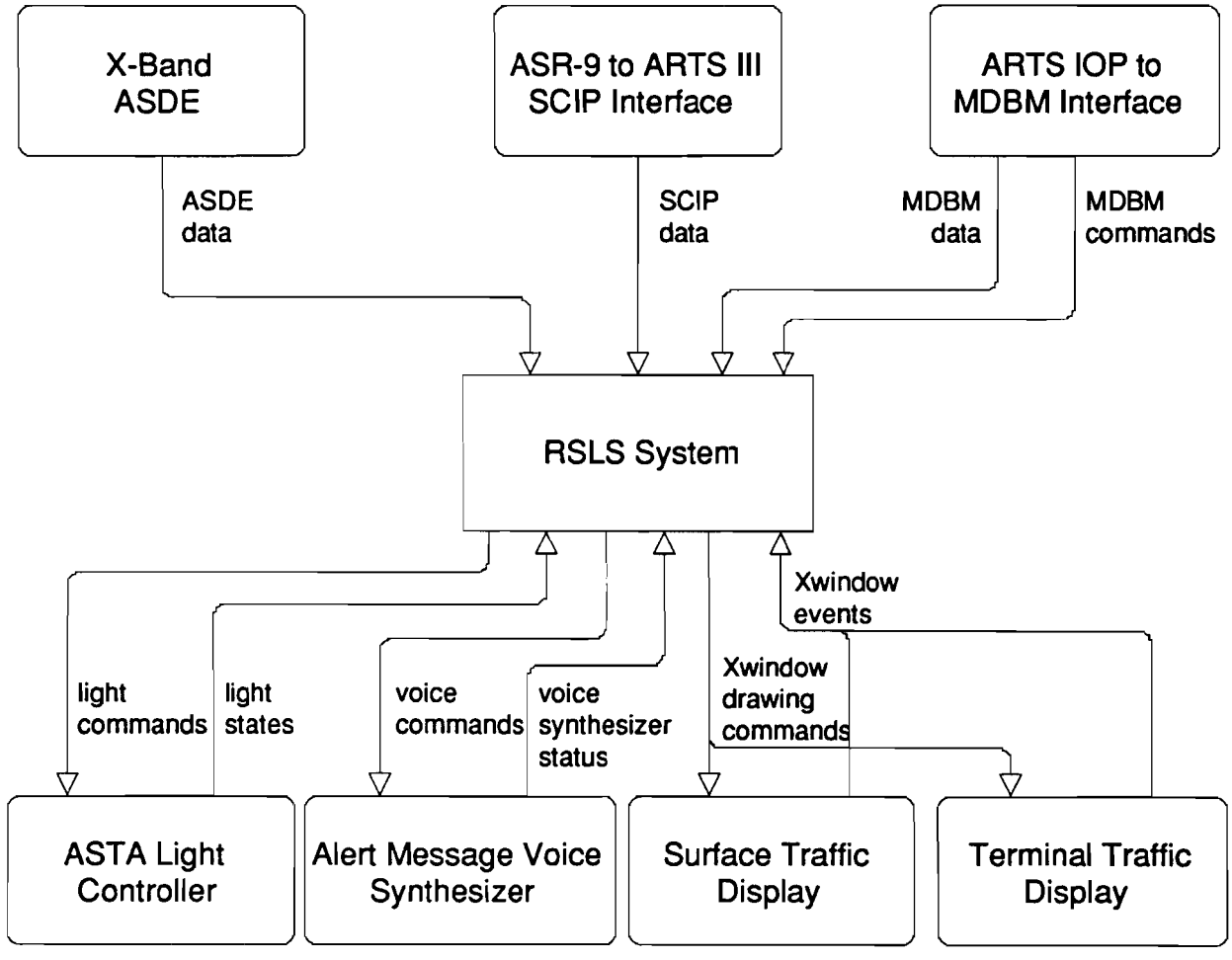


Figure 7-2-1. The context data flow diagram for the RSLS Software.

The SCIP interface collects surveillance data from the ASR, applies geometric and message-type filters, decodes the packed binary format used by the ARTS into standard data types, and adds a time stamp. Because the location of the ARTS hardware is physically remote from the majority of the RSLS hardware, the SCIP interface is implemented in two parts. The first part, ADIDS-SCIP, which executes

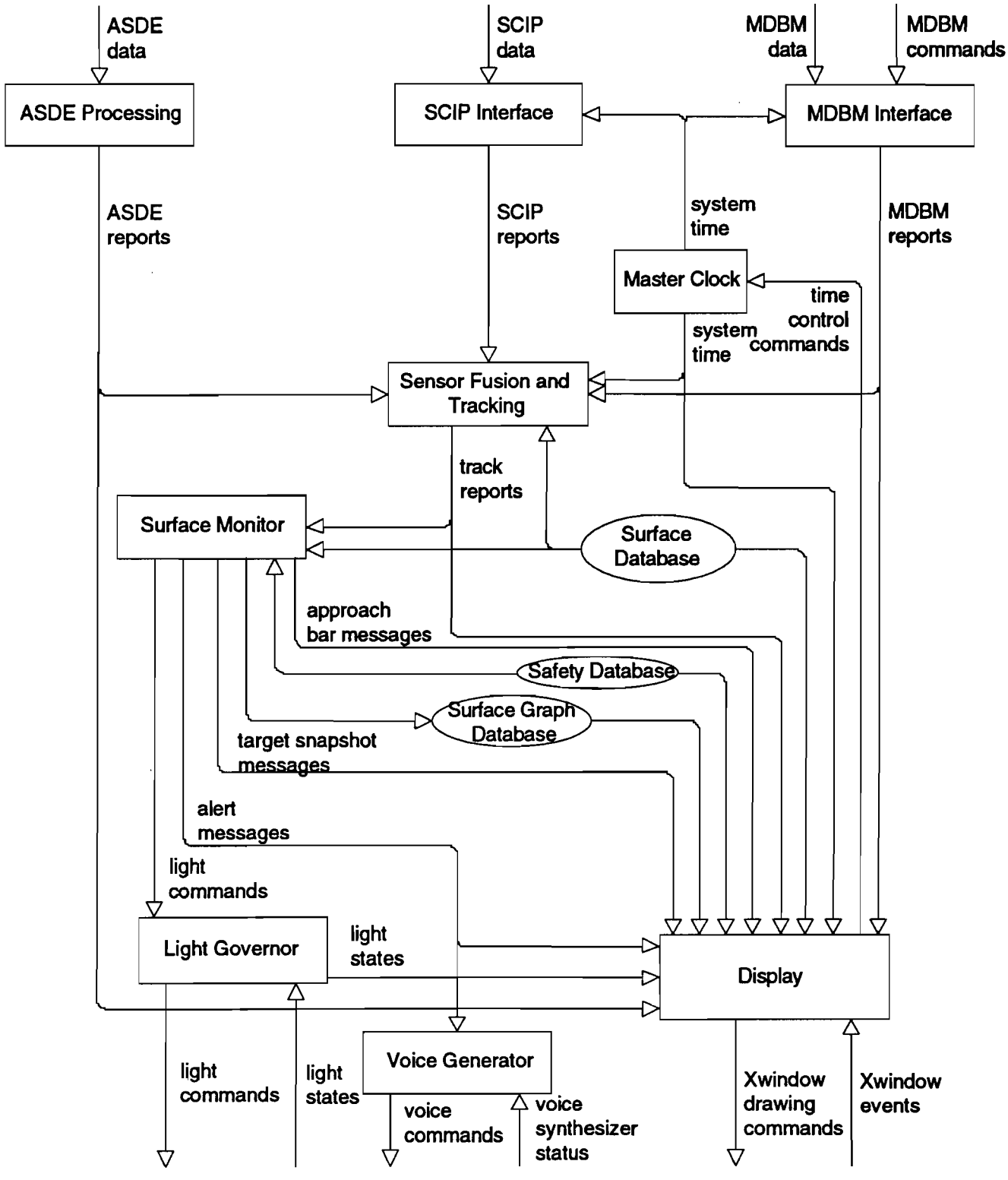


Figure 7-2-2. The top-level data flows and functional elements of the RSLS software.

on a PC adjacent to the ARTS hardware, extracts the data, applies geometric and message-type filters, adds a message identifier and time stamp, and transmits the data over a modem. The second part, which executes on a UNIX workstation, receives the data via a second modem, unpacks it, and makes it available to the rest of the system.

The MDBM interface listens to data from the ARTS-IOP-to-MDBM link, emulates the updating of tables performed in the MDBM, extracts target updates, applies geometric and message-type filters, decodes the packed binary format used by the ARTS into standard data types, parses portions of the data block for targets with tags, and adds a time stamp. Because the location of the ARTS hardware is physically remote from the majority of the RSLS hardware, the MDBM interface is implemented in two parts. The first part, ADIDS-MDBM, which executes on a PC adjacent to the ARTS hardware, extracts and interprets the data, applies message-type and geometric filters, adds a message identifier and time stamp, and transmits the data over a modem. The second part, which executes on a UNIX workstation, receives the data via a second modem, unpacks it, parses portions of the data block for targets with tags to extract aircraft identity and airframe type, and makes the data available to the rest of the system.

Sensor fusion receives the target data generated by ASDE processing, beacon data from the SCIP interface, and flight-identity and airframe-type data from the MDBM interface and produces a unified picture of the aircraft and vehicles that are on or near the airport surface. Sensor fusion's responsibilities include the following:

- translating all target positions to a common coordinate system
- determining and applying time offsets to synchronize time stamps
- smoothing target positions
- estimating target velocity and acceleration
- de-multipathing the beacon reports
- filtering false targets from the ASDE data
- transferring identity and type information from the MDBM data to the beacon data
- determining which ASDE targets and beacon targets represent the same target and should be combined into a single track
- coasting tracks through coverage gaps.

The surface monitor uses the output of sensor fusion and several databases to control the operation of the lights, to generate audible alerts, and to drive the depiction of targets on approach on the displays. For each target, the surface monitor determines the target's state, calculates the range of possible future positions within a defined time horizon, assesses whether the state of the lights should change in response to the target's new state, and determines if an alert should be generated due to possible interactions with other targets.

The light governor processes light state commands from the surface monitor and uses them to send commands to the light control system. It forwards light state change messages to the displays when the

light control system confirms that the light has turned on or off. It uses a database to correlate the light IDs used by the surface monitor with the light channel numbers used by the light control hardware.

The voice generator formats alert commands from the surface monitor into messages suitable for a voice generation system (i.e., DECtalk). In case of temporally overlapping alerts, it queues messages according to the priority that the surface monitor has assigned to each. It uses a database of message templates.

The display generates a plan-view of the current situation. It is normally used show the position (and other characteristics such as identity, type, heading, radar image size, and alert status) of the tracked targets, the state of the RSLS lights, and the approach bars for targets on approach to the runways. It can also show additional information generated by the system such as all targets identified by ASDE processing (unfiltered), target fusion status generated by sensor fusion, and target states and predicted paths generated by the surface monitor. It uses a map database to provide visual reference.

The master clock is a time server that provides a common time reference to all processes. In real-time operation, the master clock is used to provide synchronized time stamps on data recorded by multiple processes. In data playback mode, the master clock allows the system to run at varying time rates.

7.3 RADAR PROCESSING

7.3.1 Radar Processing Overview

The algorithmic description of the Radar processing was discussed in an Section 3 of this document. Figure 7-3-1 shows an algorithmic block diagram. The following stages are included:

1. ASDE-X Data Extraction
2. Clutter Rejection
3. Morphological Pre-Processing
4. Connected Components
5. Components to Objects
6. Scan-to-Scan Association.

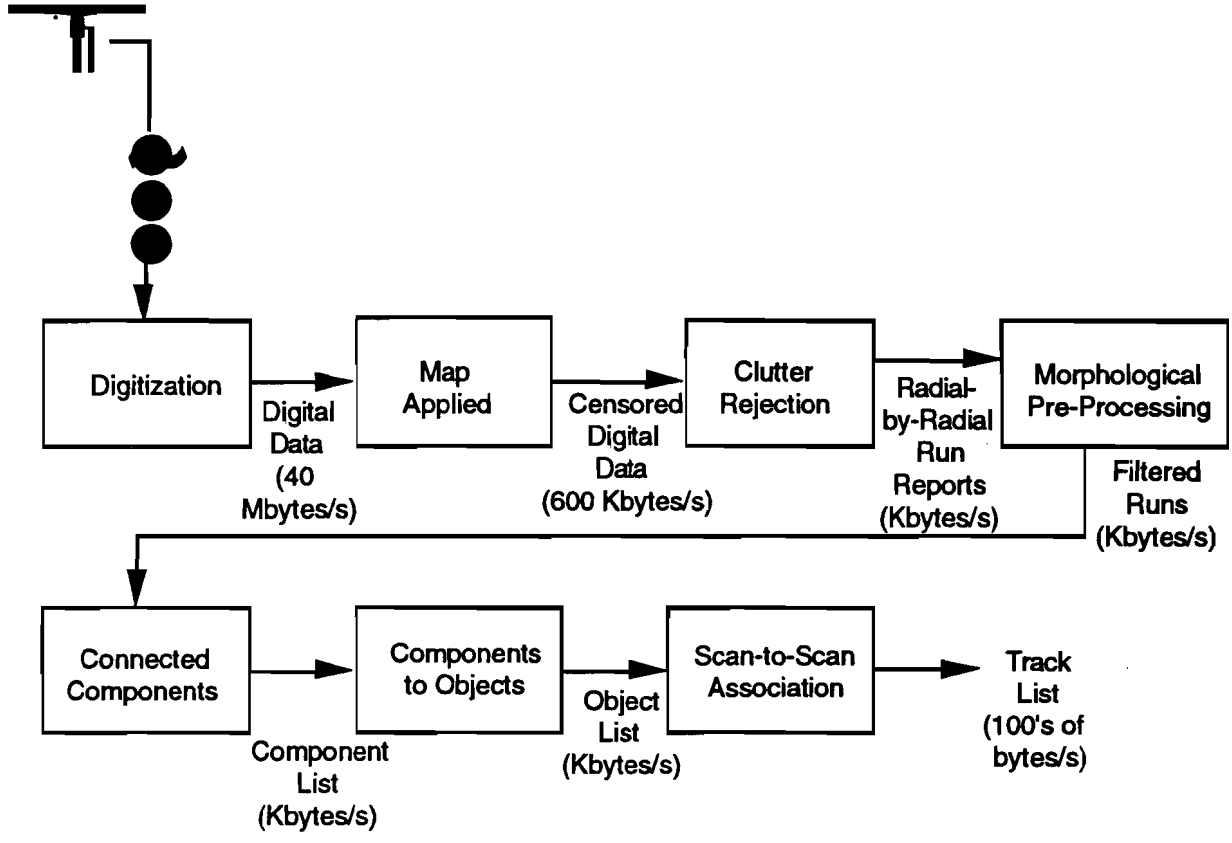


Figure 7-3-1. Algorithmic block diagram for X-Band surveillance processing.

It is entirely possible to create a version of software that performs these operations serially. However, the advantages for real time of parallelizing several of these operations motivated a rather elaborate scheme of dividing the functionality between several processes. Figure 7-3-2 provides an overview of the major processes.

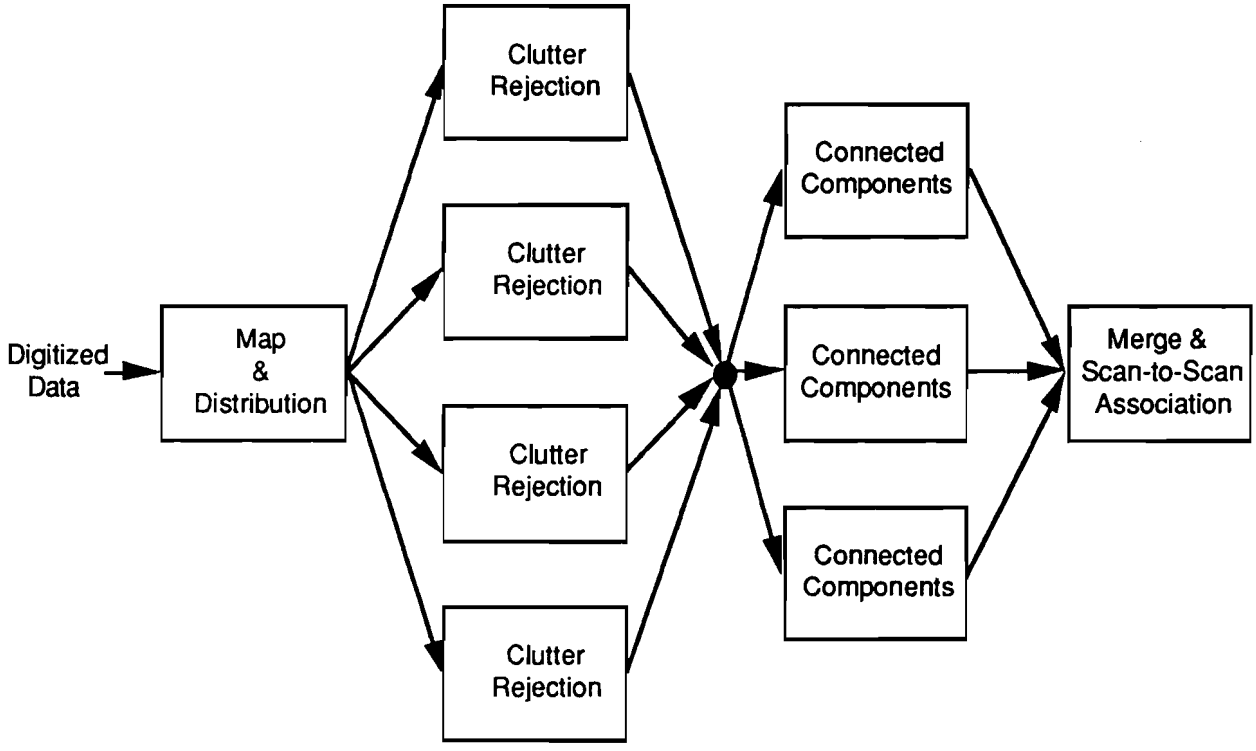


Figure 7-3-2. Parallel process overview.

Initially, the analog radar data is digitized and enters the system at a very high rate (typically 40 Msamples/s). Using custom hardware, only data from the airport surface and approaches are passed on for subsequent processing. The surviving data is passed to a general-purpose single-board computer that serves to distribute data to several clutter rejection processes based on airport region. Section 7.3.2, Radar Interface, describes this stage of processing.

Four clutter rejection processes, running on processors within the SGI multi-processor system, each cover about a fourth of the airport samples. An individual process always sees the same airport region. These processes maintain a surface-sample-by-surface-sample history of radar returns for use in a

dynamic CFAR mechanism that detects target pixels. Rather than reporting each target pixel individually, they are strung together to form runs that are sent to the connected components processes. Each process sends out these runs in n-radial packets with n currently being 128. Section 7.3.3, Clutter Rejection, describes details of this mechanism. It should be noted that while the data to the three processes is fed sequentially, due to processing delays some packets from the end of one processor's region may be completed and sent after the first packets from the next region. It should also be noted that since the number of operations per pixel is almost constant, load balancing for this stage can be performed statically (before run time).

It was found that the number of runs emerging from the clutter rejection processes were sufficient to require parallelization of the algorithm. This leads to two difficulties. First, while most objects may be completely contained in the "wedges" sent from clutter rejection, some straddle boundaries. Secondly, connected components load balancing between processes is completely data dependent.

The first problem is handled as follows (see Figure 7-3-2): connected components is performed on an entire airport surface wedge. Components are then grouped together into *objects* (see Section 7.3.4, Connected Components, below for more detail). Objects are then sorted into two lists. The first consists of all objects totally contained within the surface wedge. The second list consists of objects that straddle the surface wedge boundary (see Figure 7-3-3). Packets containing these lists are sent by several connected components processes to a single merge process whose responsibilities include splicing together boundary objects.

The second problem (load balancing) is handled by a first-come-first-served scheduling algorithm for the connected components processes. Each clutter rejection process places its result on its own queue with a time stamp. All available connected components processes compete for access to the set of clutter rejection output queues, which is mediated by a semaphore. The "winner" checks all queues for data and chooses the oldest packet and releases the semaphore. All connected components processes maintain no history – i.e., after processing a packet and sending off the results, no information from it is maintained. Thus, when a packet is finished, a connected components process is available for processing any other packet.

As the last stage of radar surveillance processing, the merge/scan-to-scan association process must collect data from all the connected components processes and produce a view of the whole surface as well as maintain history on all known tracks. In this single process, a table of wedges already seen in the current scan is maintained and updated. As a new wedge is received, the targets completely contained within it are immediately put on a new target list. If any boundary targets exist, the "wedges already seen" list is checked to see if the appropriate neighbors are present. If the neighbors have the completion of the partially seen object, the completed object is placed on the new target list. The new target list is then matched to the various lists of existing tracks as described in Section 7.3.5, Merge and Scan-to-Scan Association. Finally, tracks that are updated in the present scan are made available to the next processing stage, namely sensor fusion.

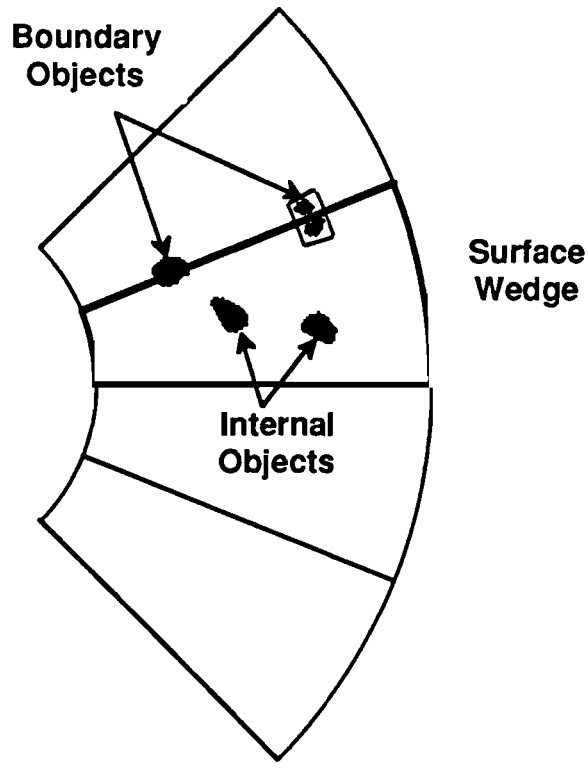


Figure 7-3-3. Wedge boundary objects.

7.3.2 Radar Interface

While current general purpose computers are available that are capable of supporting the surveillance algorithms, the safety system and assorted displays, no commercial hardware exists for the esoteric task of interfacing to a high-speed imaging radar in real-time. Since designing, simulating, wirewrapping, debugging, and maintaining custom hardware is expensive and time-consuming, the radar interface design has to be flexible. It must support the ASDE-X radar, sampling at 40 megabytes/second, and the ASDE-3 radar, sampling at 42 megabytes/second. It must support real-time radar processing, raw radar data recording, and playback of recorded radar data. Processing and recording the radar data at the full real-time rate would be prohibitively expensive- and it would mostly process samples of no interest, where airport surface traffic cannot possibly travel. Therefore, the radar interface must map in only the areas of interest and ignore all other areas. To meet all these needs, the radar interface consists of three modules: the Radar Interface Board (RIB), which does the analog to digital conversion, the Map board, which filters the data in range and azimuth, and a Mercury 3200 single board computer (MC3200), which formats and outputs the resultant data.

The Map board and MC3200 are placed on a VME bus where formatted radar samples are copied into the VME memory of the system that processes (or records) the data. The RSLS real-time ASDE surveillance system is configured with the Map and MC3200 boards placed in the VME side of a Silicon Graphics Incorporated (SGI) UNIX multiprocessor. Another application of the radar interface is in a VME computer system called the Lincoln ASDE Recording System (LARS), which records the data onto a high-speed Honeywell Very Large Data Store (VLDS) tape recorder. The LARS can record data and the real-time SGI system can process the same radar data simultaneously and independently. The MC3200 in the LARS runs slightly different software than the MC3200 in the SGI.

The data tapes produced by the LARS can be played back in real-time through the SGI-based surveillance system by directly connecting the MC3200 in the LARS to the MC3200 in the SGI (bypassing the RIB and Map board). The MC3200 in the LARS runs software that mimics a RIB/Map board connected to a radar, so the SGI real-time system processes as if it were really connected to a radar.

The importance of playing back recorded radar data in real-time cannot be exaggerated. System development and debugging are greatly speeded by processing the same data that deterministically reveals the same bugs each time. While keeping the number of algorithm parameters as small as possible was a major design goal, optimizing even a small number of parameters would be extremely difficult without the ability to assess the effects of parameter changes on the same data. Rare system failures, which would otherwise be almost impossible to reproduce, can be captured in the tape recordings and played back to determine the cause.

7.3.2.1 The Radar Interface Board

The RIB has the following three inputs: the radar receiver video-frequency output, the radar's Azimuth Change Pulses (ACP), and the radar's Azimuth Reference Pulses (ARP).

The radar receiver's video output is an analog signal where the amplitude represents the received power for some location on the airport surface. In the case of the ASDE-X, the video is the logarithm of the received power. The exact location on the surface that corresponds to the video can be determined by knowing when the radar transmits a pulse and where the antenna was pointed when the pulse was transmitted. This is the purpose of the ACP and ARP signals.

An ACP pulse is generated whenever the antenna moves past a series of fixed positions. An ARP is generated whenever the antenna is pointed directly "North"- actually this is simply a reference direction and is arbitrarily chosen. After each ACP, the radar transmits a pulse. The number of ACPs since the last ARP are counted and therefore determine the position of the antenna when the pulse was transmitted. The time lag since the pulse was fired determines the distance from the radar of each part of the video.

A related concept to ACP/ARP is the Pulse Repetition Interval (PRI). PRI is technically the time between radar pulses. There are many radar video samples during each PRI. So each sample during the same PRI represents the received power at a different range but at the same azimuth. Therefore, the term PRI is often used to mean the set of samples for a given azimuth that are all generated from a single transmitted pulse.

The RIB does A/D conversion on the incoming video stream using the ACP and ARP pulses for time association. The output is 40 (or 42) megabytes/second of digitized radar data along with clock and control bits.

7.3.2.2 The Map Board

The Map board maintains a censor map of each position on the airport surface. Each bit of the map represents a group of samples whose extent is 4 range gates by 32 azimuths. The azimuth and range gate (the number of samples since the start of the PRI) of each sample is looked up in the map to determine if the sample is to be thrown away or transmitted to the MC3200.

The Map board receives the RIB data and filters it based on range and azimuth as described above. The samples of each PRI are grouped together, and the Map board's internal independent free-running clock is used to time-stamp each PRI. Sixteen PRIs are buffered in memory and then transmitted over a 16-bit high-speed parallel cable to the MC3200.

7.3.2.3 The Mercury 3200 Single Board Computer

The MC3200 has two jobs. It controls the Map board; initializing, status monitoring, downloading the censor map, and receiving the radar information. The MC3200 also groups the radar information into packets and determines where the data is to be sent for further processing. The software for the MC3200 is written entirely in C.

The MC3200 in the SGI communicates with clutter-rejection processes running on any of the SGI CPUs. Each clutter-rejection process operates on a fixed region of the airport surface. This is currently arranged so that, if there are N clutter-rejection processes and M samples/scan, each process will receive

$P=M/N$ samples/scan. A further convention is that the 1st clutter-rejection process always receives samples 0 through P-1, the 2nd clutter-rejection process always receives samples P through $2P$-1, and so on. The division is performed so that PRIs are not split between clutter-rejection processes, but the concept remains the same.

The MC3200 determines which clutter-rejection process is to receive each packet of radar data based on the azimuth of the data. After inserting a small packet header, the MC3200 performs a VME block move of the packet into the ring-buffer associated with the appropriate clutter-rejection process. Each packet of radar data contains the samples (censored by the Map board) for 16 PRIs, the number of samples, the azimuths of the PRIs, the clock time, and a sequence number incremented with each transmitted datum.

In addition to the PRI packets, the MC3200 also communicates other information to the clutter-rejection processes. Since the exact range of the samples cannot be decoded without knowing which samples have been censored, the censor map is transmitted to each clutter-rejection process during system initialization. The time of day associated with the clock counter value of zero is also transmitted to each clutter-rejection process.

After the last PRI packet for the current scan has been sent to any clutter-rejection process, an End of Scan record is sent. This allows the clutter-rejection process to immediately realize that no further data is forthcoming for this scan and the remainder of the scan can be spent calculating new clutter statistics and performing other bookkeeping functions. The clutter-rejection process could monitor the azimuth of each PRI packet to determine this, but in the unlikely event of an error, the end of scan could not be determined until data from the next scan arrived.

During playback, when the data source is not a radar but a tape in the LARS, data corruption due to tape errors is possible because of the very large volume of data. When this occurs, an error record is sent to each clutter rejection process that would normally receive the corresponding data. Since the software for the MC3200 in the SGI is identical during playback and real-time processing, the software for error records is enabled, but never activated during real time.

7.3.3 Clutter Rejection

The clutter rejection processes have three major goals. The first goal is to eliminate as much receiver noise, radar returns from unimportant objects, interference from other radars, and weather interference as possible. The second objective is to detect as many radar returns from aircraft and other ground vehicles as possible. The third goal is to support the real-time system.

There is some conflict between these goals. Detecting as many targets as possible actually means allowing a fair number of clutter returns to be declared as target returns. This way the rest of the system, which has history and target shape information can prune out these false detections without increasing the overall system False Alarm Rate (FAR) and maintain an extremely high Probability of Detection (Pd). But the real-time needs of the rest of the system may necessitate reducing the number of target detections to lower the data rate. A balance must be found. The processing power of CPUs commercially available in the next few years may make tradeoffs between real-time behavior and algorithm performance unnecessary, but the current system must take this into consideration.

As the first goal implies, clutter is composed of the radar returns from objects on the airport surface which are not aircraft and ground vehicles (such as buildings, lights, signs, grass, etc.), received pulses from other radars, and the backscatter from rain, snow and other adverse weather. This implies that the only correct output from the clutter rejection algorithm is runs of samples caused by aircraft, ground vehicles, and multipath. Since multipath does not have the same statistics as the background clutter, it is not appropriate for the clutter rejection algorithm to reject the multipath. The multipath will be eliminated in further stages of the system.

7.3.3.1 Input Data

The clutter rejection processes have three input sources; the command line arguments, the runway/taxiway map (a run-length table), and the MC3200. The command line arguments and the runway/taxiway map will be discussed in the Initialization section (Section 7.3.3.6.3). The data from the MC3200 is also discussed in the Radar Interface section (section 7.3.2).

7.3.3.1.1 Ring Buffers

The MC3200 communicates with the SGI processors through a shared memory circular buffer (a ring buffer). The MC3200 is on the VME bus inside the SGI rack. The ring buffer memory is actually on the SGI proprietary high-speed bus, but is mapped onto the VME bus by a SGI I/O board. Unfortunately, there is no way for the MC3200, which is executing a stand-alone kernel, to use Unix System V shared memory segments and semaphores to communicate to the SGI processors.

The memory size required for the ring buffers is calculated and the Unix kernel is configured to reserve this memory. A program (called Xmc3200) is executed on the SGI which determines the size of the reserved memory and downloads the MC3200 program. The operator of the real-time system enters the number of ring buffers which are to be used. Xmc3200 sends the number, sizes and addresses of the

ring buffers to the MC3200. Each clutter rejection process is started with an address of a ring buffer as a command line argument.

Since the ring buffers occupy memory outside the control of the SGI Unix kernel, operating system services such as semaphores are not supported. A complete semaphore package could be implemented using VME test-and-set instructions. Fortunately, since only one process writes to the ring buffer, and only one process reads the ring buffer, a rigorous semaphore is not needed for this application. A pair of pointers is sufficient. One pointer points to the byte past the last record in the ring buffer. The other pointer points to the first record in the ring buffer. The MC3200 determines if space is available in the ring buffer for the next record, writes the next record, then updates the last record pointer. The clutter rejection process determines if there are any records in the ring buffer, reads the record, then updates the first record pointer. There cannot be any contention since the instructions to update the pointers are atomic.

7.3.3.1.2 MC3200 Records

The first thing the MC3200 sends each clutter rejection process is the basic constants that each process needs in order to properly interpret the radar samples. The MC3200 first sends the clutter rejection process the azimuth numbers of the PRIs which will be sent, the total number of clutter rejection processes N, and this process's sequence [1..N-1].

Next, the MC3200 sends a record which contains the basic radar constants such as the PRF, the sampling frequency, the scan time, etc. This record also contains the censor map. The censor map contains a bitmap of the airport surface. Each bit corresponds to a cell of 4 samples in range and 32 samples in azimuth. When the bit is on, the samples in this cell will be included in the radar data, when the bit is off the corresponding samples will not pass the RIB (see section 7.3.2).

The final initialization record is a clock record. The RIB contains a real-time clock (see section 7.3.2) called the radar clock. When a new ACP is received, the radar clock value is sent, along with the azimuth number, and record sequence number in the header for each PRI. The clock record contains the SGI system clock value which corresponds to the zero radar clock value. This time record is needed in order to convert the radar clock count into a time and date.

The MC3200 groups 16 PRIs into one PRI packet. Each PRI packet has it's own header, in addition to the header the RIB attaches to each PRI. An error condition is indicated if the radar clock value for the first PRI in the PRI packet is zero. In the event of an error, the data for the PRI packet is ignored.

When all of the PRI packets for the current scan have been sent to the clutter rejection process, the MC3200 sends an End of Scan record.

When the system is operating in playback mode, receiving radar data from the LARS instead of the radar, it is possible (in fact quite likely) that there will be some tape errors during the playback. The LARS will send an error record containing the number of packets lost. The MC3200 sends an error record to each clutter rejection process which is affected. Then the MC3200 sends empty PRI packets for each

packet lost. This error record is only possible during playback and cannot be sent during normal real-time operation.

7.3.3.2 Processing Steps

Each clutter rejection process has two phases. First, the target detections are made and transmitted as soon as possible to reduce the latency. The latency requirements are discussed in more detail in section 7.3.3.4.1. Then , after all the radar samples for this process for the current scan have been thresholded, the clutter map is updated.

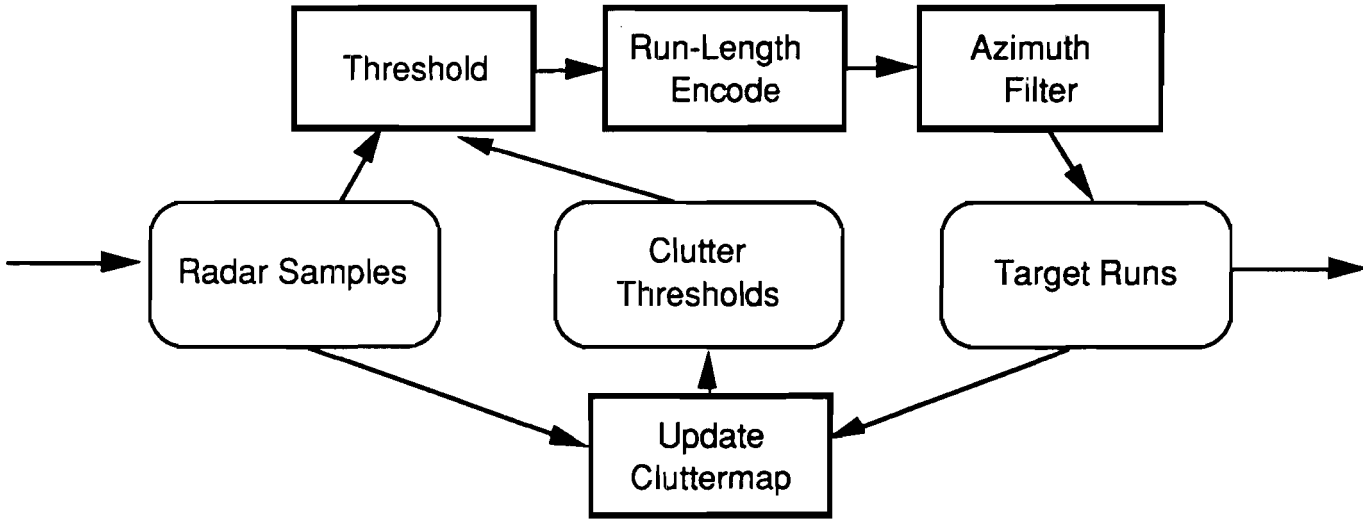


Figure 7-3-4. Processing and data flow of each clutter rejection process.

It is important that targets that do not move for long periods of time continue to be detected and tracked. Therefore, the estimation of clutter statistics should not include target samples in the calculation of the clutter thresholds. The internal feedback loop in Figure 7-3-4 is designed to avoid this. The thresholds are updated for each sample except for those samples that have been declared as target samples.

It is easy to see that if the algorithm functions perfectly that only the samples which correspond to surface clutter will be updated each scan. But, of course, the algorithm cannot be perfect. So how do errors affect system performance? The algorithm parameters are intentionally set so that there is a very high Probability of Detection, $P_d$at the expense of the False Alarm Rate, FAR. In the event of a false detection, the sample erroneously declared to be a target pixel does not update the clutter statistics. Since

the false detections are independently distributed over the airport surface each scan, it is extremely unlikely that a given clutter sample will be declared to be target for several consecutive scans. Therefore, the clutter statistics will not be significantly affected by false detections.

Another possible event is a missed detection. This occurs when a target sample is erroneously included in the clutter statistics. Then the sample from the target distribution will be included in the clutter statistics estimation and bias the clutter thresholds. This frequent occurrence is illustrated in Figure 7-3-5. When an aircraft moves onto a location, the pulse shape, and antenna horizontal beamwidth cause the smooth sample value vs. time in Figure 7-3-5. There is an interval where the reflected power from the target is not significantly above the clutter as to be detectable. In this interval the clutter thresholds can be seen to be effected.

Knowing the scan rate, the radar characteristics, the algorithm parameters, and the target RCS, it would be theoretically possible to operate any ground vehicle a "stealth" target which would never be detected. It could be driven at such a rate so that the clutter statistics rise so high that the target is always below the threshold. Fortunately, this pathological case is practically impossible. In practice, the thresholds are raised slightly as the target moves into the area, but well below the peak target values so that the target is always detectable. This can be seen in Figure 7-3-5.

One lesson to be learned by Figure 7-3-5 is that, in addition to lowering P_d, a large value for K will increase the effects of missed detections on the clutter statistics. The target sample which is above the mean will increase the standard deviation, and this error is then multiplied by K to determine the new threshold.

If samples from radar interference are included in the clutter estimation, it will not significantly effect the clutter statistics since like random false detections, radar interference will not appear in the same place consistently enough to significantly bias the clutter statistics. For this to occur, the PRF and scan rate of the interference source would have to be synchronized with the radar. This is not impossible; just extremely unlikely.

Earlier versions of the clutter rejection algorithm did not have the internal feedback loop in Figure 7-3-4. Instead, an external feedback loop was used. While the clutter rejection process only performs point target detections, the rest of the system has spatial and temporal information which makes target detections from the overall system much more reliable than the point detections from the clutter rejection processes. This is a much more desirable source for the target pixel feedback used to train the clutter map.

The reason that external feedback was abandoned was not due to algorithm performance issues, but real-time implementation issues. The external feedback loop was an additional hard real-time constraint on the system. The target information was required before the clutter statistics could be estimated for the next scan. The timing of the different modules in the surveillance system would need to be much more precise to avoid excessive latency. Since the internal feedback mechanism was found to be satisfactory, external feedback was gladly eliminated.

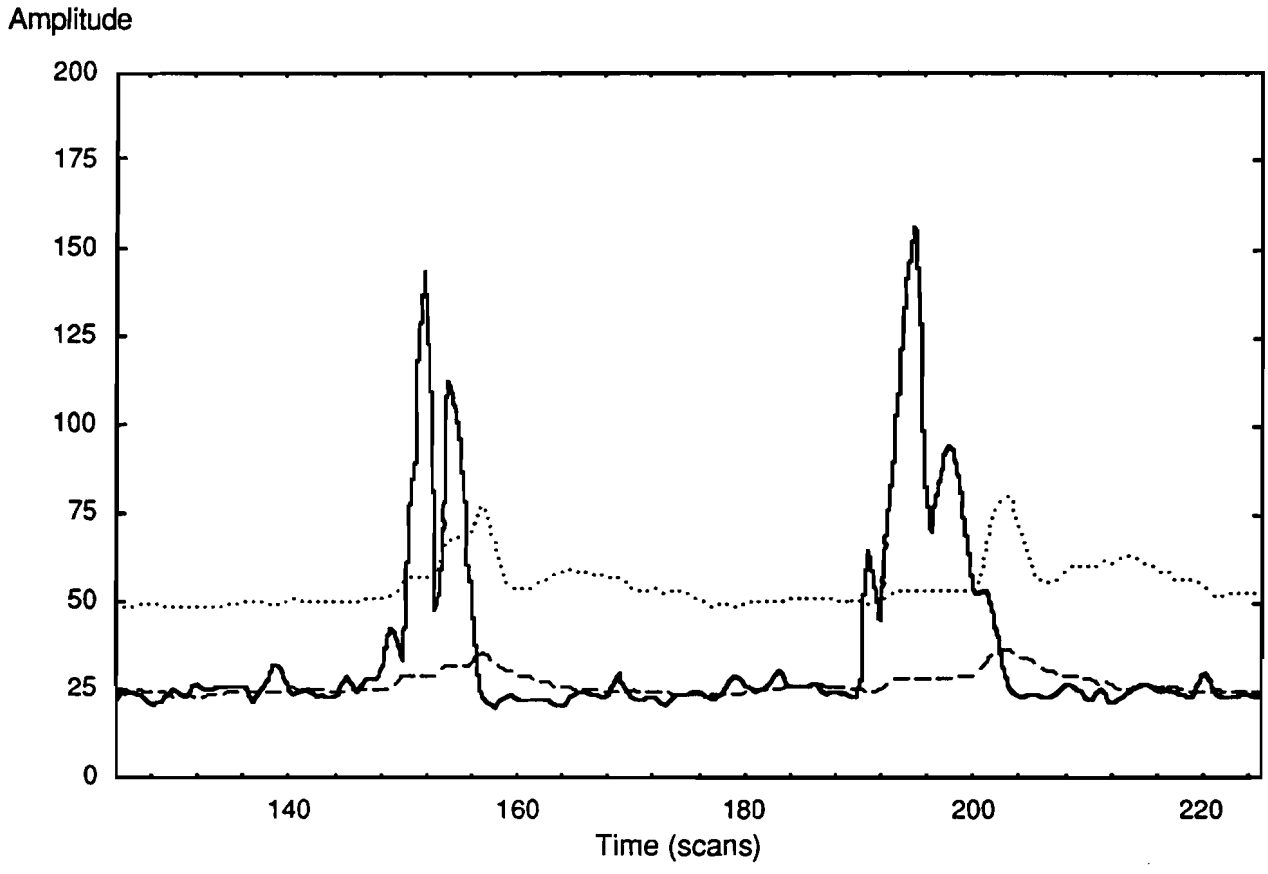


Figure 7-3-5. Clutter elimination thresholding. The solid line is a plot of a sample on a runway. The dashed line is the mean estimator. The dotted line is the threshold. The threshold is constant while samples are above the threshold, but when the target samples are below the threshold the statistics are effected.

7.3.3.3 Output Data

The output data from each clutter rejection process can be grouped into two types, initialization information, and target detections. The initialization records are sent each time a client connects. If a client disconnects, then reconnects, the initialization records are sent before any target runs are sent.

Radar constants (e.g., PRF, scan time, sampling frequency) are bundled with the censor map. Since the censor map bitmap used by the RIB hardware is difficult to manipulate, the bitmap is converted to a run-length table (RLT). Each run in the censor map RLT consists of the starting sample number and the ending sample number of the samples whose bits are on in the bitmap.

The time record which contains the time of day which corresponds to the zero of the radar clock is sent. This record is identical to the record from the MC3200.

To minimize boundary effects, the target runs from several PRI packets are grouped together into a larger wedge. After all the runs from each PRI in the wedge have been collected, the entire wedge is sent.

After all the target detections for the current scan for this process have been sent, and End of Scan record is sent.

7.3.3.4 Real Time Constraints

There are three major considerations for real-time processing; peak processing load, average processing load and latency. The radar data flows from the MC3200 into the clutter rejection processes in uneven bursts. At Logan Airport, during the portion of the scan where the radar faces away from the airport the data is ignored. Even during the portion of the scan when the radar faces the runways the censor map causes uneven data rates. Some PRI's will have hundreds of samples which must be processed, other PRI's have very few, but since the radar sampling rate is a constant 40 MB/s, the data rate fluctuates. The solution to this peak data rate problem is simple. Sufficient memory is allocated to buffer the data so that the peaks and lulls in the data rate are averaged together.

No matter how much buffering is performed, however, if the processing is not capable of handling the average data rate, real-time performance cannot be sustained. The sampling rate of the radar, 40 MB/s for the X-Band radar, is much too great to process in real-time. The RIB applies a censor map to the data which reduces the data rate to a much more manageable rate of approximately 600 KB/s. But even this rate is too great for the clutter rejection software running on one MIPS R3000. To achieve real-time rates with the current generation processors, the clutter rejection software runs in parallel on several processors.

The total surveillance system must deliver the aircraft tracks in a timely manner to be useful. The latency of the target tracks is defined as the elapsed time between the time the radar scanned the target to the time the target reports are transmitted. Even though both the clutter rejection algorithm and the connected components algorithm operate in parallel processes, each packet of radar data must flows serially through a clutter rejection process and then into a connected components processes, and then the merge process.

7.3.3.4.1 Latency

The latency of each clutter rejection process is reduced by a very straightforward ordering of the algorithm steps. As can be seen in Figure 7-3-4, each clutter rejection process first compares the input samples to the corresponding clutter threshold in the clutter map and run-length encodes the samples which are above the threshold. Next process applies the azimuth filter, and these filtered detected runs are then transmitted to the connected components algorithm. Only after every sample has been transmitted (for this process) does the process start to train the clutter map.

7.3.3.4.2 Parallel Processing

There are several schemes which could be used to split the clutter rejection into separate processes. For maximum flexibility and ease of implementation, a loosely coupled scheme with local memory was chosen. It is likely that a tightly coupled scheme with only global memory would work quite well and could possibly take better advantage of the SGI system resources. But a tightly coupled approach would restrict the hardware that could support the software. This tightly coupled approach will be discussed, but has not been implemented.

The simplest approach to parallel processing was taken. The airport surface is split into N wedges (these wedges should not be confused with the wedge definitions in the connected components processes). Each clutter rejection process is responsible for only the samples in it's wedge. Each process thresholds and run-length encodes the samples for it's wedge, applies the azimuth filtering, then calculates the clutter thresholds for the next scan in the remaining time available.

While the current parallel architecture was easy to implement, it is not optimal. In the beginning of each scan, the first clutter rejection process receives data faster than it can process, yet all the other clutter rejection processes are idle. The first clutter rejection process will also finish training the cluttermap and become idle while other clutter rejection processes still have work to do for the current scan. Both the average processing rate and the latency can be improved by taking better advantage of parallel processes.

The architecture of the SGI would also support a tightly coupled approach. The clutter map (the clutter thresholds and the statistics needed to calculate the thresholds for next scan) for the entire scan would reside in a global shared memory segment. Each process would have the same basic control structure; thresholding and run-length encoding, apply the azimuth filtering, transmit the runs, then calculate new clutter thresholds. But rather than wait for a large portion of the airport samples to be thresholded, each process would operate on just one packet.

The only critical resource which would have to be protected with a semaphore is the ring buffer which receives the data from the MC3200. A clutter rejection process would first grab the semaphore (blocking if it is unavailable), then copy the PRI packet into local memory, update the ring buffer's pointer which points to the next packet to be read, then release the semaphore. The azimuth range of the PRI packet determines the portion of the shared clutter map in shared memory which will be accessed and modified. This section of the clutter map is unique to this scan, so as long as each clutter rejection process is operating on the current scan, the clutter map shared memory segment does not need to be protected with a semaphore.

For airport topologies like Logan Airport where the radar points away from the airport for a large fraction of the scan, this is probably sufficient because there is a large percentage of a scan where no new data is available to be processed. If it is possible that there could be contention for the same area of the clutter map by two processes one process operating on the previous scan, one process operating on the current scan, then each portion of the clutter map will need to be protected by a semaphore. This presents two difficulties. Many computer systems may not be able to support this large number of semaphores [the SGI can] which will further restrict the portability of the software. The second problem is that this scan overlap implies that there are times when the rate of data processing is less than the required real-time

rate. If this scan overlap occurs often, then the processing load is greater than the average processing cababilities and the real-time system cannot be stable.

The previous parallel implementations of the clutter rejection algorithm are both Single Instruction Multiple Data (SIMD) processes. The code for each process was identical , but the data which each process received was different. Another tightly coupled parallel implementation of the clutter rejection could be performed by splitting the clutter rejection algorithm into two different parts. This is called Multiple Instruction Multiple Data (MIMD) parallelism.

A MIMD approach could be implemented by having one type of process which thresholds the data, and another type of process which trains the clutter map. The data would flow from the MC3200 into a shared memory segment to be read by both the threshold and clutter map training processes. Each threshold process would use the global clutter map scheme previously described. The target runs would be transmitted into queues for both the connected components processes and the clutter map training processes. A clutter map training process would grab the oldest packet of target runs from the queue and lock the appropriate section of the clutter map, and locate the radar samples which correspond to the target packets. Then the new clutter thresholds would be calculated and the lock on the clutter map segment released.

7.3.3.5 Target Detection

As previously discussed, clutter rejection is performed in two phases. The first phase performs the target detection, and the second phase updates the clutter map. As shown in Figure 7-3-4, the target detection phase is composed of three parts: thresholding, run-length encoding, and azimuth filtering. The thresholding and run-length encoding are performed together.

7.3.3.5.1 Thresholding and Run-Length Encoding

Each 8-bit sample is compared to the threshold in the clutter map for the corresponding azimuth and range. The output is this binary decision. Either the sample is above the threshold or it is not. Rather than send a 1 or 0 for each sample, the samples above threshold (the target samples) are run-length encoded. A run of target samples consists of consecutive target samples and is represented by the sample number of the first sample in the run, and the sample number immediately following the last sample in the run.

There are actually two clutter map thresholds: a Short Term Memory (STM) threshold, and a Long Term Memory (LTM) threshold. The STM performs the bulk of the clutter rejection while the LTM exists only for the special case caused by target shadows (see Section 7.3.3.6). Each sample is first compared to the STM threshold and if it is greater compared to the LTM threshold. Since the vast majority of the samples are clutter, most samples are tested only to the STM threshold (and fail). Only a relatively small minority are then compared to the LTM threshold.

Samples are numbered by considering the censor map. The 1st sample in the censor map is sample 0, the second is sample 1, and the last sample is sample N. Since it would be too laborious to run through the censor map to determine each target sample's sample number, a table lookup is used.

After the censor map is read during initialization, the sample number for the first sample on each PRI is calculated and stored in a lookup table. The sample number for sample s on PRI p is simply $lookup[p]+s$.

The convention for runs used throughout the surveillance system is that the starting sample number is the sample number of the first sample in the run. The ending sample number is 1 plus the sample number of the last sample in the run. The length of the run is therefore, end - start. This is an arbitrary convention used in many image processing systems.

Target runs never overlap PRIs. In other words, if a target run extends to the last sample on PRI n, and another run starts on the first sample on PRI $n+1$, the two runs will not be consolidated into one run.

7.3.3.5.2 Azimuth Filtering

The clutter rejection algorithm is a CFAR algorithm. In other words, there is a finite positive probability that a clutter sample will be above the threshold. To raise the probability of detection, the number of false detections may be in the order of one in a hundred. This will not raise the overall system false alarm rate since these random false detections will be spatially isolated and easily removed by further processing.

But these false detections can cause problems for the clutter map training. When the clutter statistics are estimated, no samples above the threshold will ever be considered as part of the clutter distribution. But some of the samples in the clutter distribution will be above the threshold. This would bias the estimation of the clutter statistics and lower the clutter threshold.

One solution would be to calculate the bias and add it to the threshold. But since these false alarms add extra load to the rest of the system and will be eliminated anyway, a simple spatial filter is used to eliminate the salt and pepper false detections before the target runs are transmitted and used for training the clutter map.

Any target run which is not adjacent to a run in the preceding PRI or the next PRI is eliminated. Runs on the first or last PRI in a packet are never eliminated. Since the target runs from several PRI packets are grouped together for transmission, the azimuth filter would have better performance if the entire wedge was filtered. The current software, however, does not do this. Each packet of 16 PRIs are filtered separately.

7.3.3.6 Updating the Clutter Map

The clutter map is composed of two parts: a Short Term Memory (STM), and a Long Term Memory (LTM). The basic workhorse of the clutter rejection algorithm is the STM. The STM holds the clutter thresholds which rapidly adapt, on the order of several scans, to the current environment.

The only reason for the LTM is to handle shadows. A large target, such as an aircraft, will cast a shadow. The samples in the shadow of an aircraft will be lower than the clutter threshold and therefore bias the clutter statistics. When an aircraft stops for longer than the STM time constant, the clutter

statistics in the shadow may go down to the level of receiver noise. If there was a significant amount of clutter in the shadowed area, when the aircraft departed the clutter could be above threshold and declared to be target samples.

All of the clutter estimators are performed with floating point arithmetic. Empirical testing determined that, with the MIPS R3000 CPUs, there was no significant performance improvement when integer arithmetic was performed. The additional accuracy with floating point arithmetic was much more significant.

7.3.3.6.1 Short Term Memory

There are three floating point numbers for each sample in the STM: the STM threshold, the mean estimate, and the average square value estimate. The STM threshold is equal to the mean plus a constant, K, times the standard deviation ($\mu + K\sigma$).

7.3.3.6.1.1 Estimating the Mean

The mean is estimated with a simple recursive filter: $\mu_i = ((N-1)\mu_{i-1} + x)/N$, where μ_i is the mean on the ith scan, x is the current sample value, and N is the time window which controls the decay. The current parameter N is 6.

7.3.3.6.1.2 Estimating the Standard Deviation

The standard deviation, σ, is the square root of the variance. The Markov estimator for the variance is equivalent to xbarsquare - xsquarebar, where xbarsquare is the square of the mean, and xsquarebar is the mean of the squared sample values. The mean has already been estimated and can simply be squared. A recursive estimator for the average square value is used. The estimator has the same form as the mean estimator: $ASV_i = ((N-1))ASV_{i-1} + x*x)/N$, where ASV_i is the average squared value for scan i, x is the sample value for the current scan, and N is the time window for the estimator. Since the ASV estimator is noisier than the mean estimator, the value of N should be larger than the mean estimator time window-- twice as large is a reasonable value.

Rather than calculating the square root of the variance and then multiplying K times the result, a table lookup is used to combine both calculations. Since the standard deviation estimate is much noisier than the mean estimate, an upper and lower bound on the standard deviation can be given. This algorithm parameter is expressed in the product $K\sigma$ and is also combined into the table lookup.

7.3.3.6.2 Long Term Memory

The LTM is calculated by taking the maximum of the current LTM threshold with the current STM threshold. Perhaps a better, but more CPU intensive, update rule would be the maximum value of the current LTM threshold times a decay constant and the current STM threshold. The LTM decay would be trivial to implement, but is not in the current software.

The current software has the LTM threshold disabled. This has not been found to adversely affect the system performance.

7.3.3.6.3 Clutter map Initialization

All the discussions of the clutter rejection algorithm in this section have been about steady state performance where the clutter map is already assumed to contain the correct statistics and thresholds. But how is this state achieved?

There are two parts to the clutter map initialization. The first is convergence to the proper values. The initial clutter map is set so that the STM statistics are very high and the LTM statistics are zero. Therefore everything is assumed to be clutter. After a few time constants, the STM converges to the proper clutter estimates and thresholds-- but still assuming that all stationary objects are clutter. When a target which was stationary during the clutter map convergence period moves, it moves onto an area which has correct clutter statistics and is then detected. After a few time constants, the area where the target was stationary converges also.

The LTM is not updated until the STM has converged.

Since targets may be queued up for considerable periods, the time before the entire airport surface has converged may be excessive. Therefore a map of the runways and taxiways is loaded during initialization. After the STM has been initialized, the areas in this map are reset to values higher than receiver noise (the normal RCS of pavement is lower than the receiver noise), but significantly lower than the peak values for aircraft. This allows the runways and taxiways to converge quickly, while the other more complex areas converge at a slower rate.

7.3.3.7 Desirable Improvements

7.3.3.7.1 Reconnect to MC3200

The clutter rejection software is very robust and has a very long mean time between failure. Nevertheless, it is possible that a clutter could crash. It would not be difficult to create a process which monitors the clutter rejection process and restarts any process which fails. The problem is, the current MC3200 program only send the initialization information (the censor map, and the time record) once.

It would be a nice feature if a clutter rejection process could signal the MC3200 that it needed another copy of the initialization records.

7.3.3.7.2 Additional CFAR for rain

The current system performs quite nicely in low to moderate rain. Higher levels of rain, however, cause some problems. One reason for this could simply be that less sensitive parameters are required. Parameters for heavy rain have not been explored yet.

But it is likely that lowering the sensitivity of the clutter rejection software will not lead to good rain performance. The greatly reduced signal to clutter ratio may lead to low probability of detection with reduced sensitivity. And local rain gusts may cause large false detections (the area CFAR algorithm used in the display of the ASDE-3 can also cause this).

Better results might be found by applying both the current time CFAR with an area CFAR. This could be done without excessive CPU requirements by counting the number of target samples detected in each wedge. If the number of detections becomes unreasonable, a higher threshold could be used until the number of detections was reasonable. The threshold offset could gradually be reduced either with time, or as the scan moves away from that region on the surface (or both). The threshold offset could be performed in a table lookup, requiring no additional processing load beyond the reprocessing of a wedge.

7.3.3.7.3 LTM Decay

As mentioned earlier the LTM delay, while trivial, has not been implemented because the LTM is currently disabled.

7.3.3.7.4 Additional Feedback

Either in addition to the LTM, or instead of the LTM other feedback mechanisms could be implemented. Each of the following mechanisms have the desirable feature of not requiring any additional hard real-time constraint between other parts of the surveillance system and clutter rejection.

7.3.3.7.4.1 Resetting the Clutter map

The output of the surveillance system could be monitored to locate any false detections that clutter rejection consistently declares. This could be done by noting that a target has been declared out range of where a large target was stationary. If the target never moves and has all the characteristics of clutter that was shadowed the location of the false target could be sent to the appropriate clutter rejection process.

The clutter rejection process can reset the STM in that area to high values so that everything in that area will be declared as clutter. Then a few time constants later, the STM will converge to the correct clutter values. If the LTM is used, it should be turned off in this area until the STM has converged.

There is no hard real-time constraint since the clutter rejection process does not absolutely have to reset the STM on the very next scan.

7.3.3.7.4.2 Shadow Detection

It may be possible to detect shadows and not update the clutter statistics there, exactly as clutter statistics are not updated where targets are detected. This could be done analogously to target detection. Targets are declared when the sample is greater than $\mu + K\sigma$. Shadows could be declared when the sample is less than $\mu - K\sigma$.

7.3.3.8 Clutter Rejection Parameters

The following section enumerates clutter rejection algorithm parameters which have been described elsewhere.

7.3.3.8.1 Wedge Size

While the radar samples are received in packets of 16 PRIs per packet. To limit boundary effects, a larger packet size is desired for the output target runs. The -wedge N command line argument controls the number of PRI packets which are processed and sent in one wedge. The current default value for N is 8, which means that the wedge size is 8 * 16 or 128 PRIs per wedge. N must be an integer.

7.3.3.8.2 Mean Estimator Window

The mean estimator window size is controlled by the -t N command line argument. The argument N is the integer number of scans to use as the time window. The current default is 6 scans.

7.3.3.8.3 K

The constant *K* used to calculate the STM clutter thresholds in the formula threshold = mean + K sigma is controlled by the -k X command argument. The floating point argument X is used as the constant. The current default value is 5.

This argument controls the sensitivity of the clutter rejection. This indirectly has strong effects on the real-time performance of the system since a more sensitive detection will result in a greater data rate.

7.3.3.8.4 Runway/Taxiway Map

The RLT used as the map of the runways and taxiways for clutter map initialization is a compile time constant. In RejectClutter.C the constant name is runway_map. The current default value is "runup.img." This will probably be made a command line argument. in the future.

7.3.3.8.5 Runway Initial Clutter Value

The initial STM clutter map value assigned to each sample in the Runway/Taxiway map is a compile time constant in the file RejectClutter.C The constant name is runway_initial_threshold. The current default value is 75.

7.3.3.8.6 Ring buffer Address

The address of the memory to map in to the process address space for the ring buffer which receives data from the MC3200 is controlled by the -buf A command line argument. The argument A is a hexadecimal number. This argument is required and there is no default value.

7.3.3.8.7 Ring buffer Size

The number of bytes in the memory segment for the ring buffer which receives data from the MC3200 is controlled by the -len N command line argument. The argument N is a hexadecimal number. This argument is required and there is no default value.

7.3.3.8.8 Convergence Delay

The clutter rejection process will threshold and train the clutter map for a number of scans before the clutter map has converged. During this period initialization messages and End of Scan records will be sent to all clients which connect, but no target runs will be sent. The number of scans to wait before sending target runs is controlled by the -wait N command line argument. The integer argument N is the number of scans to wait. The current default value is 50 scans.

7.3.3.8.9 Sigma Bounds

Since the standard deviation (sigma) estimation is noisier than the mean estimator, it is desirable to limit the range of sigma. But sigma is not actually calculated, rather the quantity K sigma is returned from a table lookup to achieve better processor efficiency. For this reason, the K sigma product range is controlled by the -min_k_sigma X0 and -max_k_sigma X1 command line arguments. Both arguments X0 and X1 are floating point numbers. The current default value for X0 is 15 and the current default value for X1 is 35.

7.3.4 Connected Components

7.3.4.1 Process Architecture

The architecture of the Connected Component process is outlined in Figure 7-3-6. The basic processing stages are:

1. Unflattening of received data structure and conversion to a Run Length Table.
2. Opening
3. Component Extraction Loop
4. Components to Objects
5. Object Feature Extraction
6. Sorting of Components into "totally internal" and "boundary" categories.
7. Censoring of "totally internal" category based on area
8. Flattening and transmission of "totally internal" and "boundary" lists.

When the list of runs is received from Clutter Rejection, it is in the form in which it was extracted. For each radial, a list exists of target to background and background to target transitions. No transition is recorded if a run extends to a region mapped out by the censoring map. In addition, a list of pointers is maintained to the start of the data for each radial as well as the number of "runs" for each radial.

This structure is converted to the representation used for Connected Components processing and for the Merge process as well. This representation actually consists of two "sub-representations" which share a number of characteristics but differ in some respects. In both representations, runs are described relative to an enclosing "wedge". A region of interest is defined from some start radial to some end radial and from start range to some end range. We define an origin as the point of minimum azimuth and minimum range.

In the first representation, every run is defined by two numbers. Each is the number of a point with the count beginning at the origin. The first point corresponds to the location of the first target point after a background point and the second point corresponds to the first background point after the last target point in a run. No information is recorded for the ends of radials so that if a target run extends to the far range limit and another (unrelated) target run starts at the near range limit of the next azimuth, these runs will be represented by a single run starting in one radial and ending in the next. This is the representation into which the data is initially transformed. It is most efficient for operations such as r.l.t. logical operations (see below). This representation will be refered to subsequently as an *integer r.l.t.*

In the second representation, every run is defined by four numbers. For the two points defining the run, the row and column value of the start and end of run are given. In this representation, a point is always inserted for the end of a radial if a run extends to the far range end and for the start of run if a run starts at the near range. This representation is most useful for the morphological operations such as openings and closings and contour tracking where it is important to know the neighbor relation of nearby runs. This representation will be refered to subsequently as a *coordinate r.l.t.*

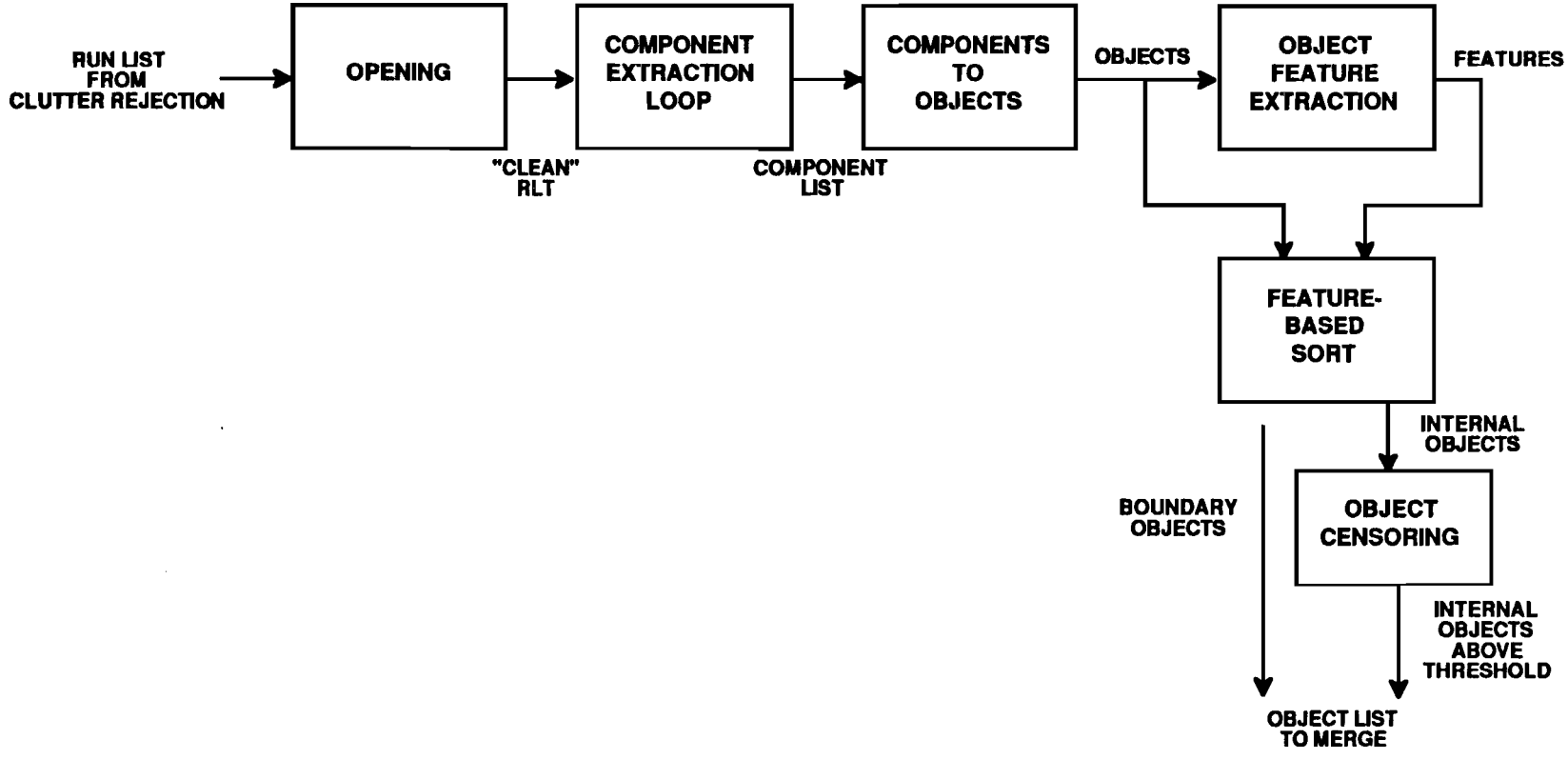


Figure 7-3-6. Architecture of connected component process.

The two representations are easily convertible, and the conversion operation is relatively inexpensive computationally. The advantage gained by having a particular operation performed in a more efficient representation is usually worth the slight expense of a conversion between representations.

After conversion, the received data is in the integer r.l.t. form. It is then converted to a coordinate r.l.t. for the opening. The opening consists of an equal number of erosions and dilations. It has been found that in the presence of increased numbers of small components, typically non-target in nature, the processing time for Connected Components rises rapidly. By performing multiple erosions and their corresponding multiple dilations, considerable real-time savings occur at the expense of missing smaller real objects. Thus a threshold number of runs is used (currently 200) below which one erosion followed by one dilation take place and above which two erosions followed by two dilations takes place.

Further, while an opening on the entire surface run length table would be quite possible, the Connected Component process has only a single surface wedge available to it at one time. It would be possible to produce a functionally equivalent r.l.t. if neighboring wedges had overlapping data. However, for code simplicity, an approximation to a "correct" opening was used at wedge boundaries. Conceptually, one assumes that both neighboring wedges consist of all target. This is a conservative assumption, leading to the removal of the fewest number of pixels at the boundary. It is achieved in a computationally efficient manner by adding several runs at the start and end of the wedge that extend from the minimum to the maximum range. The openings are then performed, and any runs outside the wedge that remain are removed.

The erosions are implemented as follows: each run is first decremented by one at either end. We then make a new r.l.t. with each of the runs in the original r.l.t. replicated three times - one with the previous row number, one with the current row number, and one with the subsequent row number, with all the runs merged into ascending order. Then, a pass is made through the resulting r.l.t. using a counter that increments at every start-of-run and decrements at every end-of-run. A final run is only reported if the counter goes up to three and as soon as the counter falls below three, it is terminated. A dilation is similar, with a run beginning whenever the counter goes above zero and ending when it returns to zero and, of course, lengthening the run rather than shortening. It should be noted that the coordinate r.l.t. representation of this operation is more efficient since row information is explicit, and even more important, since it represents all end of radials explicitly.

The main workhorse of the Connected Components process is the component extraction loop. This loop can be viewed as a recursive process. Its input consists of a "master" r.l.t. containing multiple components. Its output consists of the r.l.t. of the first component in the master r.l.t. and the rest of the master minus the one component. This process is repeated until the master r.l.t. is empty.

Each iteration of the loop is performed as follows: first, the first run in the master r.l.t. is found. Then, a technique called *contour tracking* is applied. This consists of walking around the component in a counter clockwise direction from the first run and recording all 8-connected points visited. The resulting list is referred to as a *chain code.* The chain code is then converted into an r.l.t., which is the first output product. This r.l.t. is then subtracted from the master. Run length table subtraction is one of a class of Boolean r.l.t. operations that include r.l.t *anding, oring,* and *not.* In fact, the subtraction is implemented using the anding of the master r.l.t. and the not of the component r.l.t. The operation is performed in the integer r.l.t. format.

The component extraction loop results in a list of components. Note the implicit data reduction from number of runs to number of "eight-connected items." However, it is unfortunately not true that

each such component corresponds to a real-world *object*. This is most frequently because of aircraft breakup, but can be due to objects being close enough to merge into single eight-connected components. In an attempt to handle the breakup case, a nearest distance threshold is established, and components that are closer than the threshold are put together on a single object list. The most accurate method of performing this check would be a pairwise chain code point-by-point comparison. This is computationally much too expensive. Therefore, for each component an enclosing wedge is produced with some padding added. A computationally inexpensive check for pairwise wedge overlap is performed. If two wedges overlap, the two components involved are dilated. If a single object results, the components (the before dilation versions) are placed on the same object list. Unfortunately, since the dilations are not symmetric in range and cross-range, this test suffers from rotational asymmetry. Computationally inexpensive, better approximations are available, but they have not yet been implemented because of time considerations.

The previous stage results in a list of objects, each composed of one or more components. Then, features are calculated for each object. So far, the features implemented include object centroid, area, enclosing wedge, and perimeter. The perimeter is not currently calculated in the real-time system because of speed considerations.

The next step involves deciding which objects are totally internal to the *surface wedge* for the current packet. The surface wedge is the start azimuth to end azimuth region associated with the current packet. We associate an enclosing *object wedge* with each target detected. The object wedge consists of the first and last azimuths of the wedge as well as the start and end range of the target, plus some padding. If the object wedge is totally contained within the surface wedge, the object is considered to be totally internal to the wedge. If some part of the object wedge is outside the surface wedge, the object is considered to be a boundary object. Using this definition, the objects are then sorted into a list of internal objects and a list of boundary objects. To decrease the processing load for the subsequent Merge and scan-to-scan association processing, a small area threshold is imposed, currently 40 square meters. Objects below this threshold are discarded.

Finally, the lists of internal and boundary objects, where the internal objects include features, are serialized for transmission (flattened). As explained above, all pointers are removed, and the various data fields are placed in a continuous buffer along with sufficient information to reconstruct the various lists. This buffer is transmitted, along with the time stamp of the wedge, to the Merge process.

7.3.4.2 Output Data Description

The output data, as described above, is a list of internal targets with features and a list of boundary targets. These have been "flattened" into a buffer for transmission via the server/client mechanism.

7.3.4.3 Databases

The adjustable parameters maintained in the database for this process are minimal. They include the radar parameters (range bin size, range and azimuth offset, number of radials, etc.). In addition, the database contains the threshold area and the components to objects association distance.

7.3.4.4 Significant Problems/Solutions

The structure of the Connected Process and the techniques used are intended to address a number of surveillance and real-time issues. A list of the most salient issues (in no particular order) follows:

1. Achieving a high degree of parallelism
2. Salt and pepper type small objects cause processing time to climb faster than linear and confuse components to objects
3. Target breakup
4. Handling wedge boundary conditions
5. Extra load on the Merge process from "small" objects
6. Extensible feature set

Given the real-time requirements of this system, which include the ability to apply maximal processing power both when the overall data load went up and when the data load for a few geometrically localized packets goes up, a reasonable scheduling algorithm was required. Such an algorithm would also need to take latency considerations into account. If we allocate processes on a geographical basis (one process per region) it would potentially lead to a single overloaded process handling many targets while other processes on other processors were idle. Since the regions of high load cannot be know a priori, we decided that all processes be geographically independent. Thus, they typically could not have information about neighboring wedges, or history for that matter, since an individual process will have information from different regions for each of its iterations. Thus, the Connected Components processes have no memory of the previous wedges processed. Therefore, when processes go to retrieve new wedges, all processes are equivalent, so any one of them can be chosen. To reduce latency, the oldest available wedge is chosen.

The openings were instituted to address problem number 2. Since the time spent in the contour tracker is proportional to the perimeter of a component, a hefty penalty is extracted for processing small objects with high perimeter to area ratios. In addition, since the beamwidth corresponds to five cross-range samples and the pulse corresponds to 2-3 range pixels, anything larger than a point target, provided that is reasonably bright, is not lost in a single erosion. Finally, the components-to-objects code can be significantly confused in the presence of a great deal of multi-path clutter.

The problem of object breakup (problem 3) is handled by associating components together into objects. The assumption that components in close physical proximity are most likely part of the same object is born out most of the time. In regions such as runup pads and crowded taxiways aircraft can be so closely spaced that this assumption is broken. This can be a significant problem.

The fourth problem, the one of handling objects at boundaries between wedges, is really an offshoot of the region-independent Connected Component processing. Since each Connected Component process has no history, it can have no knowledge of its neighboring wedges. Therefore, the ability to patch together boundary components must be left for down stream.

The small object filter addresses the fifth problem. Its purpose is merely to reduce the load on the Merge process. Since the Merge process is not parallelized, this is one means to adjust the load.

Finally, an attempt has been made to produce a standard template for object features so that new features can be easily added and tested without disrupting the remainder of the code.

7.3.4.5 Suggested Improvements

A number of improvements in the Connected Components system would improve system performance. Some were known at development time but were not implemented in order to produce a working system quickly. Others became apparent upon studying system output. These improvements include the following:

1) Wedges should overlap for improved opening quality
2) R.L.T should be rescaled to have square pixels in components to objects
3) Several binary object features should be added
4) Greyscale data should be passed for use in calculating greyscale features.

As discussed above, the assumption is made in the opening that the neighboring wedges consist of pure target. When objects on boundaries are reconstructed, this leads to distortions in shape. If surface wedges overlap by a few radials, the resulting openings will look exactly as if complete data was always available. This would constitute improvement 1.

Improvement number 2 would better the results of the components-to-objects process. In initial offline forms of the code, the closest distance between components was calculated from point-by-point comparison of the points along their chain codes. While highly accurate, this is remarkably inefficient. This method is approximated by checking if the enclosing wedges with padding from two components overlap. If they do, a number of dilations are performed on the two components, and the number of components in the result is checked. The number of dilations depend on a distance parameter. The number of dilations is the distance parameter divided by the range cell size (this is rounded of course). The difficulty is that the amount of dilation in the cross-range direction is range dependent. This can be corrected by rescaling the r.l.t.'s involved so that the pixels are roughly square and then performing the dilations.

Currently, the perimeter feature has been written and tested but has been removed from the real-time system for speed. This feature should be added to the real-time system as well as various shape descriptors, major and minor axis finders, and several other features. These would aid in bad drop reaquisition (see *Merging and Scan-to-Scan Association*) as well as several other goals such as heading determination. This could be achieved by more code optimization (difficult) or by purchase of faster hardware.

Finally, if the greyscale values were available for target pixels (those pixels that are above threshold in Clutter Rejection), features relating to greyscale could be calculated. These include centroids at different thresholds, peak values, etc. This information would be helpful in multipath rejection and inclement weather handling.

7.3.5 Merging and Scan-to-Scan Association

7.3.5.1 Process Architecture

This executable module actually combines two separate processing stages. These two stages are the last in the Radar Surveillance sequence and must be performed serially. Rather than incur the overhead of "flattening" and "unflattening" involved in implementing them as separate processes, it was decided to have them co-reside in a single process. The first stage involves merging the parts of surface wedge boundary targets and adding them to a list also containing targets internal to wedges. This list constitutes the new targets for the current wedge that will be made available for tracking. The next stage is the scan to scan association, which produces tracks. Each stage will be discussed separately.

7.3.5.1.1 Merge

Figure 7-3-7 outlines the architecture of the Merge stage. The basic processing stages are as follows:

1) Receive and unflatten lists for Connected Components
2) Decide to process or enqueue
3) Register wedge number with wedge list and calculate maximal enclosing wedge
4) Add objects internal to wedge to new target list
5) Attempt to match new boundary components to boundary components currently on unfinished list
6) Perform the Components-to-Objects procedure on targets finished in (V)
7)Calculate features for objects found in VI
8)Add completed objects to new object list.

As described above, each Connected Components process outputs two lists. One contains the objects totally contained within the surface wedge of the packet and the other contains boundary components. In addition, the surface wedge as well as the time stamp for the wedge are sent. In step I, the Merge process chooses the oldest packet from all the Connected Component queues. This packet is unflattened and made ready for processing. In step II, we determine whether this packet is from the "current" scan or from the next scan (step III discusses incrementing scan numbers). If the packet is from the current scan and has not been seen before, processing continues. If it is from the next scan, the packet is enqueued awaiting completion of the current scan, and the next wedge is retrieved. If it is from the current scan and has been seen before, an error is declared. Next, the wedge number is entered in the wedge list (step III). If this wedge completes the current scan, the wedge list is cleared. Further, if this is the first wedge of a new scan, it is used to project a time boundary for the current scan. By splicing together all adjacent wedges seen earlier in sequence number and later in sequence number without gaps, the algorithm calculates a maximal enclosing wedge for the current surface wedge.

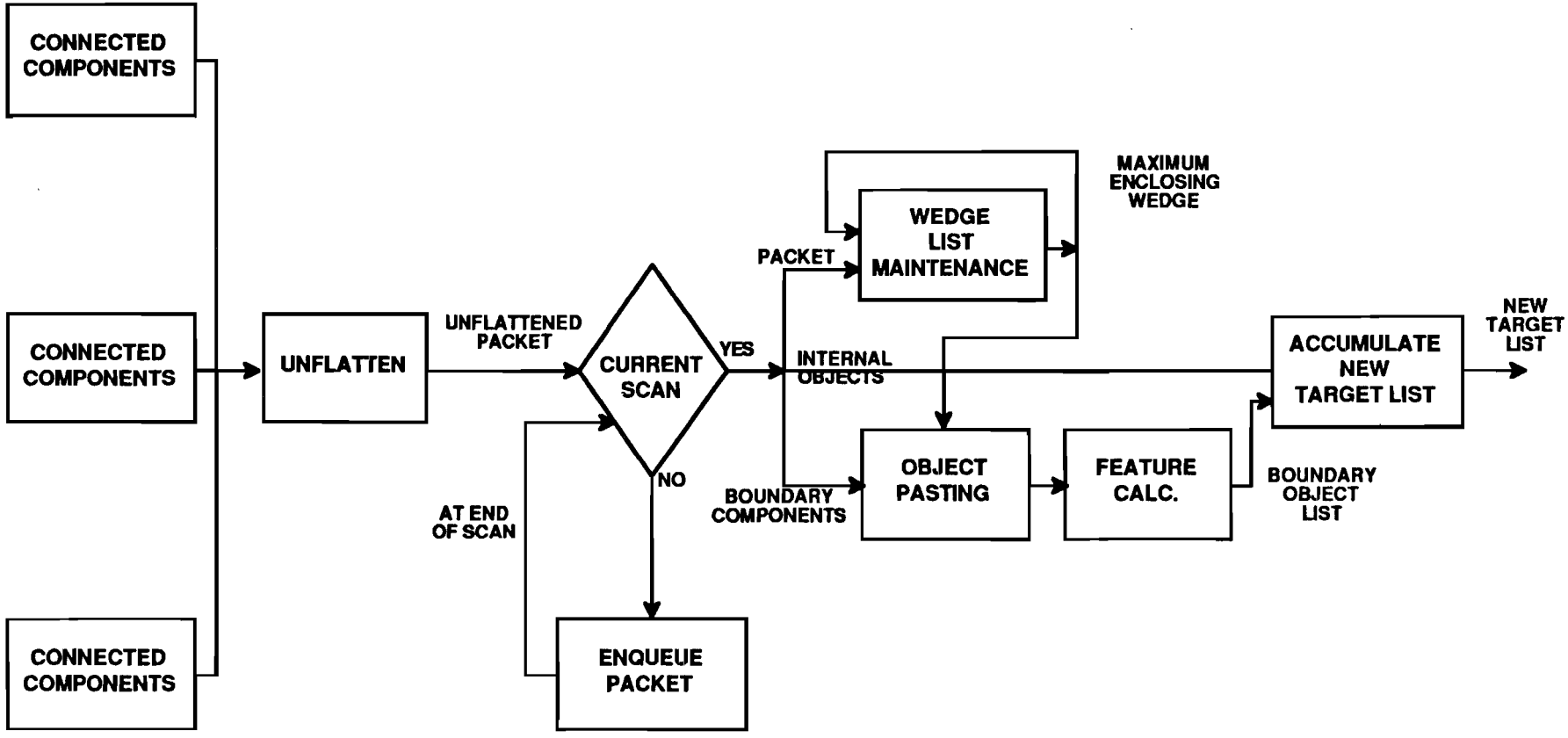


Figure 7-3-7. Architecture of merge stage.

The new target list is initially empty for each new wedge. At this point, all objects internal to the wedge are added to the list. A cumulative unfinished list for all boundary components not yet completed is kept. Steps V and VI are then calculated together. First, the Components to Objects procedure is performed with the unfinished list serving as the object list and the new boundary component list as the component list. The resulting objects are then sorted as either being within the maximal enclosing wedge or not. Those that are within the maximal enclosing wedge are removed from the unfinished list. Their features are calculated. They are then transferred to the new target list.

When a target is cut by a wedge boundary, the above method puts both parts as components into a single object. This leads to an identical centroid and area, but is inappropriate for features such as perimeter. The code has been written and tested for determining which components need to be pasted back together and for performing the pasting operation. Due to real-time load, this capability is inactive.

7.3.5.1.2 Scan-to-Scan Association

In order to explain the basic engine for the Scan to Scan Association, the needed data objects must be described. As previously mentioned, lists of components form objects. Lists of objects from subsequent scans form tracks (hence they are lists of lists). A list of tracks that all meet a particular criterion is referred to as a *tracklist*. The system currently maintains six tracklists, one of which is a list of tracks to be terminated in the current scan.

Starting with a list of new objects from the current scan, for each tracklist for which it applies, we proceed as follows:

1) A "best match under threshold" algorithm is applied to match objects to tracks. When a match takes place, appropriate track features are calculated.
2) Matched tracks are reported along with certain features to Sensor Fusion
3) Unmatched objects are collected for matching with the next list.

At the end of the scan the following operations are performed:

1) Promote tracks from one list to another
2) Extract certain track features, most notably position projection.

The tracklists are as follows:

1) High-confidence tracks
2) Bad drop tracks
3) Low-confidence tracks seen more than once
4) Single-object tracks from previous scans
5) Single-object tracks from the current scan
6) Dropped tracks.

Lists 1,3 and 4 use the "best match under threshold" algorithm for matching. The "best match under threshold" algorithm takes as input a list of new objects from the current scan (some may have been removed by matching with previous tracklists) and a tracklist. The algorithm proceeds as follows:

1) Take the next object off the list of objects to be matched. Call it O.

2) Set the min distance (M.D.) to infinity and set a pointer to the beginning of the tracklist.

3) Take the next track off the tracklist. Call it T.

4) Find the distance according to the match metric for the tracklist between T and O. If it is above the threshold, go to 5. Otherwise, compare it to M.D. If it is greater than M.D., go to 5. If it is less, check if the latest entry in the track is from the current scan. If it is not , make the new distance the M.D. and save a pointer to this track. If it is, compare the distance of the new object to the distance of the current matched object . If the new distance is smaller, make it M.D. and save a pointer to it.

5) If there are no more tracks, continue. Otherwise, go to 3.

6) If M.D. is still infinity, then put the target on the unmatched list (to be passed to the next track list and go to 1. If M.D. is less than infinity, check if the best track has an entry from the current scan. If it does not, insert O into the track and report the track immediately. If it does, remove the current last object and put it on the new object list, and then insert O into the track and report it immediately. Certain features for O are also calculated.

7) If there are no more objects, we are finished with the tracklist. Otherwise, go to 1.

In the above algorithm, a track can be reported more than once in a single scan if multiple objects match it. The last report is the correct one. Also note that the distance metric is the geometric distance from the projected position to the object position. For most situations, the projection is simply a two-point projection. However, for two runways, a pier at the end of the runway leads to a high-clutter situation and a lower-gain filter for position projection is needed. Thus, when a target is lined up with these runways and is over the pier, the projection is calculated from the average velocity from the last five sightings.

The features calculated for matched tracks include duration of track in number of scans, integrated distance of track from first sighting and whether this is over a threshold, whether the latest sighting is on a map of the "normal" movement area, both two-point and averaged velocity, and the *aspect graph* for the track.

As described in the previous section, an aspect graph is a table of aircraft features as a function of range from and orientation with respect to the radar. Currently, the table has 18 entries at 10 degree spacing for orientation and a log distance scale with an entry for every doubling in distance. The only feature currently used is area. The *aspect angle* is based on the orientation of the velocity vector with respect to the radial from the radar to the centroid (see Figure 7-3-8). Since the aspect angle calculated this way is unreliable for very slow moving or stopped aircraft, cells are only populated when the velocity

is above a low threshold. Also, for efficiency aspect graph entries are only calculated after the aircraft has passed the leadin criterion.

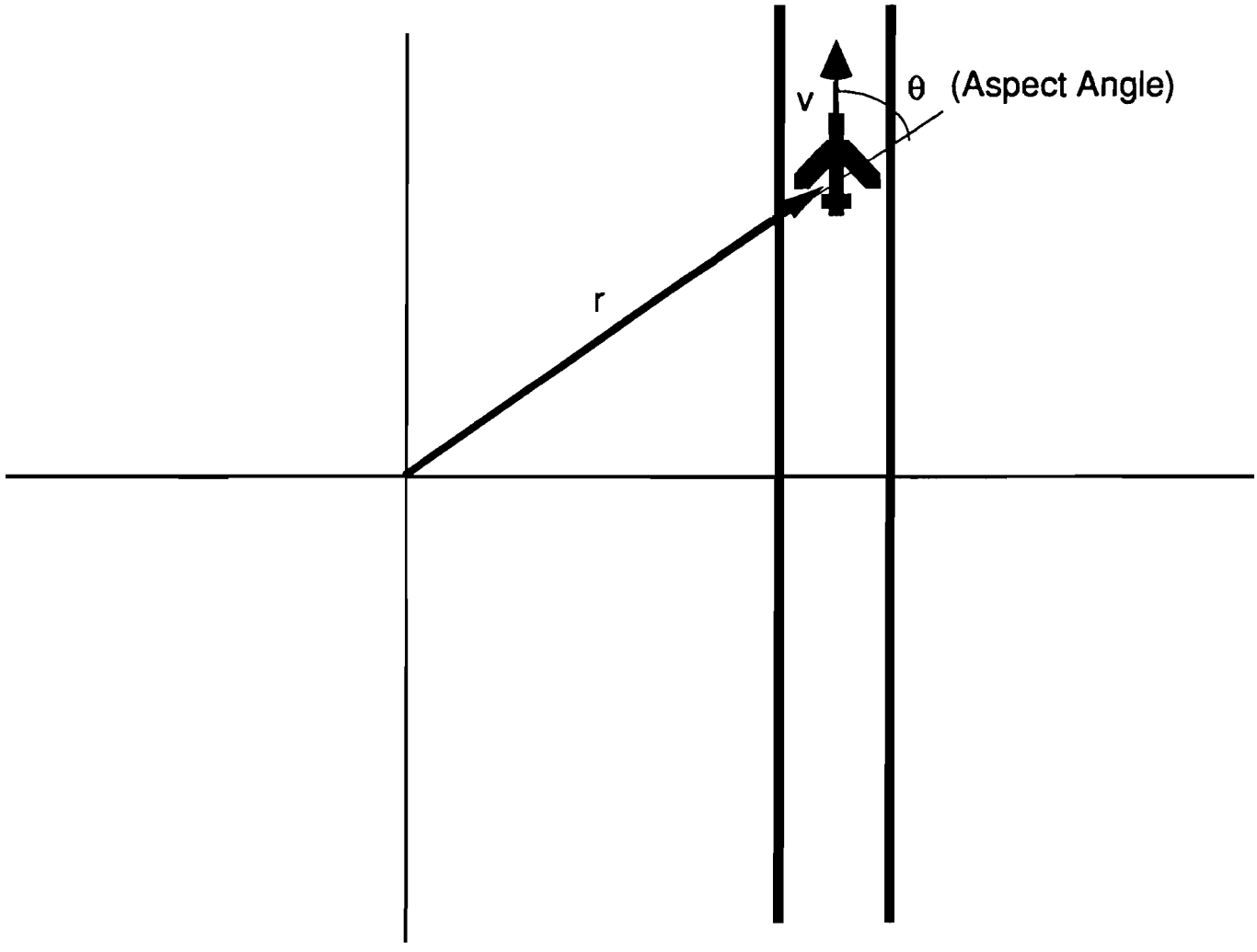


Figure 7-3-8. Definition of aspect angle.

7.3.5.1.3 Track Lists

The high-confidence track list contains those tracks whose integrated distance is above the threshold. These tracks are almost never caused by multipath. As described above, objects match this list using the standard position-projection-based best-fit algorithm. The bad drop list contains tracks that appear to have been dropped incorrectly. The criterion for this is discussed below in the section on transitions between lists. Best fit is not used here, and the first object that matches a track is chosen. We

did this only for ease of implementation and should be corrected. The match criterion between track and object is more complicated here. The aspect graph of the track in question records the last cell in which the aircraft was seen. To match the bad drop, a new object must satisfy the following:

1) It is close to the feature values of the last seen aspect graph cell
2) The new object is in the region covered by the surface map.

The next list (list 3) consists of tracks that have at least 2 sightings but have not passed leadin. As described in the section on algorithms, leadin is an accumulated travel distance threshold that a target must have traveled to be declared high confidence. It also uses projection-based best fit for matching. List 4 consists of tracks seen once in previous scans and matches to it also use projection based best fit. All objects that fail to match any of the other lists initiate their own tracks on list number 5. These lists are given track numbers that remain with the track for its lifetime.

At the end of each scan, tracks are allowed to transition between lists. The transition graph is shown in Figure 7-3-9. If a high-confidence track has not been matched this scan, several tests are performed on its recent values. If the recent average velocity is below a threshold and the last two out of three hits are on the movement area, it is moved to the bad drop list. If it does not meet these criteria, it is position coasted for a few scans. Entries in a bad drop list track must "position-project" correctly for several scans for the bad drop to be promoted to the high-confidence list. Low-confidence tracks in list 3 must pass the leadin criterion to be promoted to the high-confidence list. A second sighting within a large (~130 m) radius is all that is needed to progress from 4 to 3. List 5 is automatically promoted to list 4. Tracks that have run out of coasts or have gone beyond the bad-drop reacquisition time limit go onto the dropped list.

7.3.5.2 Output Data Description

The output consists of individual track updates. The data includes the following:

1) Track number
2) Centroid (2-d)
3) Area
4) Confidence (goes from 0 to 1 after passing leadin)
5) New and dropped track indication.

This is sent out for every target whether it has passed leadin or not. Because of best fit, multiple reports for the same target may occur. With respect to bad drop, while the track is on the bad drop list, no reports are made.

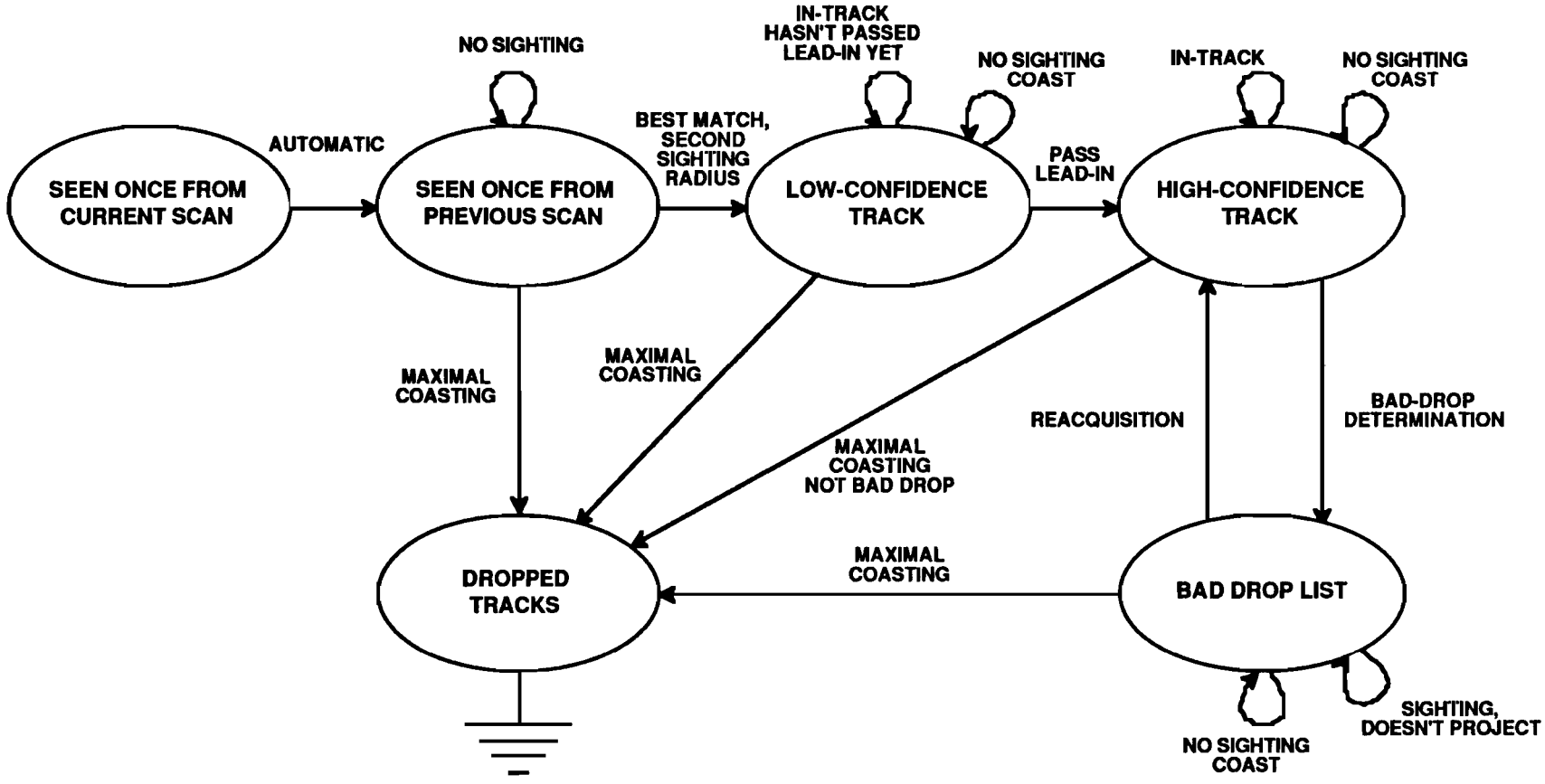


Figure 7-3-9. Transition graph. Allowed transitions for tracks are shown.

7.3.5.3 Databases

The parameter database for the Merging and the Scan-to-Scan Association is larger than for the Connected Components process. It includes the following:

1) Area threshold
2) Components-to-Object distance
3) In-track projected position tolerance
4) Search radius for second sighting
5) Number of coasts for various situations
6) Pier fix tolerances
7) Equations and locations of pier
8) Minimum number of scans for high-confidence tracks
9) Minimum accumulated track length for high-confidence tracks
10) Minimum maximum distance from first sighting for high-confidence track
11) A file containing surface map
12) Velocity range for the bad drop mechanism
13) Aspect graph size and limits
14) Number of radials in a packet
15) Radar parameters
16) The region in which to search for bad drop reacquisition
17) Timeout for bad drop
18) Fit parameters for features for reacquired targets.

7.3.5.4 Significant Problems/Solutions

The following are the most significant problems addressed in the scan-to-scan association portion of the algorithm:

1) Multipath targets
2) Inappropriate drops
3) Quick reacquisition of high confidence targets if dropped
4) Capture of "important" targets by lower confidence tracks
5) Providing surveillance over the piers (high-clutter region)

We identify the multipath targets primarily through the leadin criterion. This consists primarily of a threshold for the distance moved. However, the target must also have moved more than a minimum "max" distance from its initial sighting to prevent stationary objects with a small amount of dither from passing leadin.

Drops that take place on the movement area, that take place below departure velocities, and involve high-confidence targets are deemed to be inappropriate drops. This criterion quite accurately detects inappropriate drops. Once these drops are detected, it is important to reacquire them quickly (before forcing them to pass leadin again). This is accomplished by using the aspect graph to match features to the dropped target. In the initial stages of development, various features were analyzed for scan-to-scan consistency within a track. It was found that major discontinuities were found for even simple features such as area in regions where targets turned, and enormous longer-term variations were found as well.

The reason for this is straightforward. Provided that an aircraft is constrained to the roughly two-dimensional surface of the airport, its return, neglecting ground bounce effects and shadowing, is dependent only on its heading and range. It was found that targets at about the same range and heading had very similar features, with the standard deviation being typically under 10%. Using the features from the last cell of the aspect graph in which the target was seen seems to match the dropped target in many situations. Some exceptions include shadowing and a target that has turned significantly since the latest sightings.

It was also found that allowing all tracks to compete on an equal basis for new targets led to frequent "theft" of important in-track targets by new, low-confidence tracks. Clearly, this is so for first sightings since the association radius for these targets is quite large; but it was also found that if bad drops had "first crack" at the targets, some unmatched bad drops would latch on to subsequent aircraft. To minimize this, reliable tracks are matched first and only those targets that are not taken are made available to bad drops.

Finally, a low-gain filter is used for the piers to compensate for the degraded surveillance, using the assumption that targets fly straight over the piers.

7.3.5.5 Suggested Improvements

The areas for improvement include the following:

1) The concept of scan boundaries is rather artificial. It fits in well with the Logan geometry where surveillance is provided for only about 180 degrees of the airport. However, it is unclear where to put the cut for an airport with the radar in the middle of the runway configuration. In addition, since processing for wedges from subsequent scans is suspended until the current scan is completed, it leads to sub-optimal use of the processor.

2) While the algorithm exists for splicing parts of aircraft cut by boundaries back together again for such features as perimeter, they must be sped up to be consistently real-time (faster hardware would do the trick as well).

3) ARTS information is fused with ASDE data downstream in the sensor fusion process. If ARTS information were provided directly to the ASDE processing, it could assist in such things as initial acquisition based on feature matching for a known aircraft type.

4) The aspect graph used currently for reacquisition is a dynamic aspect graph, i.e., it is calculated on the fly for a given aircraft. If an aircraft is lost in a turn, the last-cell-seen aspect graph information will not be a good fit to the features of the turned aircraft. A library of aspect graphs for different aircraft would do much better in this respect. With ARTS, the correct aspect graph could be immediately accessed for arrivals. For departures, a fit to a sufficient number of cells could be used to match an aircraft to a library aspect graph.

5) The aspect graph boundaries are currently arbitrary, based on even heading divisions and even log range divisions. The boundaries of cells should be adaptive, depending on such things as the obscuration and illumination of scatterers.

6) Currently, the first bad drop list entry that is "close enough" to a new target is matched to it. For isolated bad drops this is fine, but in crowded environments like the runup pad, best fit should be implemented.

7) For more accurate fits, multiple features should be implemented for each cell of the aspect graph. Perimeter is available but its calculation takes too long for the current processors. Eventually, greyscale features should be used as well.

8) For the second sighting of an aircraft, since velocity information is not yet available, a large match radius is used. The number used is based on the maximum distance an object can move in a single scan. While this is a conservative assumption, it leads to many matches within the radius for which only the best fit is used. However, in certain areas of the surface, a much smaller radius should be used corresponding to lower maximum velocities.

9) While most multipath is a result of aircraft-to-aircraft (or ground vehicle) reflections, some involve reflections off of fixed targets like buildings. Mapping these multipath sources would allow us to quickly eliminate them.

7.4 SCIP INTERFACE

7.4.1 Subsystem Architecture

As was mentioned in the System Overview, the SCIP interface consists of two parts, which are shown in Figure 7-4-1. The first part, ADIDS-SCIP, executes on an IBM PC-compatible computer and extracts the information flowing across the ASR-9-SCIP-to-ARTS-IOP interface and sends it out to a serial interface or modem. The second part, SCIP-receiver, executes on a UNIX workstation and receives the SCIP information and makes it available to other processes on the same machine or connected via a network.

ADIDS-SCIP has two execution threads. The first thread is interrupt driven. It pulls the data in from the ARTS interface hardware, checks parity and framing, applies the geometric and message-type filters, and for messages that pass the filter, it adds a 32-bit header and sends the message to the serial interface. The second thread receives the commands from the serial interface, detects loss of incoming data to allow automatic SCIP port switching, and updates the local display of message statistics.

ADIDS-SCIP receives the following four message types from the ARTS: beacon target messages, radar only target messages, sector messages, and alarm messages (which include a heartbeat message delivered once per scan). The format of these messages is discussed in Section 4.2.1. The messages can be filtered based on the following:

- message type (all messages)
- min/max range (beacon and radar only messages)
- start/end azimuth (beacon and radar only messages)
- min/max altitude (beacon messages).

The output message format is identical to the input except that a 32-bit header is added and the parity bits (bits 30 and 31) of the data are cleared. The header has the following format:

- bits 30 and 31 set (output framing marker)
- bit 29 set if input from SCIP port B, otherwise cleared
- bits 22-28 are a 7-bit ASCII message type code -- 'B' for beacon, 'R' for radar only, 'S' for sector, and 'A' for alarm
- bits 0-21 are a 22-bit integer time-stamp representing the number of DOS clock ticks since midnight (approximately 18.2 ticks per second).

The client of ADIDS-SCIP can issue commands to perform the following:

- set minimum or maximum range or altitude
- set the start and end azimuth
- set the time-stamp clock
- set which of the four message types should be sent or suppressed

- request which SCIP port to listen to
- turn on data transmission (password protected)
- stop data transmission (also occurs automatically when the connection is via a modem and the Data Carrier Detect (DCD) signal is lost).

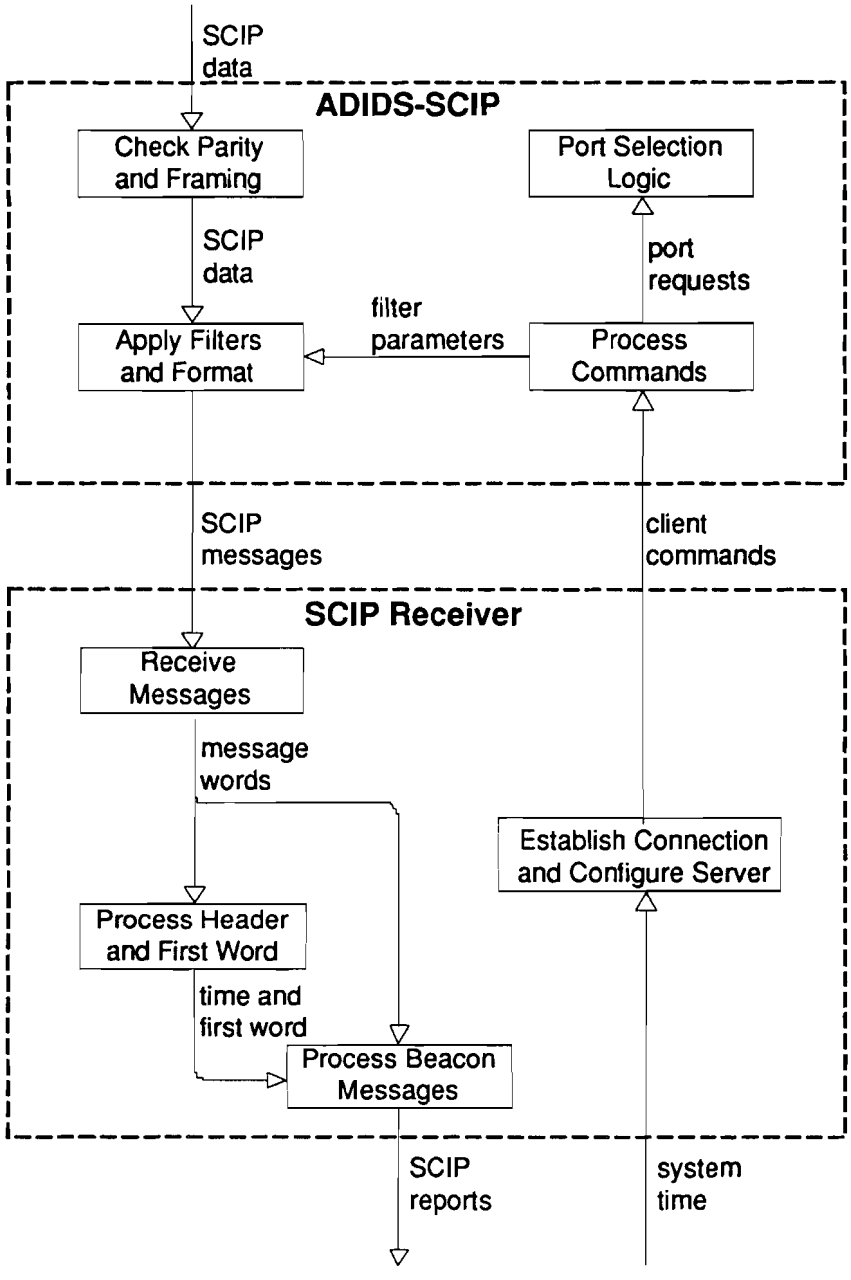


Figure 7-4-1. SCIP interface.

The SCIP-receiver receives messages from ADIDS-SCIP via a modem connection, reformats the data, and acts as a server to make the data available to other processes. SCIP-receiver has four primary functional elements as shown in Figure 7-4-1. The first element interacts with the modem to establish the connection to ADIDS-SCIP and sends a sequence of client commands to initialize the ADIDS-SCIP filtering parameters and the time-stamp clock. The second element performs framing and reads the first two 32-bit words of a message (the header word and the first word of data). Framing is based on the two marker bits of each word (bits 30 and 31), a valid message type, and any required set or cleared bits in the first word of ARTS data (second word of message). The second element also checks that the time-stamp is within 1.5 seconds of the master-clock time and sends ADIDS-SCIP a time command if it is not. The third element processes beacon messages. It reads the remaining two 32-bit words and, for messages with valid mode-A codes, it unpacks the range, azimuth, mode-A code, and altitude fields into standard floating point and integer representations and performs unit conversions. It then passes the re-formatted data on to the interprocess communication software for transmission. The fourth element is responsible for the actual reading of the data from the modem. It is designed to detect the loss of the modem Data Carrier Detect signal or the loss of incoming data. When the loss of the modem Data Carrier Detect signal is detected, the first element is invoked to re-establish connection and re-send the client commands (including the transmit command). When the loss of data is detected, the sequence of client commands is re-sent to command ADIDS-SCIP to transmit. In both cases, the complete sequence of commands is sent in case ADIDS-SCIP has been re-started.

7.4.2 Output Data Description

Table 7.4.1 shows the SCIP-receiver output message fields.

TABLE 7.4.1

SCIP-Receiver Output Message Fields

Field	Comments
mode-A code	only targets with highest mode-A code confidence are passed on
range	meters (floating point), resolution is 1/64 nmi
azimuth	radians (floating point), resolution is 1/4096 of a circle, 0 azimuth is magnetic north
altitude	feet (floating point), resolution is 100 feet
altitude valid flag	boolean, 0 => false, non-zero => true
time	seconds (floating point), based on the master-clock

Range and azimuth are relative to the radar with zero azimuth typically aligned with Magnetic North. These messages are only sent when a beacon message is received and the message's mode-A code confidence is high.

7.4.3 Significant Problems/Solutions

Considerable development was required to make the processes of establishing the modem connection and of starting the data transmission robust. When trying to establish the modem connection, for example, the connection sequence (i.e., hang up, dial, wait for Data Carrier Detected signal) is repeated for many times before giving up. The ADIDS-receiver can re-establish the data flow after most connection or command transmission failures as long as ADIDS-SCIP is operating.

7.4.4 Suggested Improvements

Much of the difficulty involved in getting this subsystem to function reliably centered around details of the modem connection. In a more permanent installation, a dedicated and more robust network connection or improved modem handling software in both processes would be advisable.

The SCIP-receiver currently does not verify that all the client commands were received correctly (and ADIDS-SCIP provides no mechanism to support such verification) other than to check the incoming time stamps and to detect the failure of a start-transmission command. On the other hand, in a more permanent installation, this flexibility may not be required.

7.5 MDBM INTERFACE

7.5.1 Subsystem Architecture

The MDBM interface is structurally very similar to the SCIP interface. It consists of two parts, which are shown in Figure 7-4-2. The First part, ADIDS-MDBM, executes on an IBM PC-compatible computer and extracts the information flowing across the ARTS-IOP-to-MDBM interface and sends it out to a serial interface or modem. The second part, MDBM-receiver, executes on a UNIX workstation to receive the information and make it available to other processes on the same machine or connected via a network.

ADIDS-MDBM has two execution threads. The first thread is interrupt driven. It pulls the data in from the ARTS interface hardware and places it in a ring buffer. Command words and data words are received from the hardware separately, but are placed into the same internal buffer with an extra marker word included to flag IOP-MDBM command words. This marker word is used to find the first command word when performing input data framing. The second thread is a continuous loop. It extracts the data from the ring buffer, and checks parity and framing. Because the data flowing from the IOP to the MDBM only represents MDBM memory updates, the main thread must emulate the internal memory of the MDBM and the MDBM memory-update process. After a section of emulated memory has been updated, the main thread applies the geometric and message-type filters to each updated message, and for messages that pass the filter, it adds a 32-bit header and sends the message to the serial interface. Between each memory update sequence, the main thread handles the receipt of commands from the serial interface and updates the local display of message statistics. The serial interface is also checked whenever the input ring buffer is empty because there is often a delay between the arrival of command words and the data words.

ADIDS-MDBM processes the following four message types from the IOP: full data block, altitude (limited) data block, MSAW data block, and single symbol data block. The format of these messages is discussed in Section 4.2.2. The messages can be filtered based on the following:

- message type
- DBM number (all unsuppressed messages)
- DBM number (full and MSAW data block, regardless of suppression)
- min/max X coordinate
- min/max Y coordinate.

The output message format is shown in Tables 4.2a-d except that a 32-bit header is added and the parity bits (bits 30 and 31) of the data are cleared. The header has the following format:

- bits 30 and 31 set (output framing marker)
- bit 29 set if input from MDBM port B, otherwise cleared
- bits 26-28 are an 8-bit integer representing the DBM number
- bits 22-25 are the low order 4-bits of an ASCII message type code -- 'F' for Full, 'A' for altitude, 'M' for MSAW, and 'S' for single symbol

- bits 0-21 are a 22-bit integer time-stamp representing the number of DOS clock ticks since midnight (approximately 18.2 ticks per second).

The client of ADIDS-MDBM can issue commands to perform the following:

- set minimum or maximum X or Y coordinate
- set which of the four message types should be sent or suppressed
- set the time-stamp clock
- turn on data transmission (password protected)
- stop data transmission (also occurs automatically when the connection is via a modem and the Data Carrier Detect (DCD) signal is lost).

The MDBM-receiver receives messages from ADIDS-MDBM via a modem connection, reformats the data, and acts as a server to make the data available to other processes. MDBM-receive has four primary functional elements as shown in Figure 7-4-2. The first element interacts with the modem to establish the connection to ADIDS-MDBM and sends a sequence of client commands to initialize the ADIDS-MDBM filtering parameters and the time-stamp clock. The second element performs framing and reads the first two 32-bit words of a message (the header word and the first word of data). Framing is based on the two marker bits of each word (bits 30 and 31), a valid message type, and six required set bits in the first word of full, MSAW, and altitude data blocks. The second element also checks that the time-stamp is within 1.5 seconds of the master-clock time and sends ADIDS-MDBM a time command if it is not. The next element processes each message. For altitude, full, or MSAW data block messages, it reads in the remaining B word and C words and decodes the characters in the C words to form 1, 2, or 3 strings respectively that represent the 1 to 3 lines of text in a DEDS data tag. These strings are parsed to extract a heavy weight indicator and, for full and MSAW data blocks, flight id and aircraft type. Since the aircraft type field is shared with controller-entered scratch pad information, the aircraft type is checked for validity against a database of valid types. The coordinates in the B words are converted from their 1's complement representation to floating point and from 1/16 nmi to meters and the symbol TI code is converted to an ASCII character. The process message element then passes the re-formatted data on to the interprocess communication software for transmission. The fourth element is responsible for the actual reading of the data from the modem. It is designed to detect the loss of the modem Data Carrier Detect signal or the loss of incoming data. When the loss of the modem Data Carrier Detect signal is detected, the first element is invoked to re-establish connection and re-send the client commands (including the transmit command). When the loss of data is detected, the sequence of client commands is re-sent to command ADIDS-MDBM to transmit. In both cases, the complete sequence of commands is sent in case ADIDS-MDBM has been re-started.

7.5.2 Output Data Description

Table 7.5.1 shows the MDBM-receiver output message fields.

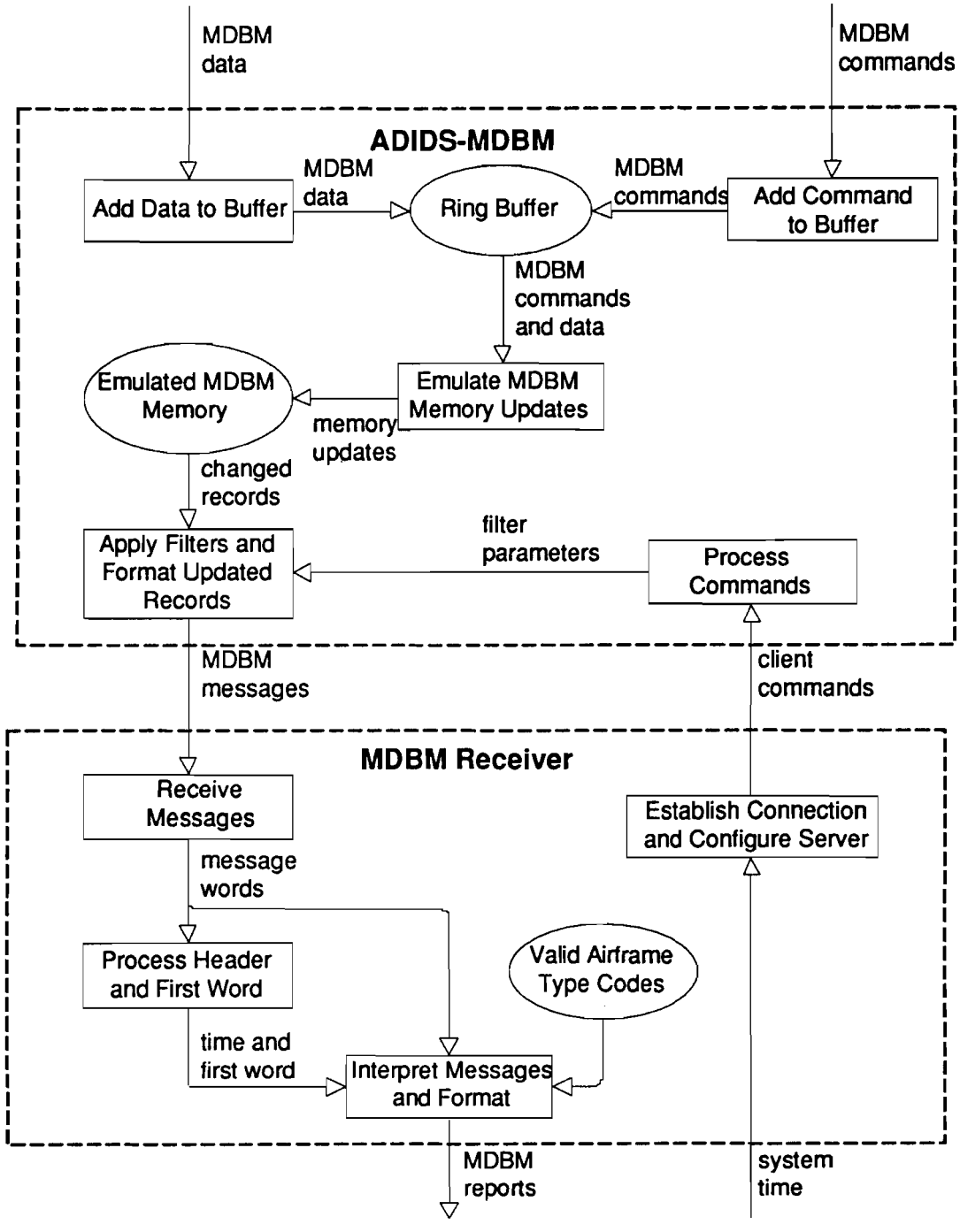


Figure 7-5-1. MDBM interface.

TABLE 7.5.1
MDBM-Receiver Output Message Fields

Field	Comments
X and Y position	meters (floating point), resolution is 1/16 nmi, Y axis is aligned with magnetic north (SCIP 0 azimuth)
flight id	UAL1234, N123AB, etc.
flight id valid	boolean, 0 => false, non-zero => true
aircraft type	B727, DC9, etc.
aircraft type valid	boolean, 0 => false, non-zero => true
heavy weight class	boolean, 0 => false, non-zero => true
heavy weight valid	boolean, 0 => false, non-zero => true
control position	letter designating the control position for the target
ctl. pos. valid	boolean, 0 => false, non-zero => true
dbm number	0-7, which dbm is showing this information
tag direction	NE, SE, SW, NW
tag line 0	string for MSAW/CA line of ARTS data tag
tag line 1	string for identification line of ARTS data tag
tag line 2	string for altitude and speed/type/scratch pad line of ARTS data tag
time	seconds (floating point), based on the master-clock

Most of the fields in the message have associated validity flags because not all message types provide each piece of information and because some pieces of information time-share the same position in an ARTS data tag.

7.5.3 Databases

The MDBM-receiver uses a database of valid aircraft type codes to assure that all aircraft types parsed from the data block information are valid. This database is an ASCII file containing all the (currently 597) valid codes that are separated from each other by spaces, tabs, and/or carriage returns.

7.5.4 Significant Problems/Solutions

A fair degree of effort was required to understand the operation of the MDBM and to develop a proper emulation of the MDBM memory updating process. Also, since the MDBM interface has the

same modem-connection architecture as the SCIP interface, the comments in Section 7.4.3 apply here as well.

7.5.5 Suggested Improvements

Much of the difficulty involved in getting this subsystem to function reliably centered around details of the modem connection. In a more permanent installation, a dedicated and more robust network connection or improved modem handling software in both processes would be advisable.

The MDBM-receiver currently does not verify that all the client commands were received correctly (and ADIDS-MDBM provides no mechanism to support such verification) other than to check the incoming time stamps and to detect the failure of a start-transmission command. On the other hand, in a more permanent installation, this flexibility may not be required.

7.6 SENSOR FUSION

7.6.1 Introduction

Sensor fusion is responsible for accepting surveillance data from the ADIDS ARTS tap and ASDE processing, and for producing a single fused surveillance data stream.

Sensor fusion is an event-driven executable that has four real-time input streams: ASDE, ARTS-SCIP, ARTS-MDBM, and timer events. Sensor fusion translates all target positions to a common coordinate system, determines and applies time offsets to synchronize time stamps, smoothes target positions, estimates target velocity and acceleration, de-multipaths the ARTS-SCIP beacon reports, filters false targets from the ASDE data, transfers identity and type information from the ARTS-MDBM data to the ARTS-SCIP beacon data, determines which ASDE targets and ARTS-SCIP beacon targets represent the same target and should be combined into a single track, and coasts tracks through coverage gaps.

The output of sensor fusion is a single data stream that contains fusion track reports of reliable tracks. Sensor fusion also has a basic capability to write selected internal information to a file for debugging purposes.

7.6.2 Process Architecture

Sensor fusion represents surveillance input events as SCIP_TargetReports, MDBM_TargetReports, and ASDE_TargetReports, each a class derived from a common base class TargetReport. The various TargetReport derived classes have member functions responsible for initialization, input, output, and hashing.

Sensor fusion represents surveillance subtracks using two classes, ARTS_Track and ASDE_Track, derived from the common base class SubTrack. The base class methods are responsible for implementing the alpha-beta-gamma filter, performing current position estimates, and maintaining a copy of the most recent TargetReport. The derived classes implement the sensor-specific functionality corresponding to updating SubTracks with TargetReports, performing the time offset, maintaining the sealevel correction for ARTS altitude, eliminating ARTS multipath, and loading and performing sensor-specific target filters.

No MDBM_Track class exists. This is because MDBM data represent the same surveillance information as SCIP data. The MDBM_TargetReport handler performs the MDBM-SCIP subtrack subfusion directly, without maintaining any MDBM subtrack independent of the ARTS_Track. All pertinent information from the MDBM_TargetReports is maintained by the ARTS_Track on subfusion. Because consecutive MDBM_TargetReports corresponding to the same ARTS_Track may contain different types of valid data, only the valid data from each MDBM_TargetReport is updated into the ARTS_Track. In this manner, after only a few successful subfusions, an ARTS_Track will contain all the useful information and will correctly reflect any changed information available from the MDBM.

Sensor fusion maintains tracks using the Track class. Tracks are responsible for fusing and unfusing SubTracks, estimating surveillance clock offsets, coasting, loading and performing fused track filtering, and generating TrackReports for transmission to client processes.

TrackReports are of three basic types, distinguished by the flags set in the message: normal track report messages, track suspend messages, and track drop messages. Normal track report messages are sent for all Track surveillance updates that pass the fused track filter. Suspend messages are sent for Tracks that are no longer to be sent to clients, but that are still maintained by sensor fusion. Drop

messages are sent for Tracks that have been reported, but are no longer being maintained by sensor fusion. The normal clients of sensor fusion are the surface monitor and ASDE display processes.

7.6.3 Process Data Flow

Figure 7-6-1 shows the process data flow for sensor fusion. Sensor fusion is event driven. It is a client of ASDE surveillance processing, the ARTS-SCIP tap, the ARTS-MDBM tap, and the master clock. On initialization, sensor fusion reads an airport database, creates an output channel to which it will write TrackReports, connects to the master clock, and then reads the configuration files as specified in its command line. These specify which sources of surveillance sensor fusion should connect to. The Event corresponding to each channel is registered with the EventDispatcher along with the function for handling that event. A periodically occurring event is also registered to trigger the coasting function. Each type of TargetReport has its own report handler. The target report and clock event handlers together control the processing flow once the process is done initializing.

A SCIP or ASDE surveillance input event, represented by a TargetReport, causes sensor fusion to check the appropriate SubTrack store for a SubTrack with the same ID. If no such SubTrack exists, then a new one is created. The SubTrack is then updated using the new surveillance information. An MDBM surveillance input event causes sensor fusion to check the SCIP SubTrack store for a match. If such a SubTrack exists, then it is updated using the new surveillance information. If no such SubTrack exists, no action is performed for this MDBM_TargetReport.

The updating process for ASDE_TargetReports is as follows. The first 100 ASDE reports received by sensor fusion are used to estimate the timestamp offset between ASDE and ARTS surveillance. All reports are used to perform an overall latency estimate that is used in coasting. If the ASDE report is a rereport (a second report on the same track for a particular scan), the previous erroneous report is removed from the SubTrack and its effects on estimated parameters of that SubTrack are undone. The new report is compared with the SubTrack's projected position. If the projection error exceeds a threshold, the report is added to a list of reports deferred for a short time to wait to see if a better match will occur. This list of deferred reports is checked every time a new report is received to determine if the report has been deferred sufficiently long to warrant its use or if it has been outmoded by subsequent reports for the same SubTrack. Once an ASDE TargetReport has passed the deferral test, it is used to update the SubTrack's position, velocity, and acceleration estimates using an alpha-beta-gamma filter, a copy of the TargetReport is appended to the SubTrack, the creep velocity is estimated if the projection error is small, the prefusion ASDE Filter is applied, and the SubTrack's Track is told to update itself.

The updating process for ARTS_TargetReports is similar to that for ASDE TargetReports, with some exceptions. No initial timestamp offset estimate is performed. Before any SubTrack maintenance is performed, the special ARTS multipath rejection logic is invoked to determine if either of the present or the previous report to this ARTS_Track is multipath and should be rejected. The altitude list maintenance is performed to produce an ground level pressure altitude correction estimate. No list of deferred reports is maintained, although this could be added easily. A prefusion ARTS Filter is used.

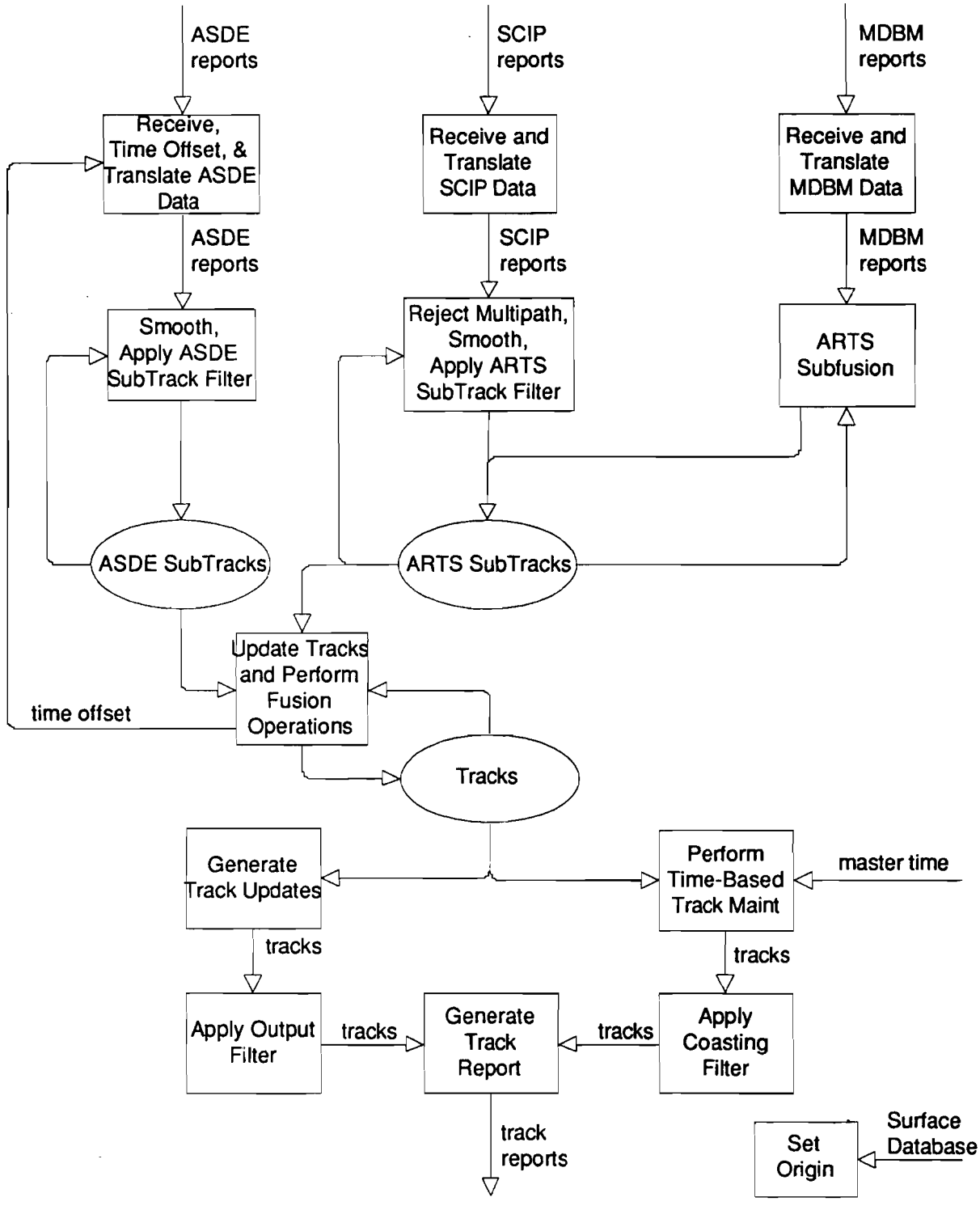


Figure 7-6-1. Data flow diagram for sensor fusion.

The updating process for MDBM_TargetReports results in updating ARTS_Tracks when appropriate. The MDBM_TargetReport is used to update the first found ARTS_Track whose flight ID or position matches that of the MDBM_TargetReport. The updating results in copying the appropriate new information from the MDBM_TargetReport into the ARTS_Track. No fusion consequences are possible and no TrackReport is generated.

When a Track is told to update itself based on a surveillance update, it performs track status maintenance. If it is a fused track, it verifies that the two SubTracks still should be fused together. If fusion is no longer appropriate, the ASDE and ARTS SubTracks are separated into two Tracks, and the resulting two unfused Tracks are allowed to check for fusion with any other appropriate Tracks. If the original Track is not a fused track, it looks for an appropriate fusion candidate Track. If an unambiguous fusion partner is found, the two Tracks are fused, and the surveillance time offset is reestimated. The fusion is required to be unambiguous in both directions. After all Track maintenance is performed, the Track copies the appropriate data from each of its SubTracks and invokes the postfusion Track Filter. If the Track passes the Filter, a TrackReport is generated for transmission to any client process.

Coast events from the master clock cause sensor fusion to review all Tracks for currency. Those that are to be dropped are deleted. Those that are stale but not suppressed are coasted to the current time minus the appropriate surveillance source's estimated latency. If the Track passes a coasting Filter, a TrackReport is generated, otherwise the Track is suppressed and a suspend message is generated if appropriate.

7.6.4 Sensor Fusion Databases

Sensor fusion reads several databases to derive the parameters needed for its various functions. Sensor fusion reads three radar databases, one for each of the ASDE, SCIP, and MDBM surveillance inputs. These databases include a source indicator, a real/simulated indicator, the radar location, and range and azimuth correction factors. The ASDE and SCIP databases also include the alpha-beta-gamma filter gains, latency estimate gain, stale time, number of reports to save, and the number of reports before acceleration is valid. The ARTS database also includes the altitude coast time, the ARTS multipath rejection distance parameters, and the minimum airspeed for nonzero altitude. The ASDE database also includes the poor-match filter gains, the poor-match distance criterion, the drop time, and the creep distance parameters.

Sensor fusion reads a database containing parameters that affect the fusion process. These parameters include the ASDE surveillance radius, the padded ASDE surveillance radius, the ARTS altitude fusion limits, the firm fusion count, the fusion radius, the fusion radius velocity weight factor, the fusion radius default velocity error, ASDE and ARTS old track hit counts, the time between extrapolations, the surveillance offset time gain, the minimum velocity and maximum acceleration to update the surveillance offset time, the fusion radius multipliers, and the time limit on fusion verification.

Sensor fusion also reads a database containing the specification of its various Filters. Each Filter has a keyword indicating which Filter is being specified followed by a list of filter Regions with their appropriate parameters. The Filters are described in more detail in the following section.

7.6.5 Significant Problems and Solutions

One of the significant capabilities provided by the sensor fusion software is a flexible and generic area-dependent feature-based filter. Sensor fusion presently uses filters in four places: once for each of the two input surveillance datastreams, once for the fused datastream before output, and once for coasted

tracks before output. The coasted track filter is actually implemented as two separate Filters: an ARTS drop reprieve filter and a track extrapolation filter. Filters are represented using the Filter class, which includes member functions that allow application of the filter to the different types of SubTracks and Tracks, and a RegionList to specify the areas, features, and actions of the filter. The RegionList allows each Filter to operate over a number of Regions, each defined by a polygon with zero or more excluded polygons. Each Region tests a number of features of the SubTrack or Track. The features presently tested include allowed surveillance source (ARTS, ASDE, fused), track length, target area, net travel distance, sum travel distance, speed, track confidence, altitude, and stale time. If the features match and if the current position lies within the Region's defined area, then an action is applied to the SubTrack or Track. The actions presently defined include blessing and suppressing. The architecture allows the addition of additional features and actions as the need may arise. Filter specifications are read from an ASCII file at runtime. Thus much of the behavior of sensor fusion is determined by easily changed parameter files that specify the Filters.

7.6.6 Output Data Description

The output of sensor fusion is represented by the TrackReport class. The TrackReport output message fields are outlined in Tables 7.6.1a–b.

TABLE 7.6.1A

Sensor Fusion Output Message Fields (Part a)

Field	Comments
initial track bit	bit, 1 => new track, 0 => not a new track
track drop bit	bit, 1 => track being dropped from sensor fusion store, 0 => not being dropped
track suspend bit	bit, 1 => track not being dropped, but should no longer be used, 0 => otherwise
surveillance source	2 bits, ASDE, ARTS, or COAST
track swap bit	bit, 1 => track may have swapped, 0 => otherwise
heavy indicator	bit, 1 => track report is from a heavy aircraft, 0 => otherwise
simulated bit	bit, 1 => track report is simulated data, 0 => track is real data
track ID	unsigned long, sensor fusion-assigned unique track identifier
ASDE track ID	unsigned long, ASDE track number as assigned by ASDE processing
measurement time	double, seconds, time of position measurement for this track report
target reliability	float, chance that this target is really there, not used, always zero
track length	short, number of times this Track has been seen
coast length	short, number of times in a row this Track has been coasted by sensor fusion
position	two doubles, meters north and east of an airport reference point, always valid
velocity	two doubles, meters/sec north and east
velocity validity	boolean, 0 => velocity is invalid, 1 => velocity is valid
creep velocity	two doubles, meters/sec north and east, low-gain velocity estimate for slow tracks
creep velocity validity	boolean, 1 => creep velocity is likely to be a better estimate than the velocity, 0 => otherwise
acceleration	two doubles, meters/sec^2 north and east
acceleration validity	boolean, 0 => acceleration is invalid, 1 => acceleration is valid
position uncertainty	two doubles, meters north and east, as reported by ASDE processing, not used
velocity uncertainty	two doubles, meters/sec north and east, not used
acceleration uncertainty	two doubles, meters/sec^2 north and east, not used

TABLE 7.6.1B

Sensor Fusion Output Message Fields (Part b)

Field	Comments
track status	short, ARTS, ASDE, firmly fused, tentatively fused, or coasted
target extent	four shorts, meters ahead, behind, left, and right from the centroid
target centroid	two doubles, meters north and east of an airport reference point, as reported by ASDE processing
heading	float, target heading radians clockwise from north, as reported by ASDE, not used
extent uncertainty	four shorts, meters ahead, behind, left, and right, not implemented, 0
heading uncertainty	float, radians, as reported by ASDE, not used
altitude	short, feet above ground level
altitude validity	boolean, 0 => altitude is invalid, 1 => altitude is valid
mode A code	short, mode A squawk code reported by transponder, 0–4095
ground speed	short, meters/sec, as reported by ARTS, if known, else 0
aircraft ID	string, 0-7 ASCII characters, as reported by ARTS, if known, else null
aircraft type	string, 0-4 ASCII characters, as reported by ARTS, if known, else null
control position	ASCII character, as reported by ARTS, if known, else null

7.6.7 Suggested Improvements

Sensor fusion needs to have its subtrack code expanded and its track fusion code regularized to accept more and different kinds of sensors. These additional sensors may eventually include multiple ASDEs, ADS-Mode S, and the ASR-9 primary radar. This will likely require some additional capabilities, most notably an n-way clock offset equalization algorithm, SubTrack scan-to-scan association algorithms using Track and SubTrack information from multiple surveillance sources, correct treatment of corotating sensors, additional SubTrack multipath removal techniques, and more flexible modeling of surveillance coverage areas.

7.7 THE SURFACE MONITOR MODULE

7.7.0 Introduction

This section describes the surface monitor, which is responsible for implementing the rules described earlier in the Safety Logic chapter. First, the surface monitor products will be described. These include status light commands, alert message, diagnostic aids, and approach messages. Next, the architecture will be detailed. As an object-oriented, event-driven system, the surface monitor does not easily lend itself to a classical description of sequential data transformation; to get a sense of the system the important relationships between high-level objects must be understood. The architecture description will begin with a very general outline of what happens when a target report is received from Sensor Fusion. After this context has been supplied, the high-level objects will be presented and described in some detail. Design decisions will be explained as necessary.

The two databases that give the surface monitor a large measure of site independence and configurability will then be described. The first one, the Surface database, will be briefly described while the second one, the Safety database, will be taken in greater detail since its specific function is to allow surface monitor configuration. The mechanisms for configuration-dependent parameter specification will be explained here.

At this point the current implementation will have been described in moderate detail. A discussion of the design goals will then be presented. The purpose of this discussion is twofold: First, it demonstrates how and to what degree the architecture supports those goals. Second, it will be useful for anyone considering implementing such a system to compare their goals to these. It may be that some of the goals here would not be applicable in a different context, and in such cases a different approach may be warranted.

Finally, a list of problems will be given. These include numerous small issues that have not been implemented due to time constraints, as well as some more fundamental design-level problems that Lincoln Laboratory feels should be addressed when building a subsequent version.

7.7.1 Surface Monitor Products

The surface monitor runs in a distributed client-server environment, and has roles as both a client and a server. This section will briefly describe its client role, and will then focus on detailing the services it provides other system processes.

The surface monitor is a client of the sensor fusion process, which is the source of all received target data. Sensor fusion is essentially responsible for presenting seamless coverage of targets tracked by a variety of sensors. It merges the multiple data streams and performs parameter estimation, providing such target attributes as position, velocity, acceleration, heading, altitude, etc. The product of the sensor fusion module is a track report (encapsulated in an object called TrackReport), which contains all received and derived parameters for a target and represents a single sampling event in that target's history. For a full description of the track report see the Sensor Fusion chapter of this document.

The four separate servers operated by the surface monitor are used to drive displays, models, and audible devices. The clients of these servers include the ASDE display process, the Alert Monitor process, and the Light Governor process. Any combination of clients (including multiple instances of them) may be present when the surface monitor runs.

The simplest service is the status light service. This contains messages that indicate what state a given light on the airport surface should be in, and includes both Runway Entrance Lights and TakeoffHold Lights. The contents of the message are simply a light ID (as read from the safety logic database) and the current state of the light (ON or OFF).

The alert message service contains more information and is slightly more complex. It includes a runway name, an alert ID, up to two target IDs, a priority level, a time, and an operation field. The runway name is self-explanatory. The alert ID is obtained from the safety database and maps to a predetermined text message (e.g., "Warning: arrival conflict. Traffic stopped on runway "). The ID is simply a means of sending smaller messages over the network. The target IDs represent the target(s) involved in the alert. The priority level is not currently used by anyone, but could be used with a device that supported pre-emptive processing. The idea behind this is that a message warning of a relatively benign situation could be interrupted by a message signalling a need for immediate attention. The time is used to define the order the alerts were issued in, and the operation field indicates whether the alert is being added (a new alert) or removed (the situation was resolved).

The approach message service consists of messages about targets determined to be on approach to runways. This is used by the display program to maintain the approach bar, a display enhancement that displays the position of a target on approach relative to the outer marker and the runway threshold. The approach message contains the runway name, the target ID, the position of the target, a drop target indication, the runway status, and the distance of the target from the threshold. The runway name and target ID are as in the alert message. The target position is a point in meters north and east of the airport reference point. The drop indication, when set, indicates that the target is no longer considered as being "on approach." The runway status is one of the following: Primary, Secondary, or Unambiguous. This is determined by the surface monitor when it attempts to figure out which runway the target is approaching (this is not always obvious). If the target could be on approach to more than one runway it "ranks" the possible approaches and labels the most likely as the "Primary" approach. Thus, Primary refers to an ambiguous projection onto the most likely runway. Secondary refers to an ambiguous projection onto a runway other than the primary runway, and Unambiguous refers to a projection where only a single runway is involved. The distance from the runway threshold is measured along the target path. For targets on approach, this could be a straight line, a curve with a single arc, or an S-curve. This determination is done by the surface monitor, which provides the length so the client can accurately display a distance from the threshold.

The final service provided by the surface monitor contains a target "snapshot." This was developed as a diagnostic tool and has proven to be very valuable in understanding and demonstrating the surface monitor operation. A target snapshot contains a summary of the surface monitor's prediction of target behavior. As will be described in detail below, the surface monitor estimates target state and performs

prediction of where the target will be some *t* seconds into the future. Actually two predictions are made: one in response to the question "Where could the target be?" (the maximum possible path), and the other in response to "Where must the target be?" (the minimum possible path). The target snapshot contains the target state (e.g., STP, TAX, DEP, etc.) and two "trees" denoting the minimum and maximum possible paths. These are used by the display process to graphically depict what the surface monitor is doing.

7.7.2 Architecture Overview

The surface monitor is an object-oriented application. It consists of a main program and some 60 native classes, totaling approximately 28000 lines of C++ code. The application currently runs on Apollo, HP, and Sun platforms; to date porting it has simply required compiling it on the target machine.

The surface monitor accepts target reports from the Sensor Fusion module and produces status light commands, alert messages, and approach bar messages. This is accomplished by the interaction of the software objects, which are abstractions of real-world objects and concepts. Examples of software objects are TakeoffHoldLight, Runway, and AlertMessage. In this document the generic real-world object will be lower case, while the software construct will be capitalized. For example, a reference to a target will mean some aircraft or vehicle operating on the airport surface, while a reference to a Target will mean the software construct and its associated state variables and methods. An effort has been made to name the objects meaningfully; often the abstraction they represent is obvious from the name.

In this section a brief, high-level description of the events that occur when a Track Report is received from the Sensor Fusion module is given. Up to a point these events are ordered and the invocation of certain functions is predictable. After this the events that occur do so as a result of the target's predicted behavior, and it is no longer possible to list them "in order of occurrence." Certain of the objects used by the surface monitor are named here with little or no explanation. A detailed description of each one will be presented in the next section.

The surface monitor maintains a table of Target objects, each one corresponding to a unique target track as determined by the Sensor Fusion module. Each time a Track Report is received, the Target is updated with the Track Report (if no Target exists for that Track ID a new one is created).

The Target first locates itself on the airport surface, determining in which PropagationCell(s) it resides. (As will be explained, the airport movement area and approach areas are blanketed with a grid of irregular polygons, each of which marks the boundaries of a PropagationCell. As part of its setup procedure, the surface monitor determines the connectivity of these cells by building a directed graph.) The locating is performed with the help of a CellLocator object, which contains rules about how the search of PropagationCells should be ordered.

After a target has located itself, it determines its current TargetState using a state machine embedded in the Target object. (A TargetState is one of eight specific operational conditions that any Target must be in: STP, TAX, DEP, etc. Refer to the Safety Logic chapter and appendix for a detailed description of the state transition rules.) Before entering the state machine, however, it must determine if any state transition thresholds will be locally overridden by a high-level object with an interest in the target. For example, the default speed value that must be exceeded for transition from TAX to DEP is 50

kts. This prevents many hi-speed vehicles from being declared "departures" all over the airport. On active runways, however, where departures are *expected*, the value is reset to 37 kts. This threshold is thus *overridden* by the active runway. (A priority scheme is assigned to all high-level objects that might seek to override thresholds, since at some points on the surface two or three contending objects will each attempt to do so.) The mechanism by which this is accomplished will be described below. The state machine is then invoked and the target TargetState is determined.

Using its new TargetState, the Target then loads a set of performance parameters that will be used in predicting its behavior. These TargetState-dependent values include acceleration and deceleration, velocity thresholds, and lookahead time to determine the prediction time horizon.

The Target then *inserts* itself into the PropagationCell in which it is located. This causes the PropagationCell to initiate the prediction process, which proceeds in a breadth-first manner through the directed graph of cells until the edge of the graph is reached or the prediction lookahead time is exceeded. Briefly, (the prediction process is described in detail below) each PropagationCell participating in the prediction process locally stores information concerning the Target's path through itself. It also asks the Target to add it to the list of cells that the Target is predicted to traverse (the TargetPath). When the prediction process is complete, control is returned to the Target object.

The Target then requests that each PropagationCell contained in its TargetPath *notify* any high-level client objects about the new prediction information. This notification could take the form of a "removal" message (i.e., the Target is no longer predicted through this cell and the cell is being removed from the TargetPath) or an "update" (indicates new or potentially altered prediction information is contained in the cell). There is a simple reason for waiting until prediction is complete to perform notification, rather than cell by cell as it progresses: a high-level object might be notified many times (by many different cells) of the target's presence. By waiting until prediction is finished, the entire TargetPath is placed at the disposal of the object, which can then perform its task on the first instance of notification and disregard any further notifications arising from that particular prediction.

The notification process is what causes the safety logic rules to be invoked, and the Target, TargetPath, and PropagationCells contain the information used by the rules. Currently, the following high-level objects register with the PropagationCells that they overlap for notification: Runways, TakeoffHoldLights (THLs), RunwayEntranceLights (RELs), and ArrivalAlerts.

The Runway object has special status among notification client objects in that it must be notified before any others. This is because the remaining objects may need to query the Runway about the status of the Target relative to that runway. For example, a THL may want to know if a Target "satisfies REL conditions" for a particular runway. This is a temporal dependency that is not well documented - the Runways are guaranteed to be created and registered before any other objects because of the order in which the objects are initialized.

Each high-level "notifee" invokes its own set of logic rules. For example, a REL notified of a Target prediction will check if the target is along the runway it is monitoring. It will also use the TargetPath to compute when the Target is predicted to enter and exit its area of interest (its "ActivationArea"). Based on these and other checks, it decides what its state should be and sets itself

appropriately. It then ignores subsequent notification events generated by that Target for that Track Report. Similar action is performed by the THLs, ArrivalAlerts, and Runways.

The high-level objects themselves are not completely independent of each other. For example, Runways may unconditionally set the state of a REL to "ON" when certain conditions are met (i.e., the Whole Runway Switch is on - see Safety Logic chapter). To handle these interdependencies some objects will register with each other. In fact, it is usually true that objects overlapping runways will register with the Runway as well as the PropagationCells from which notification is desired. There is another reason for this: Runways provide a convenient and natural way to set airport-configuration dependent parameter values. For example, under VFR conditions some constraints may be relaxed on active runways. Setting these parameters "for the entire runway" is easier and more intuitive than doing so for each individual object.

7.7.3 Surface Monitor Objects

7.7.3.1 High-Level Objects

There are approximately 60 classes of objects defined and used by the surface monitor. Perhaps another 100 or so "utility" classes are present as well, making a complete class diagram an unwieldy document to decipher. Fortunately, to gain an adequate understanding of how the surface monitor operates less than a dozen of these classes need be examined in any detail. Most of these classes have been mentioned in the above operational summary; a full list appears here. Each of these classes will be described in some detail in this section. The high-level object classes used by the surface monitor are as follows:

RunwayEntranceLight	TakeoffHoldLight	Runway
Target	PropagationCell	SurfaceGraph
ExitPoint	CellLocator	ArrivalAlert

The collection of objects that performs target path prediction and notification is sometimes referred to as the Prediction Engine. This collection includes PropagationCells, ExitPoints, the CellLocator, and the SurfaceGraph. The Prediction Engine generates the events that cause the higher-level objects to invoke the safety logic. Additionally, these "clients" of the Prediction Engine can query the PropagationCells to obtain further information about the target's predicted path should they require it.

7.7.3.2 Target

The Target class is an abstraction of targets seen in the airport surface environment. One Target object is maintained for each unique track monitored by the Sensor Fusion module. A Target is created upon receipt of a new Track Report and subsequent Track Reports are used to update the Target.

The Target interacts with many other objects and exports a variety of methods to give client objects access to its internal state and to modify that state. Each Target object maintains a list of internal attributes that define its state. This list includes the following:

1. All estimated parametric values supplied by Sensor Fusion (e.g. acceleration, velocity, position, altitude, valid altitude flag, valid velocity flag, etc.)

2. A number of attributes describing its location, including a list of PropagationCells, whether it is unambiguously located (Targets may occupy more than one approach zone PropagationCell simultaneously), whether it is inside the airport region, whether it is "along" a runway, and others. These are derived and sometimes set by external objects via exported Target methods.

3. A number of attributes pertaining to TargetState, including current and previous TargetState, whether it has landed, whether it is dropped or suspended, its initial acquired TargetState, and a list of MarkTags maintained as a convenience for client objects. These tags are used when a client desires to "mark" the particular target for any reason, and remain set until the TargetState transitions from its current value. For an example of this see the TakeoffHoldLight object description. One simple but important state variable is the Target iteration stamp. This is an integer that is initially set to 0 and incremented on receipt of each subsequent Track Report. Client objects use this stamp to determine if the prediction they are being notified of is a new one or one that has already been processed.

4. A list of PropagationCells that form the Target's current predicted path.

5. A TargetPath object containing PathSamplePoints defining the Targets current predicted path. This is only created from item 4 if a TargetSnapShot is requested. A PathSamplePoint is a tuple containing <position, velocity, acceleration, timestamp> information, and a TargetSnapShot is a list of segments defining the target's path, currently exported to the display module to observe the dynamic performance of the system.

6. An optional TargetTrace object. A TargetTrace is a packed summary of the Target's internal state that is recorded to a disk file on each update. If recording is enabled, the most recent TargetTrace is retained to help build the next one. The TargetTrace records differences in Target state from one update to the next, so the current one is always retained.

7. A TransitionValueBlock, which contains and manages the state transition values for each Target. These are initially loaded from the Safety database. There are currently 13 state transition threshold values specified by the Safety Logic. The TransitionValueBlock manages each of these as a 2-item stack, where a stack entry consists of a transition value and a transition priority. Initially the stack is loaded with the default value and priority. Any client object wishing to override a transition value must present a higher priority than the current value. If this happens, and the current value is the default value, then the default is pushed onto the stack and the new value/priority pair becomes the current value. If more than one client object attempts to override the value, the one with the higher priority wins and the loser is discarded. If more than one client with equal priorities attempt to write the same value, the first one wins. (Note that this implicitly favors the Runway object since, as in notification, it will always be the first one solicited.) The stack for each value is reset at entry to the state machine on each update, just before override solicitation. Currently Runways and TakeoffHoldLights will sometimes attempt to override transition values.

   The transition override is initiated as follows: when a high-level object that might want to override a transition value is created, it registers for transition override consideration with each PropagationCell intersecting the area it is interested in. When a Target requests overrides from a PropagationCell, the cell in turn passes the request off to each object that has registered with it. Note that not every registered object will attempt to override the value each time it is asked. Consider a PropagationCell lying on a runway. In this case it will have at least two client objects registered for transition override - one runway in each direction. When a Target travelling along a runway solicits overrides, it will be ignored by the runway going in the opposite direction. Only the runway it is travelling along will respond with an override attempt if it has one.

8. A PerformanceValueBlock containing one entry per TargetState. Each entry contains values for acceleration, deceleration, velocity threshold, and prediction lookahead time. The Target loads the appropriate values after computing its state and before requesting insertion from a PropagationCell. The PropagationCells access these values via exported methods in the Target class.

9. A CellLocator object used to generate a list of PropagationCells in which the Target is located. Although currently cells on the surface of the airport may not overlap, approach area cells may overlap each other and ground cells. The CellLocator contains logic to optimize the search for occupied PropagationCells.

The Target is primarily responsible for locating itself, computing its TargetState, maintaining its path list, and making relevant state information available to requesting client objects. Additionally, it maintains a "marking" service for client objects. The marking service allows an external object to place a state-dependent tag on the target, which is guaranteed to be cleared when the target transitions out of its current state. This service is used by RunwayEntranceLights, TakeoffHoldLights, Runways, and ArrivalAlerts for various reasons. It is a client itself of the CellLocator.

The target state machine is part of the Target object. This is a finite state machine that implements the state transitions described in the Safety Logic chapter. Hysteresis is accomplished by applying different state transition rules to enter and exit a state. The transition thresholds and transition rules will not be described here. The state machine includes a simple cycle check (based on a transition count), and

if a cycle is detected the state does not change. It is believed that cycles have been prevented by transition threshold checks, but this has not been formally shown. This is problematical because several client objects will override transition thresholds. The dependencies between transition thresholds are documented in the Safety Logic chapter (see also Appendix A), and these dependencies are enforced whenever appropriate. For example, if a client lowers the threshold governing transition from STP to TAX (vstop+) the threshold from TAX to STP will be forced to a value less than this. These constraints are automatically enforced in the TransitionValueBlock class, which manages the transition threshold values for the Target.

7.7.3.3 SurfaceGraph

The RSLS safety logic relies on an interconnected network of autonomous PropagationCells to perform target projection. In addition to projection, these cells are responsible for retaining local information about the projection, coordinating state transition threshold overrides, and generating the events that result in status light operations. The network of cells is organized into a directed graph covering the airport movement area and the approach areas. The edges of the graph are represented by ExitPoints.

This section will describe three high-level objects that are closely interrelated: the SurfaceGraph, the PropagationCells, and the ExitPoints. First, a description of surface graph creation will be given, including the roles of the PropagationCells and ExitPoints. This will be followed by a discussion of how a target projection is propagated through the graph. Next the internal structure of the cells will be examined and the role of this structure in target projection will be explained. Finally the dynamic behavior of the cell, including target projection initiation and event generation, will be described. A simplified version of a runway area will be used for an example.

7.7.3.3.1 SurfaceGraph

The directed graph of propagation cells is constructed by first creating each cell that will form a node of the graph. This is done by overlaying cell boundary definitions in the form of polygons on top of a graph of the runway and taxiway centerlines. The runway and taxiway definitions reside in the airport surface database, and the cell polygons in the safety logic database. Figures 7-7-1 through 7-7-4 show this process.

Figure 7-7-1 depicts the approach end of a runway and some taxiways that intersect it as it might be found in the airport surface database. Figure 7-7-2 shows the overlay of the cell polygons on top of this structure. The intersection points and directions of the intersection of each centerline with all polygons are found (Figure 7-7-3). These points form the "edges" of the directed graph that connect the individual cells (Figure 7-7-4); each one will become an ExitPoint. Note that each surface cell is connected to its neighbors by two edges (one in each direction), while the cell representing the approach area is connected only by a single edge. An approach area is simply a PropagationCell covering the area that a target will pass through while on approach to a runway. The direction of an edge determines the flow of target

projection, so this indicates that target projections can travel in either direction on the surface, but targets can only be projected out of an approach cell. Note also that the surface cells do not overlap. While the surface cells are not allowed to overlap, approach-area cells may overlap surface cells and each other. Both the target location algorithm (which is closely coupled to the surface graph) and the target projection mechanism take this into account, and this is why a target may be considered "on approach" to more than one runway simultaneously.

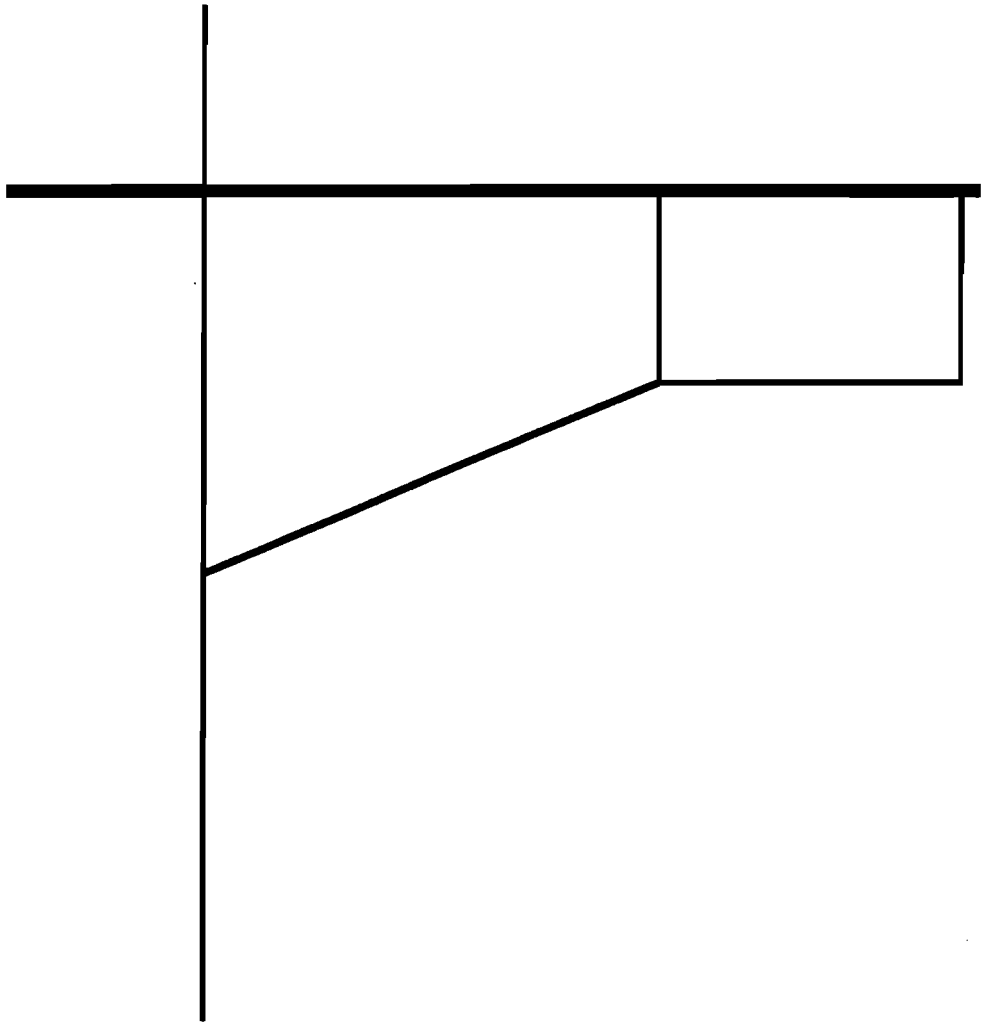


Figure 7-7-1. The approach end of a runway and some taxiways that intersect it as it might be found in the airport surface database.

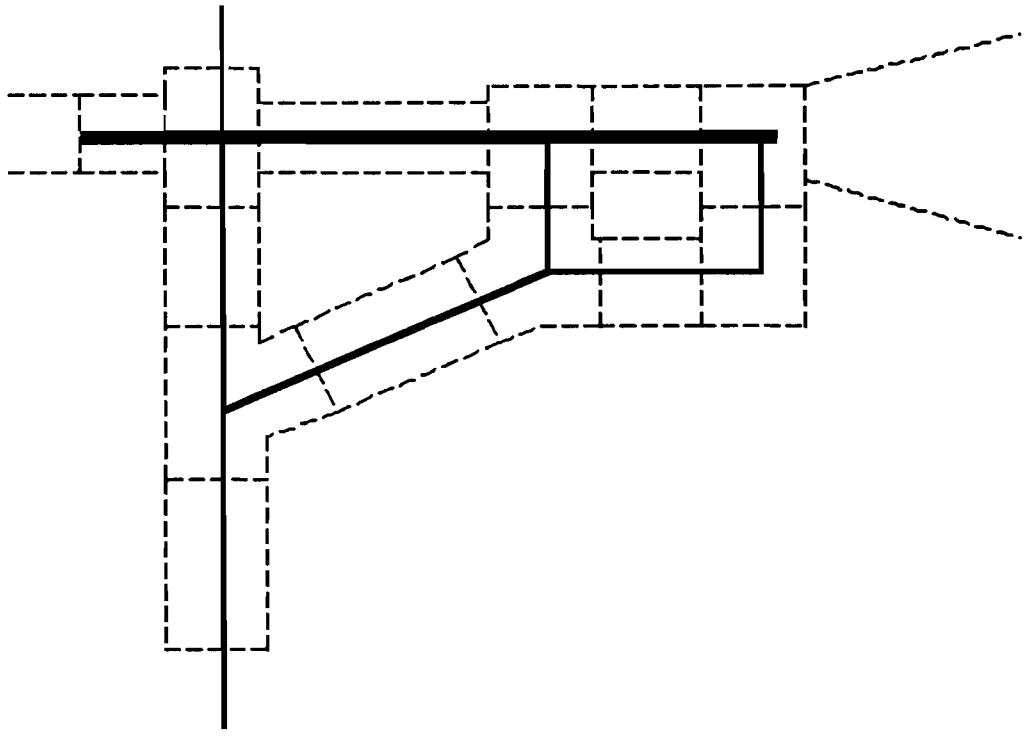


Figure 7-7-2. The overlay of the cell polygons on top of the runway-taxiway structure.

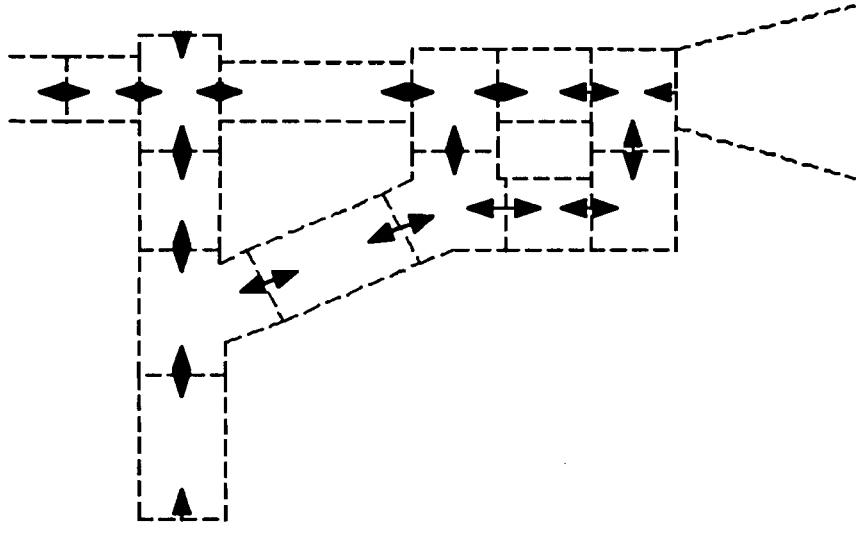


Figure 7-7-3. Intersection points and directions of the intersection of each centerline with all polygons.

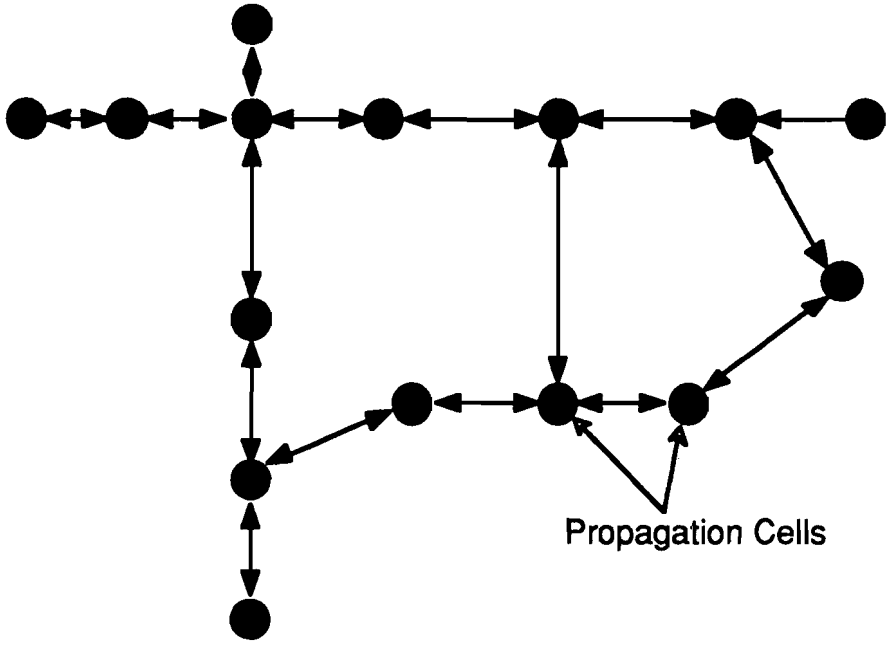


Figure 7-7-4. The "edges" of the directed graph that connect the individual cells; each one will become an ExitPoint.

Each PropagationCell (node) in the graph is completely independent of its neighbors. They share no information, and the only knowledge any cell possesses about any neighbor is that it exists. Target projection can be initiated by any node in the graph.

When the projection starts, the cell creates a "token" containing target performance data derived from the state-based performance model. The cell then determines which of its neighbors the target could reach, and for each of these reachable neighbors it makes a copy of the target token, updates it to reflect state changes incurred during propagation through itself, and hands it off to the neighbor. Cells receiving a target token repeat this process and pass on tokens to their neighbors. This continues until propagation is no longer possible, which typically occurs when either the propagation lookahead time is exceeded or the edge of the graph is reached. The propagation is done in a straightforward breadth-first manner, and no attempt is currently made to order the propagation successor nodes according to earliest handoff time.

7.7.3.3.2 PropagationCells and ExitPoints

To see how each cell performs propagation it is necessary to examine the internal cell structure. Recall from Figure 7-7-3 that, for each cell, a list of (intersection point, direction) pairs was computed based on the runway and taxiway centerline intersections with the cell boundary. These pairs are referred to as ExitPoints. Figure 7-7-5 shows a cell with its associated ExitPoints. For each of its ExitPoints, the cell computes and stores a path to every other ExitPoint. If possible, the path is a straight line. If not, the path is an arc of largest possible radius (subject to some constraints, such as staying within the cell and

not getting too close to any vertex). Figure 7-7-5 also depicts the cell with connections drawn from one ExitPoint (labeled A) to all the others.

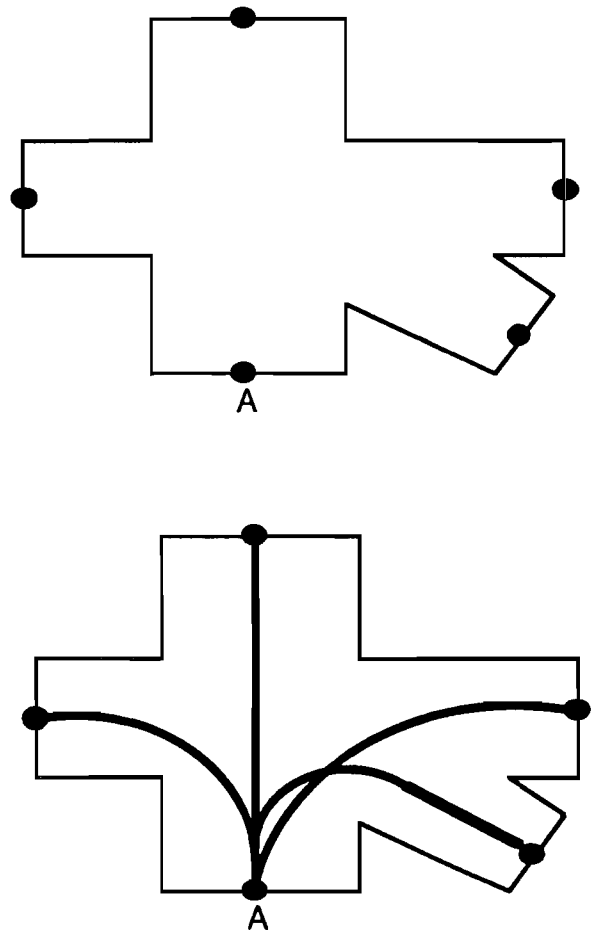


Figure 7-7-5. Cell with its associated ExitPoints, with connections drawn from one ExitPoint (labeled A) to all the others.

These pre-computed paths are used by the cell to propagate targets (actually target tokens) through themselves and on to neighboring cells. When a cell receives a token for propagation, that token will be associated with one of the cell's ExitPoints (i.e., it will enter the cell through that ExitPoint). The cell then propagates the token along each of the paths from that ExitPoint to its other ExitPoints. For each of the paths that the token would exit (i.e., propagation is not finished) the associated neighboring cell is loaded

into a queue. The cell stops processing when it has propagated along all of its paths, and the propagation process stops when the queue of cells is empty.

Propagation along individual paths is straightforward; the token contains values to be used for start time, lookahead time remaining, velocity, acceleration, and various other target performance characteristics. The initial velocity value contained in the token is the target's current estimated velocity. The other values (lookahead time, acceleration, deceleration, etc.) are taken from the target state-based performance model described earlier. These later values are taken from safety logic database. As propagation proceeds and new copies of the token are generated, token values are updated accordingly by each propagation cell for each path it projects along. These are used to compute how far along the path the target would move, and if the target would reach the ExitPoint, the token is updated to be passed to the neighboring cell.

For straight-line paths the computation is basic Newtonian mechanics. For curved paths the target is first checked to determine if it is going too fast to make the turn, based on the target's allowable transverse acceleration (a target parameter in the safety logic database). For a target to negotiate the turn, it must satisfy the constraint:

max speed <= sqrt(transverse_accel * arc_radius)

If the target is travelling too fast, its remaining propagation time is checked to see if it has time to slow down enough to make the turn (it must have a deceleration specified in its performance model). If it can slow down, its speed is set to "max speed" and its lookahead time is docked by the amount of time it would take to slow down. If the target could not slow down, the path is rejected as a candidate for propagation. The target is projected "around" the curve at a constant velocity and is otherwise handled exactly like its straight-line counterpart.

There are two important loose ends to clear up. First, how does the propagation process start? Since the target cannot be expected to start from an ExitPoint, there must be a way to initiate the process from some arbitrary position within the cell. Second, approach-area cells, because they cover much greater area, must have path models that allow for greater freedom of movement than can be provided by a straight-line and a single-arc curve.

The solution for propagation initiation turns out to be very simple. When told to begin propagation, the cell treats the target position and heading as an ExitPoint and constructs paths to all other ExitPoints. It then creates a new target token and propagates exactly as if it had received the target token from one of its neighbors.

Since an approach-area cell will never receive a token from a neighboring cell (recall that propagation is not allowed into these cells), it will always be a site for propagation initiation. In addition to straight-line and single-arc curves, the approach area cell also allows S-curve paths from arbitrary positions within itself to its ExitPoint. The S-curve is constructed by constraining the first curve to be tangent to the target position with a radius equal to the minimum turning radius. This again is dependent

on the maximum allowable transverse acceleration (airborne and surface transverse acceleration are independently specified in the target model), and so the radius is given by:

min_radius = | speed**2 / transverse_accel |

The second curve of the S-curve is constrained to be tangent to the ExitPoint and the first curve. The second curve is computed and if its radius is at least that of the minimum turning curve, then the curve is valid and can be used for propagation. The rest of the details are similar to those of the surface cells.

The propagation cell maintains a record of each target path that encounters it. It keeps a sequence of path "sample points" that contain projected position, velocity, acceleration and timestamp information for both the minimum and maximum path projections. A small summary is additionally kept tracking the targets earliest and latest arrival and exit projections for each cell. Note that this information is not gathered centrally anywhere; it remains distributed over the surface graph in the individual cells.

If a higher-level logic client (e.g., a runway-status light) requests it, the summary information kept in each cell will be gathered into a single data structure and presented to the logic. An example of this is the target "snapshot" service described in the "Surface Monitor Products" section. The resulting display shows two trees, rooted at the target position and spreading out over the movement area along taxiways and runways. One tree represents those paths and distances that the target *could* take in the prediction time using the acceleration model subject to the constraints imposed by the performance model for that target's state. The second tree is made up of paths and distances the target must travel in the prediction time using the braking model; one of the paths from the root to a leaf *must* be traversed by the target in that time (again, according to the target's initial state and its performance model). The higher-level logic has been designed to pose its questions in terms of these paths (where and how far *could* it travel and where and how far *must* it travel) rather than in terms of where *will* it be.

7.7.3.3.3 Client Notification and Event Generation

A major responsibility of the PropagationCells is to notify selected client objects whenever a target path is created, updated, or destroyed. These clients are typically the runways, takeoff-hold lights, and runway-status lights. Recall in Figure 7-7-1 that the cell boundaries were overlaid on the airport surface map to form the surface graph. The clients "register" with PropagationCells in a similar manner. Areas of interest to clients are described as polygons in the safety database. These polygons are overlaid on top of the surface graph, and the client registers with each cell whose boundary falls under the area of the polygon.

An example of such a client might be a runway-status light. Using a polygon, it defines to the surface graph which area it is interested in (i.e., it wants to know about any targets in or projected to be in the specified area). Figure 7-7-6 shows a polygon defining the "activation region" of a runway status light, and the shaded "extracted" cells indicate which cells the light will register with to be kept apprised of projection events.

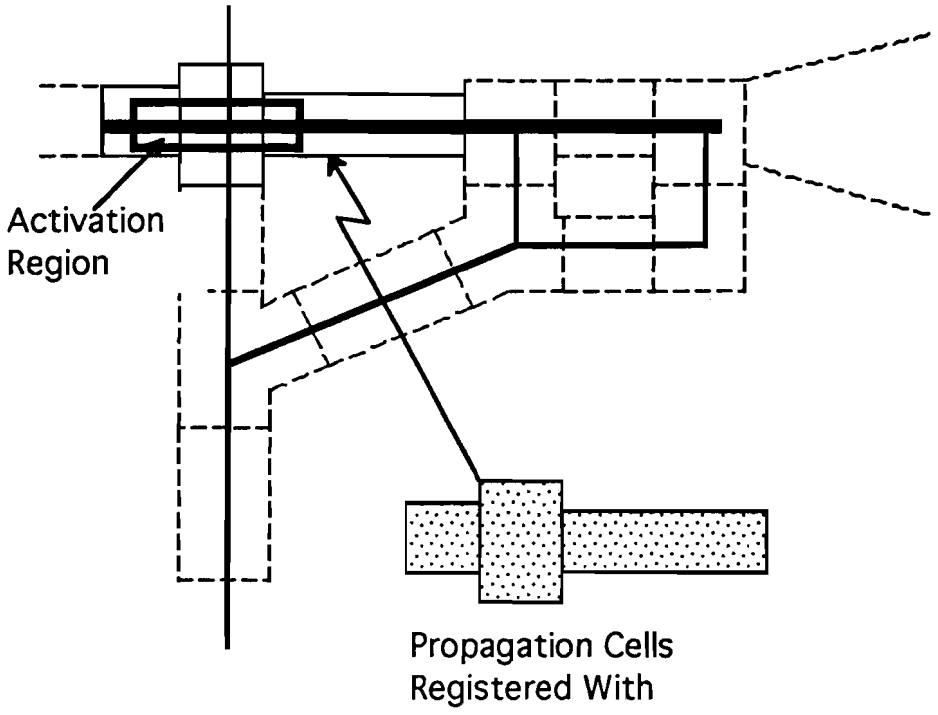


Figure 7-7-6. Polygon defining the "activation region" of a runway status light, with the shaded "extracted" cells indicating which cells the light will register with to be kept apprised of projection events.

Whenever a projected target path is updated within a cell it "notifies" all of its clients of this event and provides them information regarding the target. In this manner the clients get information about all targets in or projected to be in their area of interest. They also have access to the local information kept by the cells they have registered with so they are able to derive accurate local information about the projection for their own means.

7.7.3.3.4 Summary of Target Projection

The projection scheme can be summarized as follows: a target is determined to be in a particular PropagationCell, which creates paths from that position to its existing ExitPoints and propagates it to them if possible. The tokens are then delivered to each neighboring cell, which in turn propagate them further. Finally, each cell whose internal state has changed as result of the propagation notifies any clients that have registered with it, causing higher-level light or alert logic to be invoked then.

7.7.3.5 RunwayEntranceLight (REL)

The RunwayEntranceLight (REL) is an abstraction of a runway-entrance light. To the surface monitor, RELs are specified in the safety database by a polygon that defines the area to be monitored, and a parameter block specifying settings for various safety logic options. (This polygon is known as the

REL ActivationArea.) The portion of the runway defined by the intersection of the polygon with the centerline will be monitored by the light. Note that there is no description of where the light is located physically, or even how many lights there are for a single activation area; although this information is contained in the safety database, it is of no concern to the surface monitor. Also stored in the database are the runways that the REL monitors, and its default suppressed state. The suppressed state indicates whether the light will actually be used. For example, in most configurations RELs at runway-runway intersections are suppressed. Occasionally this may be overridden when, in certain configurations, a runway facing a light will routinely be used as a taxiway.

Each REL monitors a portion of physical pavement belonging to two runways (one in each direction, such as 4R and 22L at Logan Airport). When the REL is created, it searches the SurfaceGraph for all PropagationCells that it overlaps. It registers for notification with each one, and registers itself with each Runway it is monitoring.

Operationally, each REL maintains two lists of ActivatingTargets. If either of the lists is non-empty the light puts itself in the "ON" state; it is off only if both lists are empty. The first list manages targets whose projected paths pierce the REL ActivationArea. Whenever a target's path is predicted to intersect a PropagationCell with which an REL has registered, the REL is notified by the cell. The REL then computes the times at which the path is predicted to enter and exit its ActivationArea (if any). Using these "earliest entry" and "latest departure" times it applies the appropriate rules for that Target's TargetState. An ActivationTarget is added or removed from the first list at this point as the outcome of the tests dictates.

The contents of the second list are managed by the Runway object (this is one reason the REL registers with the Runway). In some cases the Runway will determine that the REL should be activated even though a predicted target path doesn't come near the REL's ActivationArea, for example when the "whole-runway switch" is set and a high-speed operation is in progress. The Runway will instruct the REL to add the Target to its activation list using a method exported by the REL; similarly the Runway must remove the Target when it no longer feels the light should be activated.

Whenever an operation is performed on either of the REL's activation lists the light state is updated. Note that more than one notification may result from a single target path prediction if a REL registers with multiple PropagationCells along a runway. Each time the REL encounters a new Target iteration, it sets a "CurrentProcessedTarget" internal variable to that Target. The first such notification invokes the logic. Subsequent notifications are ignored. This scheme is used by all notification clients to reduce processing overhead.

The REL uses the Target marking service to tag targets to be ignored when they exceed an altitude threshold (only if this option is enabled). This allows the REL to check quickly if the notifying Target is a candidate for activation. Note that altitude is ignored after the condition is met once - since it is volatile and not always available, this represents a severe form of hysteresis. A Target that is marked must transition out of the DEP state before it again becomes a candidate for activation. Like the Target, the REL has trace recording and cross-reference recording options available.

7.7.3.6 TakeoffHoldLight (THL)

Like the REL, the TakeoffHoldLight (THL) class is an abstraction of the takeoff-hold lights on the airport runways. Since the THL state is driven by two-target interactions, it is slightly more complex than the REL. Each THL is defined in the safety-logic-parameter database as a parameter block similar to the one for the REL, and two polygons. The polygons are used to define two regions monitored by the THL, the ActivationArea and the ArmingArea. Two lists are maintained by the THL - one for each area. The light is in the "ON" state only if both lists are non-empty.

The ArmingArea class monitors the region in which targets are expected to hold in position for takeoff. It registers with all PropagationCells that it overlaps, and upon notification of Target activity in these cells it applies logic to determine if the Target belongs on the arming list.

The ActivationArea class manages the activation list in a similar fashion. It defines an area that must be clear of Targets (or must be anticipated to be clear of Targets), and registers with all PropagationCells overlapping this area. Whenever notification is received, it uses its rules to determine if the Target should be placed on the activation list.

The ActivationArea uses special logic for crossing runways, since high-speed crossing targets on runways demand special logic for assumed separation. When the ActivationArea is created, it checks the surface database to find all intersecting runways. For each runway it creates a XrwyMonitor (crossing runway monitor) object whose responsibility it is to decide if high-speed crossing targets belong on the activation list. This logic is described in the Safety Logic outline, and is invoked by using a feature of the notification mechanism that allows objects registering for notification to supply a local object that will also be delivered with that notification. Effectively, this means that when the ActivationArea registers for notification with the PropagationCells that also overlap crossing runways, it supplies the XrwyMonitor as such a local object. Whenever the ActivationArea receives notification from these PropagationCells, the XrwyMonitor is returned as well. In this manner the ActivationArea knows when to invoke the "crossing runway" logic - it does not do so for all notifications.

THLs also make use of the "marking" service provided by the Target class, which was described earlier. This is used in VFR conditions to cause a Target to remain off the activation list when certain conditions are met. Basically, when this option is in use (VFR), and the Target is reported in the DEP state and has reached a certain altitude, it will no longer be considered an activating target (i.e., the THL will be turned off if this was the only target currently on the activation list). Since altitude coverage is undependable and volatile, this condition might be met and not met on subsequent update iterations. The marking service provides a quick and efficient way to deal with this. The THL "marks" the Target with its own identifier. On subsequent iterations, it checks to see if it has previously marked the Target - if it has then the Target is ignored. The "mark" is guaranteed to be cleared by the Target when it changes state. Thus, after the target has been marked, as long as it remains in the DEP state it will not activate the light. Should it transition to DABT or UNK the mark would be cleared, and on that iteration the THL logic would again be invoked to determine if it should be considered an activating target.

The THL also registers for transition-threshold-override consideration if this option is enabled in the database or one of the configuration-dependent overlay files. When this is in effect, it will normally compete with the Runway to override the threshold values that govern the TAX to DEP transition. The

default "priority" associated with the THL override has a value of 2, which gives it precedence over the Runway priority (which is 1). The intuitive reasoning behind this scheme is that 1) setting high threshold values in general will inhibit false declarations of departures at various points on the airport surface, 2) on active runways, this value is lowered since it is reasonable to find departures there and the active runways are likely to be clear of incidental traffic, and 3) in arming regions on active runways departures are especially likely to occur, so we would like the transition to be particularly sensitive in these regions.

The THL also registers with the Runway it is monitoring. It does this primarily to allow the Runway to set any configuration-dependent default parameters that are present.

7.7.3.7 ArrivalAlert

The ArrivalAlert class is intended to manage all types of alerts involving Targets in the arrival, landing, and landing-abort states. Currently, the only alert implemented is for a target stopped on a runway when another target is arriving or landing. Additionally, all alerts are two-target alerts. There are no alerts defined for groups of three or more targets.

The ArrivalAlert is defined by an activation region that is monitored for stopped targets, a parameter block containing thresholds and options, and a runway designation indicating which runway the ArrivalAlert services. The ArrivalAlert registers with each PropagationCell overlapping its activation region, and with the PropagationCell covering the approach zone of the monitored runway. It also registers with the Runway object, but this is only to allow the Runway to set runtime options such as whether to issue an alert if the arriving target is projected onto more than one runway (i.e., it is ambiguously projected). The ambiguous projection option allows alerts to be suppressed if the target on approach is projected onto more than one runway (i.e., it occupies and is projectable in more than one approach PropagationCell).

The ArrivalAlert maintains two lists of targets: one contains targets in the arrival, landing, or landing-abort state. The second list contains all other monitored targets, which is defined as any target that is interacting with the activation region and satisfies some state-dependent monitoring criteria. For example, since the current implementation only supports monitoring targets in the stopped state, the current criteria are as follows: 1) the target must be in the activation region, 2) the target must be stopped. All other targets in the activation region are ignored (as far as this list goes - they will still be contained in the first list if they are somewhere in the landing process).

A list of current Alerts is maintained to allow the ArrivalAlert to determine which alerts it has issued messages for, and to know when the alert condition no longer exists. When for one reason or another the alert condition is resolved, the ArrivalAlert sends a message to the AlertGovernor process and removes the alert from the active alert list.

Two mutually exclusive algorithms are currently supported - the "threshold" model and the "target" model. The threshold model seeks to issue an alert when an arrival target is within some t-second range of crossing the runway threshold and there is some stopped target in the activation area. In this model, it does not matter where in the activation region the monitored target is, only that it is inside and stopped.

The same lists of "arrival" targets and monitored targets are maintained, and the "time-to-threshold" computation is updated for each arrival target whenever a TrackReport is received. The ArrivalAlert queries the Runway when it is created to get the displaced threshold distance, if any (this can be configuration dependent).

The "target" model algorithm causes alerts to be issued if the separation in time between an arrival target and a monitored target descends below some threshold. In this model, for example, it would be possible for a target to be stopped at the far end of a runway and no alert would be issued. As opposed to the threshold model, the target model maintains separations between individual pairs of arrival targets and monitored targets. The t-second thresholds are position dependent, and are specified as part of the ArrivalAlert data block in the safety logic database. This allows specification of a variable threshold profile to fit operational needs. For example, it may be desirable to have a long threshold at intersections where takeoffs are typically performed, and a shorter one at intersections where crossings predominate. When an update is received for a monitored target in this model, its new threshold time is determined and stored with the target. When an arrival target update is received, the ArrivalAlert computes its estimated separation (in time) between it and each monitored target. For each monitored target, an alert is issued if the estimated separation is less than the threshold.

Both algorithms take into account the possibility that targets on approach often have curved paths. For this reason, the ArrivalAlert queries the approach PropagationCell directly to get the target's estimated time to the cell boundary. This automatically uses whatever path the cell generated when it performed the target traversal computation; as described above this could be a straight line, a single arc, or an S-curve. From the approach cell boundary to the target on the runway a straight-line path is assumed and a constant velocity prediction model is used.

7.7.3.8 Runway

Like the status lights and alerts, the Runway is a client of the Prediction Engine. It differs from other clients, however, in that it has clients of its own. The RELs, THLs, and ArrivalAlerts all query the Runway for status information regarding targets as part of their logic. The need to service these requests puts the Runway in a position somewhere between the Prediction Engine and the client objects in the surface monitor architecture.

Since high-speed operations on the airport surface normally are associated with runways, it is natural that the Runway object plays a central role in coordinating information dispensed to other objects. Status lights, alerts, and approach bars have an intuitive relation with runways. Indeed, when discussing individual lights, alerts, or even operations, it is natural to describe them in the context of "what runway is involved." These relationships are reflected in the many responsibilities and associations that the Runway maintains.

The Runway's activities can be partitioned into three loose areas: its role in the setup and initialization of the surface monitor, its dynamic interaction with Targets it is notified of, and its dynamic interaction with other client objects.

Since the Runway is naturally associated with sets of client objects, it is intuitive that some operational parameters be set using the Runway as opposed to individually setting them for client objects. Examples of these include the VFR switch (controlling whether Visual Flight Rules are in effect), the ambiguous projection switch (controlling whether status lights will be activated for approach targets projected onto more than one runway). The Runway maintains knowledge of the following high-level objects:

1) A list of RELs
2) A list of THLs
3) A list of PropagationCells
4) The PropagationCell representing the approach area
5) An ArrivalAlert servicing that runway.

The Runway maintains a target record for each Target it monitors. This record contains derived information describing the state of the target relative to the runway. For example, whether the target is on approach to the runway, how far the target is from the runway threshold, whether the target is currently activating any status lights, and the direction of the target relative to the runway (along, opposite, crossing) are all included in this record. This record is updated when the Runway receives notification from a PropagationCell about a target path, and is used to answer subsequent queries from other client objects.

Prior to any prediction and notification, Runways interact with Targets through the state-transition-override process. Runways may override thresholds governing transitions from taxi to departure and from landing to landing abort. The taxi to departure threshold is constant for the runway and can be set in the safety logic database. For the landing abort threshold, the runway maintains a landing velocity profile from which it selects a position-dependent value.

At this point (transition threshold override) a circular dependency in the safety logic is addressed. When notified of target activity, the Runway determines whether a Target is travelling "along" it (see the Safety Logic Appendix for the exact definition of "along"). This information is used by other prediction engine clients, who will query the runway to find the targets "Runway status." A problem arises because certain state transitions are effected by whether a target is "along" a runway, and state determination is performed before any prediction and notification (hence, Runway "along" determination). To get around this, the Runway will advise the Target if it is "along" the Runway at the same time it is attempting to set state transition parameters, not waiting until prediction is performed.

Runways are also the source of ApproachMessages, the contents of which were described in the section on Surface Monitor Products. The approach message logic is invoked whenever a target notification is received from the approach PropagationCell and the target satisfies the approach bar conditions as described in the Safety Logic Outline (see Appendix).

Runways interact with RELs in a direct way. The surface monitor supports a "whole runway switch" option that, when invoked, allows all the RELs along a runway to be illuminated for departing targets and arriving targets once the target crosses the threshold. This is done regardless of the target's predicted path, and is strictly a procedural rule. This rule is in effect until the target reaches a "landing

rollout" state (another state computed and maintained locally by the Runway - the Target has no such state in its own state machine), at which point the RELs are again controlled strictly by the Target's predicted path.

The whole runway switch logic makes use of the REL feature that allows an external object to unconditionally add a target to one of its activation lists. (Recall that the REL maintains two lists - one that it controls and one controlled externally that it monitors.) When notification is received about a target in a landing or arrival state, the Runway determines which RELs along it should be illuminated by the whole runway switch rules. For each such REL, the target is added to that REL's external activation list. This may result in duplicate entries for that particular REL and target, but that makes no difference to the REL. When the target is determined to be under control, the target is removed from all RELs who currently maintain it on their external list, leaving only those RELs continuing to maintain it on their internal list by virtue of its predicted path.

Under VFR conditions, the whole runway switch can also be deactivated when a departing target attains a certain altitude. To support this the Runway makes use of the Target marking service described earlier.

The RELs, THLs, and ArrivalAlerts are all clients of the Runway; they are all interested in whether a given target "satisfies the REL conditions" for that runway. These conditions are defined in the Safety Logic chapter and appendix, and are used by these objects to help determine if the target qualifies to be on an activation list or not.

7.7.4 Surface Monitor Databases

7.7.4.1 Overview

As part of its initialization procedure, the surface monitor uses information from two airport-specific databases: the airport surface database and the safety-logic-parameter database. The airport surface database essentially contains a graph of the runway and taxiway centerlines and details their interconnections. The safety-logic-parameter database is used to define and configure the software constructs used by the RSLS surface monitor (the safety logic process). The contents of these two databases are used to construct the directed graph that is used for target propagation and event generation and to create and initialize the higher-level objects such as Runway Status Lights and Takeoff-Hold Lights.

7.7.4.2 The Airport Surface Database

The Airport Surface database contains the positions of the centerlines of runways and taxiways, and the airport origin. All coordinates are given in latitude/longitude. The connectivity of these constructs is also represented in the database, so segments and points common to more than one runway or taxiway are represented in one place.

The application interface to the surface database allows client programs to load the database and access runways and taxiways either by name or by iteration. A special application, used for both the surface database and the safety logic database, is the surface editor. This is a graphical editor that allows fast and easy (click and drag) manipulation of the contents of both databases (among others). Unfortunately, this application currently does not run in the X Windows environment and must be run on an Apollo platform.

7.7.4.3 The Safety Logic Parameter Database

For each type of safety logic object (e.g., Runway), default values for each of its parameters are explicitly specified in the database. These represent "reasonable" choices for the values in the absence of any airport-specific requirements.

A default parameter-specification block exists for most types of high-level surface monitor objects. These include Target, Runway, REL, THL, and ArrivalAlert. Note that there is no such block for the PropagationCell or any other component of the SurfaceGraph.

The definitions of the safety logic objects typically are composed of two parts. The first part is a spatial component, for example the polygons representing the arming region and activation region for a given takeoff-hold light. The second part is a list of parameters whose value for that particular object differs from the default value.

As an example, consider the "alarm-angle" associated with the arming region of a takeoff-hold light. This parameter defines the angular tolerance relative to the runway heading that a target in the arming region must satisfy to arm the light. The default value of this parameter is 65 degrees, so a target must be heading within +/- 65 degrees of the runway heading to be considered as an "arming" target. Due to the geometry of the airport surface, almost every takeoff-hold light defined for Logan overrides this value. At some places (e.g., departure ends of runways with no crossing taxiways) the angle can be considerably widened. At other places (e.g., some intersection-takeoff arming regions) the angle is narrowed so as not to turn on lights due to heading noise reported from crossing targets.

This combination of default values and object-specific overrides provides the mechanism for "customizing" the safety logic for a particular airport. The default values are selected for general reasonableness and the overrides are only used for specific objects (usually in those cases where airport geometry requires it). Such a parameter set would now be customized for an airport, but not for any particular operational configuration. This is still inadequate for our purposes, since no single set of values can be used for all configurations and operational conditions for a given airport.

To achieve configuration-specific customization, additional parameter-override files called Configuration Overlays are used. A Configuration Overlay file contains parameter values that are used to override either object-specific parameter values or default parameter values (or both), as desired for a particular set of operational conditions. For example, configuration overlays are defined for each of the commonly run configurations at Logan.

As an example, consider one of the state machine parameters that is used to determine when a target has transitioned from taxi to departure. This value is nominally set at 50 kts to avoid declaring fast-moving surface vehicles as departures. Using configuration overlays, this value is overridden on active runways to 35 kts.

7.7.5 Design Goals: Problems and Solutions

Several different and sometimes opposing requirements drove the evolution of the surface monitor architecture. Certain problems were anticipated and designed for; others came as surprises and required more effort to circumvent.

The primary initial requirements were these:

1. The architecture should be flexible in the way it responds to requirements changes. The surface monitor needed to be used as a means of testing different ideas, not implementing already well-tested algorithms. The architecture needed to be able to respond quickly and easily to such changes.

2. The architecture should be site-independent. Site independence was important for a number of reasons. First, site-independent algorithms, if they worked, would be much more applicable, hence valuable, than a scheme that was custom designed for one particular airport. Practically, it would be virtually impossible to support any system that does not have a high degree of site independence built into it.

2a. The site independence should be automated where possible. While this requirement was not so important in the context of our demonstration system, it is extremely important to anyone considering building and fielding a production version of this system. Simply put, wherever possible system attributes should be derived from the geometry of the site and not specified by independent parameters. Where parameters need to be specified, they should have some intuitive relation to real-world objects (e.g., a speed threshold for a particular state). The importance of this is obvious to anyone who has ever tried to validate a system - too many independent parameters render such validation efforts intractable.

3. The architecture should be extensible. It was anticipated that new features would be incorporated into the requirements, and the architecture should be able to accommodate these without revision. Examples of anticipated extensions are as follows: 1) individual target performance models, when exact target type becomes known, and 2) full alert logic in addition to lights.

In addition to these requirements, the following problems were anticipated:

1. Target comparison (for 2 target conflicts especially) is inherently a problem of $O(N^2)$ complexity (i.e., the computational load increases proportionally to the square of the number of targets present). A natural way around this needed to be devised.

2. The parameters would need to be changed from operational configuration to configuration. It was not desirable to maintain separate databases for each configuration.

The adoption of the Prediction Engine approach, in particular the PropagationCell aspect of the Prediction Engine, fundamentally addressed several requirements and problems.

Initially this model was created to solve the O(n**2) problem. Consider a target on approach to a runway. How does one determine which lights should invoke logic to check their activation status? How does one determine which targets need to be checked for potential conflict. Equally important, which targets may safely be ignored?

One solution to this type of question is: let each client object concern itself only with the set of targets that could possibly affect it. Since each client object defines itself in terms of an extent on the airport surface, it would be interested only in those targets in or predicted to be in its extent. By registering with the PropagationCells contained in the extent, the object is notified only when there is a change affecting a target that may interact with it. For example, a TakeoffHoldLight monitors a specific area of the airport surface for two-target conflicts. When a target enters or is predicted to enter the activation region of the TakeoffHoldLight, the object is notified and it already knows the exact set of targets that must be examined to determine if a potential conflict exists (these are taken from the set of targets that have previously notified the light). No other targets are even considered, and if another target is updated and is not predicted to intersect the light's activation region, the light is never even notified.

The prediction engine provided additional design benefits as well. First, issues of target prediction were separated from the safety logic rules. Once a viable model of target prediction was settled on, the logic could (and did) go through many iterations and not impact the prediction part of the system. Also, the computation of target path prediction is made much easier since it is based only on local physical rules and not at all on airport context.

The PropagationCell connectivity and the internal structure of the cells are directly derivable from the airport geometry and the airport map - no human direction is necessary once the cells are drawn. The cell boundaries themselves do not need to be precise. This practically enforces a large degree of site independence, since it is not possible to tailor an algorithm for a particular geometric anomaly without generalizing it to work in the context of an arbitrary PropagationCell.

The PropagationCells also provide an easy way to facilitate communication between other high-level objects. This was used to solve the problem of specifying location-dependent parameters in a way that was natural and intuitive. It did not make sense to allow the cells themselves to override global default parameter values directly since as abstract constructs the cells do not correspond to any real-world entity. Instead, the cells provide the mechanism for connecting groups of client objects that do interact but have no other reason to be aware of each other. As an example, consider the target-state-transition threshold values again. The Target object does not need to know whom to solicit the override values from. Originally, the Runway was the only object that overrode these values. When the decision was made to allow the TakeoffHoldLight to also override some values (some in direct competition with the Runway), no change at all was necessary in the Target object. The new overriding client (the THL) simply had to add its request to be solicited to the cell. When the Target goes to solicit local values, it simply asks the cell if it has anyone on its list.

Extensibility is further enhanced by the generic nature of the client-object notification structure. A general callback architecture is implemented with a Notifier class, a base class that exports a well-defined interface to the notification mechanism. A macro provides a "template" that allows client objects to easily create type-specific class "notifiers," essentially a single object containing the callback entries for the particular client object. The callbacks actually contain the "safety logic" that is germane to the specific high-level object. The process of registering for notification with a PropagationCell consists of providing the cell with an instance of the notifier, which the cell places into a list of Notifier objects. Notification is then simply the invocation of virtual callback functions for each object in the list.

Two separate Notifier classes are maintained. The first supports notification for events caused by target path propagation, and the second supports the state-transition-threshold override function. The mechanics of Notifier creation and invocation is the same for each; they differ only in the interface they export. The Notifier classes make the addition of new client objects (and the functionality they represent) a relatively painless process.

7.7.6 Suggested Improvements

There are several ways in which the current surface monitor could be improved, ranging from minor feature enhancements to some major structural modifications. Some of these items were known from the beginning of the project and were simply not implemented due to time constraints, others were conceived as solutions that were too complicated to justify major reworking of existing code given the point in the project at which they were discovered.

In addition to enhancements, many parts of the code are cluttered with obsolete algorithms and data structures. A policy of "never delete anything" was adopted after some early approaches were discarded and deleted, only to be resurrected and recoded. In addition to clutter, this resulted in a lot of unneeded redundancy regarding stored information. Removing these redundancies alone would contribute markedly to readability.

The emphasis throughout was on correctness of the algorithm. While it was necessary for the code to run without frequently failing, no effort was given to issues of fault tolerance and recoverability. It has been analyzed for memory leakage and general correctness, but no formal approach to robustness was taken.

Likewise, no serious attention was paid to efficiency. While efficiency regarding the $O(N^2)$ problem was a major consideration, attention to details such as memory allocation were ignored. For example, the surface monitor could significantly reduce overhead if its classes supported overloaded new and delete operators. These could remove and return memory from and to an already allocated pool, eliminating the need to perform many small allocations that are done during path prediction. In a similar manner, more efficient search logic could be implemented in the CellLocator. Currently the CellLocator searches PropagationCells in the order of previously predicted target path. This seems to work fine except on the initial hit, where a linear search is performed. One of the original design plans was to perform a quadtree search, but since surface monitor performance was never a bottleneck other tasks took higher priority.

The surface monitor as it is implemented is not a real-time system. The change to a time-interrupt-based system is recommended and could be accomplished without affecting the basic architecture in any major way. This change would allow finer accuracy in predictions as well, since a feedback loop could be implemented wherein the surface monitor requests that Sensor Fusion increase the sampling rate for targets of interest (i.e., on active runways).

Moving to a time-based system would also allow removal of one of the remaining blind spots of the current logic: if a target is dropped and not removed from some client's activation list, it will remain on the list indefinitely. (This is, of course, a logic bug and should never happen, but it has infrequently occurred when a PropagationCell coverage gap existed inside some activation region.) Setting limits on the length of time an object may exist without update is an easy solution to this problem.

If necessary, the prediction engine could be parallelized. The critical sections of code (those modifying the internal data structures) are isolated, and the parallelization could be extended to all the objects. This would be something that could be looked into in the event of a rewrite.

The design goals and the developmental emphasis caused the coupling between classes to be higher than would be necessary for a production system. A version of the surface monitor that is intended to be fielded and maintained in a production would greatly benefit from a simplification and rewrite. The basic architecture and class definitions are viable and extensible. From a software engineering standpoint, the implementation should be redone to emphasize validity, robustness, efficiency, and maintainability.

7.8 THE LIGHT GOVERNOR

7.8.1 Introduction

The Light Governor controls and monitors the light states of the physical light system. In the Logan Demo, this means controlling and monitoring the lights on the Model Board.

Messages containing desired light states are received from the surface monitor and the keyboard. The Light Governor validates the messages, transmits an appropriate command to the Model Board, and then retransmits messages containing updated light states to any clients connected to it. These typically include the ASDE display clients, and may include the surface monitor in the future.

Note that the clients of the Light Governor display the light states as they physically are; if a command should for some reason not be executed by the Model Board, the ASDE display clients' representation will be consistent with the Model Board, even though this may be an "incorrect" state.

7.8.2 The Logan Demo Model Board

The Model Board is a 1:1800 scale model construction of Boston's Logan Airport. The model features operational runway and taxiway centerline lights, wig-wag lights, and RSLS runway-status lights. An API (Applications Programming Interface) for the Light Controller on the Model Board defines commands to read, set, clear, and toggle the state of individual channels. The API also supports defining groups of channels, which can then be set, cleared, and toggled with single commands.

Lights on the Model Board that indicate runway status at the same physical intersection are logically indistinguishable, having the same channel number assigned to them. Thus, when the surface monitor issues a message to change the state of a certain light, all lights with that channel will be set to the desired state. Additionally, it is possible to establish a logical group of channels and assign to the group its own channel number. This allows a large number of lights (for example, a block of runway-status lights on a runway) to be controlled with a single command.

7.8.3 Process Architecture

7.8.3.1 Internal Structure

The Light Governor is responsible for converting light IDs (defined in the safety logic database) it receives from the surface monitor to physical channel numbers on the model board. This mapping removes the surface monitor from any consideration of the physical implementation of a lighting system - in principle lights could be installed on the field and only the Light Governor would need to be modified. A Light Governor database file is used to store the channel/light ID definitions, as well as any channel group definitions.

Internally, a ModelBoardLight object is used to represent each channel. Each of these objects contains the light ID, the corresponding channel, and the current light state. Obviously, some channels

will have no corresponding light ID (e.g., any light that is not a RSLS runway-status light), so for these a dummy light ID is used.

ModelBoardGroup objects are used to represent defined groups of channels. This structure is effectively a list of ModelBoardLight objects along with methods to apply single group commands to each ModelBoardLight in the list.

At initialization a set of ModelBoardLight objects is created, one for each entry in the Light Governor database file. These objects contain the state variable that is modified and served by the Light Governor, and they are accessed by two hash tables created for this purpose. The first hash table is used for accessing the ModelBoardLights when surface monitor messages are received, and the hash key is the light ID string. The second hash table is used for accessing ModelBoardLights when individual channel commands are received from the keyboard; the channel number is the hash key for this string.

A third hash table is created to allow access to any groups that may have been defined.

7.8.3.2 Executing Surface Monitor Commands

When the Light Governor receives a light message from the surface monitor, it first validates the Light ID. It then checks the desired light against the current state maintained in the appropriate ModelBoardLight object. If they differ, it sends the appropriate light command to the Model Board channel referenced by the ModelBoardLight. Should this attempt fail, it will be tried again (a timeout mechanism is employed). If after two tries it is unable to execute the command, a fail status is returned and the ModelBoardLight state remains unchanged. If the command succeeds, the ModelBoardLight state is updated and a light message containing the new light state is sent to clients of the Light Governor.

7.8.3.3 Executing Keyboard Commands

If a keyboard command is received, the Light Governor first validates it as a channel command or a group command. If a channel command is received, the processing steps are the same as for a surface monitor command, with two exceptions: first, the ModelBoardLight is accessed via the channel number hash table, not the light ID hash table. Second, if the light ID of the ModelBoardLight object is the dummy ID, no update is sent to clients after the Model Board command has been executed.

Groups of channels can also exist, either established from initial configuration files or from keyboard commands. When a command to create a group is received, the Light Governor first ensures that all the channels defined in the group have been created. If a command to change the state of a group is received, a group "update" command is issued that will send a single command to the model board, but will update each ModelBoardLight object individually. The appropriate individual light messages are sent to clients in this case, since the definition of a group exists only within the Light Governor and the Model Board.

7.9 THE VOICE GENERATOR

7.9.1 Introduction

The Voice Generator manages the processing of audible alert messages. Alert conditions as defined in the safety database contain a priority and a message ID. The Voice Generator converts the message ID to a text string and sends this to a device that broadcasts it (in the Logan Demo, this is a DECTalk DT100 device).

Since it generally takes longer to audibilize a message than to receive one, a queue of alert messages may build up. These are processed in order of their priority. If the broadcast device supported preemption, and an alert of high priority was received while the device was broadcasting a message of relatively low priority, the low priority broadcast could be interrupted and re-queued for broadcast.

7.9.2 Process Architecture

A Voice Generator database is used to perform the mapping of message ID to text string. This allows the text accompanying an alert condition to be modified without affecting the safety logic in any way. Runway names (if they are present) are contained in the body of the alert message and not in the text string. A hash table keyed on the message ID is used to access the text strings.

The Voice Generator is a client of the surface monitor. It listens for alert messages and processes them as they arrive. It maintains a queue of alert messages ordered by alert priority.

When an alert message arrives, it is examined to determine if it is an ADD operation or a DELETE operation. If it is a DELETE operation (i.e., the situation has somehow been resolved), then the alert is removed from the queue if it has not yet been broadcast. Otherwise nothing is done.

If an ADD operation is requested, the Voice Generator validates the message by looking up the message in the hash table. The runway name, if present, is encoded as part of the message text. The message is then added to the priority queue. As long as the queue is non-empty messages will be sent to the DECTalk as it becomes ready to receive them (it cannot receive messages when it is broadcasting one).

7.10 RSLS DISPLAY

7.10.1 Process Architecture

Both of the RSLS displays (DBRITE-like and ASDE-like) are generated by run-time configurations of the same executable. The display software uses InterViews[1] as its graphics environment and is designed to directly support sub-windowing of the main display window. The software generates a plan view representation of any combination of the following items:

a map (line drawing) with up to five togleable overlays

data tags similar to those on a DBRITE display based on data from the MDBM interface

target icons and tags based on track data from sensor fusion

approach bars showing the distance from the threshold of landing aircraft on a compressed scale based on data from the surface monitor.

highlighting of targets (target icons or approach bar) deemed to be in an alert situation by the surface monitor

the state of the runway status lights

icons for targets detected by the ASDE processing

target snapshots showing the target state and projected paths based on data from the projection engine portion of the surface monitor

the arcs and segments of the surface monitor's surface graph.

The display supports zoom, pan, and rotation independently in the main window and each subwindow and provides an interface to the master clock to facilitate the control of time during playback. There is also a pseudo-pilot interface to a simulated-target generator (a description of which is beyond the scope of this document).

---

[1] InterViews is an object-oriented C++ encapsulation of Xwindows that was created at Stanford University. It completely isolates the application software from the Xlib library, and in principle would allow using the display software in a non-Xwindows environment by solely rewriting a relatively small portion of the InterViews code. In fact, InterViews currently includes a special version of the Canvas that supports generating PostScript printer output.

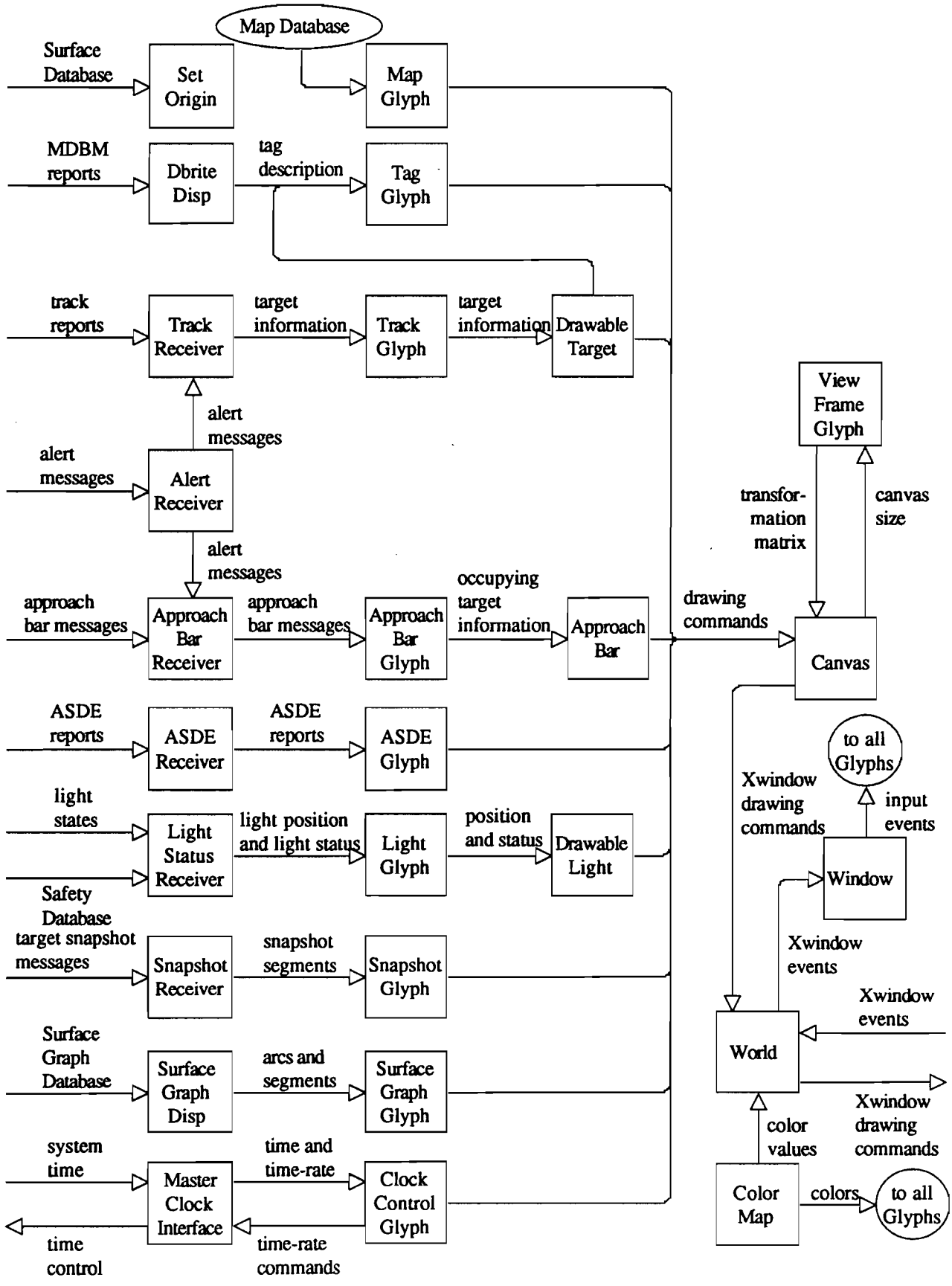


Figure 7-10-1. Display components.

The primary display software components are shown in Figure 7-10-1. These components fall into two classes, infrastructure components and data reception and rendering components. The infrastructure components provide the basic backbone of the display. They include the following:

World -- the InterViews encapsulation of the Xwindows connection to a display server

Window -- the InterViews encapsulation of an Xwindows window

Canvas -- a sub-component of a Window that serves as the InterViews interface to the Xwindows drawing operations

Color Map -- the module that encapsulates the strategy for allocating and sharing graphics bit-planes amongst the various displayed items

ViewFrameGlyph -- the module that controls the transformation matrix of the Canvas to control zoom, pan, and rotation and maintains the list of glyphs shown in the window.

These infrastructure components work together to establish a connection to the Xwindow server, create the main window and sub-windows, define and allocate colors and the color map, receive keyboard and pointer events from the Xwindow server, and deliver the events to the appropriate event handlers. The remaining components in Figure 7-10-1 are responsible for receiving and displaying specific types of information. Typically, they are in receiver and glyph pairs. A single instance of each receiver component accepts the incoming data via a callback from the dispatcher and distributes it to one or more glyph instances. There is a glyph instance for each window in which the data is displayed. An additional component that is not shown in Figure 7-10-1, the dispatcher, controls the flow of processing by monitoring the incoming data streams (including the one from the Xwindow server) and making function callbacks to data stream handlers when data is available. This callback mechanism facilitates the addition of new display components since the display components register with the dispatcher, and thus the only portion of the software that knows of the existence of all the display components is the command-line parser.

7.10.1.1 Infrastructure components

The InterViews World module encapsulates the Xwindows notion of a connection to a display server. It stores the information about the display required in many Xlib calls and serves as a repository for server and application resources such as fonts, colors, and brushes. It also maintains a list of Windows and is responsible for distributing events such as user input, window exposure, or window re-sizing that are received from the display server to the appropriate Window.

The InterViews Window module is an encapsulation of the Xwindows window. It serves two primary functions. First, it maintains a Canvas (described below) which is used to receive drawing instructions and a Glyph that performs the drawing operations on the Canvas. Second, the Window receives events from the display server. Exposure and re-size events cause the Window to tell its Glyph to redraw itself on the Window's Canvas. For user input events, the Window consults a list of registered Listeners to determine if the event is of interest, and if it is of interest, the Window passes the event to the Listener's associated Handler. Typically, Glyphs that accept user input act as both Listeners and Handlers.

The Canvas is an InterViews module that provides a generic interface to the Xlib drawing commands. This allows the application code to be more easily ported to a non-Xwindows environment. The Canvas maintains a transformation matrix that is used to perform the scaling, rotation, and translation of points from 'world' coordinates to screen pixel coordinates. The Canvas also encapsulates and efficiently maintains the Xwindows graphics context. This allows the application to specify colors, fonts, brushes, etc., in each drawing function call instead of directly maintaining the Xwindows-specific graphics context that in turn aids portability and application code clarity.

The ColorMap module interacts with the World module to provide a centralized implementation for the bit-plane utilization strategy. The RSLS display software is currently targeted to displays that support 8 bits per pixel (8-bitplane) color that are used to index into a color map of 256 entries that convert the 8 bits into red, green, and blue intensities. To facilitate independent drawing and erasing of data from multiple sources, certain bits are reserved for specific purposes, such as drawing target icons, tags, and alerts that have high priority and should thus act as overlays to the remaining displayed items. The unreserved bitplanes are used for the remainder of the displayed data. The ColorMap sets the red, green, and blue (RGB) intensity values for each of the 256 display color map entries in a manner that achieves the overlay effect (i.e., all 128 entries that correspond to having the target-icon bit set have the RGB intensities corresponding to the target-icon color).

The ViewFrameGlyph serves two purposes. First it is responsible for setting the transformation matrix in its Window's Canvas based on user requests of center coordinates, range, and rotation. To support this role, it acts as a Listener and a Handler to receive and handle certain user keystrokes. Second, it is the main Glyph seen by a Window and maintains a list of all the Glyphs displayed in that window. When the Window calls for a redraw, the ViewFrameGlyph sets the Canvas's transformation matrix for the window, clears the window, and then tells each Glyph in its list to redraw.

7.10.1.2 Airport map display

The Map module reads and maintains one or more static databases of multiply segmented lines that produce line drawings. The database consists of a list of labeled points whose coordinates are given as latitude and longitude. Sequences of the points are then referenced by label to specify multi-segmented lines. The Map contains a primary database and multiple overlay databases. Each Window's ViewFrameGlyph uses a MapGlyph to render the contents of the Map. The primary database in the Map is always rendered, while the overlay databases can be turned on and off by the user. The MapGlyph plays the roles of a Listener and a Handler to receive the overlay toggling keystrokes. The Map (primary and selected overlays) is drawn on command from the containing ViewFrameGlyph. Overlays are drawn immediately when toggled on by user input to the MapGlyph. Since an overlay can not be erased without potentially damaging other information on the screen, the MapGlyph forces its Window to perform a redraw when an overlay is turned off.

7.10.1.3 Data tag display

The TagGlyph serves as a centralized agent for rendering data tags in a Window. It keeps track of the location of each tag and attempts to avoid tag overlaps using an algorithm that gives a high priority to not obscuring targets and seeks to minimize the degree of orientation change for a particular target's tag

from update to update. The TagGlyph can display up to three lines of ten characters each in addition to a single character at the target position and a leader line. When creating or updating a tag, the caller specifies the tag color and preferred orientation (if any) in addition to the tag contents. For each tag rendered, the TagGlyph retains sufficient information to erase the tag during an update or to redraw the tag during a Window redraw. The TagGlyph module also provides a function that allows a client to get the handle of the TagGlyph associated with a particular ViewFrameGlyph. This function will create a TagGlyph and associate it with the ViewFrameGlyph if none exists already.

7.10.1.4 DBRITE-like display

DbriteReceiver creates an approximate copy of the image that would appear on the DBRITE display that currently exists in the Logan tower. It receives data from the MDBM interface and displays the data blocks that are being presented on the Final Vector controller's DEDS console. The data from the MDBM interface contains all the information required to render full tag, partial tag, and single symbol targets. Because the MDBM interface does not provide track numbers (or track drops), DbriteReceiver must determine when new data represents an update of a prior report and when a report has aged sufficiently that further updates are unlikely. To associate new reports with prior reports, DbriteReceiver maintains two tables of target information. The first table contains targets for which the MDBM interface has extracted an aircraft identification. The targets in this table can be reliably updated based on the aircraft identification contained in subsequent reports. The second table contains targets without discrete identification. DbriteReceiver uses a distance criterion to determine if new data represents an update to a target in this table. DbriteReceiver also uses the dispatcher to trigger periodic checks for stale data. The actual rendering of data tags for the DbriteReceiver is performed by the TagGlyph, which allows DbriteReceiver's tags to be shown in a non-overlapping fashion with the tags for sensor fusion's tracks when simultaneously viewing both sources of data. DbriteReceiver uses a small database file that specifies the data used to translate target positions into world coordinates, which DBM to use as a data source, and how long to wait before declaring a target to be stale. In a typical DBRITE-like configuration of the RSLS display, only the MapGlyph, DbriteReceiver, TagGlyph, and infrastructure components are active in a given Window.

7.10.1.5 Sensor fusion track display

The RSLS display renders track data from sensor fusion using the interplay of four components: TrackReceiver, TrackGlyph, DrawableTarget, and TagGlyph. TrackReceiver is responsible for establishing the inter-process communication connection with sensor fusion and receiving the track reports. It also accepts track alert status information from the AlertReceiver. It creates a TrackGlyph for each ViewFrameGlyph in which track data is to be displayed and maintains a list of TrackGlyphs to enable the broadcasting of each track report and track alert status to the TrackGlyphs. Each TrackGlyph maintains the state of several track-rendering options (tracks on, tags on, tag format, peak or current size, heading indication for stopped targets) and a table of DrawableTargets. The TrackGlyph acts as both a Listener and a Handler to accept user keystrokes that modify these options. The DrawableTarget contains the relevant information about each track and implements the drawing operations required to render a track according to the TrackGlyphs option settings. When the tags option is on for its TrackGlyph, the

DrawableTarget formats the text lines for the tag and uses the TagGlyph to render it. The DrawableTarget will also draw the alert symbology if the track is involved in an alert.

7.10.1.6 Approach bar display

The compressed scale depiction of targets on approach to the runways is accomplished using three components: ApproachBarReceiver, ApproachBarGlyph, and ApproachBar. These three components have similar characteristics to TrackReceiver, TrackGlyph, and DrawableTarget respectively. ApproachBarReceiver is responsible for establishing the inter-process communication connection with surface monitor and for receiving the approach bar messages. It also accepts track alert status information from the AlertReceiver. It creates an ApproachBarGlyph for each ViewFrameGlyph in which approach bar data is to be displayed and maintains a list of ApproachBarGlyphs to enable the broadcasting of each approach bar message and alert status to the ApproachBarGlyphs. ApproachBarGlyphs create and maintain a table of ApproachBars. One Approach bar is created for each runway and is initially positioned near the runway threshold based on data from the surface database. Each ApproachBar renders itself and the symbols for tracks appearing on it. It keeps track of its displayed position in 'airport coordinates,' how far from the threshold of the associated runway each occupying track is determined to be by the surface monitor, the actual position of each track, and whether each track is involved in an alert. Tracks are only depicted on the ApproachBar when their position is outside the limits of the ViewFrameGlyph and thus would not have a visible track icon. The ApproachBar renders itself in one of three modes: always visible, visible only when occupied by a track, or visible only when occupied by a track involved in an alert. The current mode in effect is stored by the ApproachBarGlyph and can thus be set independently in each window. The RSLS display does not currently show data tags for tracks indicated on the approach bars, but provisions have been made to support them.

7.10.1.7 Runway status light display

Three components comprise the software module that depicts the state of the runway status lights in each window. The LightStatusReceiver establishes the connection to the light governor process from which it receives light status messages. The LightStatusReceiver maintains a list of LightGlyphs to which it distributes initialization data for each light and light state updates. Each LightGlyph is responsible for rendering the state of the lights in a given window. The LightGlyph maintains a table of DrawableLights. The DrawableLights render themselves on the LightGlyph's associated Canvas according to their state: on or off. There are actually two forms of DrawableLights, BarDrawableLights and DirDrawableLights, which differ in how they render themselves on the display. BarDrawableLights, which are meant to represent what would be used for an operational display, appear as a solid line across the taxiway at the runway edge (for runway entrance lights) or across the departure end of the runway (for takeoff hold lights) when in the on state and are invisible when in the off state. DirDrawableLights, which give a clearer depiction of the light location and viewable direction for demonstrations, appear as narrow triangles 'shining' down each side of the taxiway or down the center of the runway when in the on state. DirDrawableLights appear as a much smaller triangle when in the off state to serve as a reminder of the position of the lights. The safety database is used to determine the endpoints that correspond to a given light for each BarDrawableLight. The DirDrawableLight uses the same endpoints as the apexes of the

two triangles drawn for runway entrance lights. It uses the taxiway and runway centerline data from the surface database to place the axis of symmetry of the triangle parallel to the taxiway that the light serves with the triangle fanning out away from the runway. For a takeoff hold light, the apex is placed where the segment formed by the two endpoints intersects the associated runway centerline, and the triangle fans out towards the departure end of the runway. Both types of DrawableLight can also show the polygons specified in the safety database as the light's activation region and, for takeoff hold lights, arming region. The LightGlyph serves as both a Listener and a Handler to accept keyboard input that allows the user to toggle the rendering of these polygons in each window.

7.10.1.8 Alert display

The AlertReceiver establishes a connection to the surface monitor to receive messages indicating when alerts arise and when they disappear. For each track identified in the alert message, the AlertReceiver increments or decrements an alert count as appropriate. When the count changes from zero to one or from one to zero, it notifies the TrackReceiver and ApproachBarReceiver that the indicated track is or is not in an alert state so that the track and approach bar rendering subsystems can draw or erase the alert symbology. On the receipt of a new track, the TrackReceiver and ApproachBarReceiver query the AlertReceiver for the track's alert status in case an alert message was received by the display before the first message for the track.

7.10.1.9 Unfiltered ASDE target display

The RSLS display can show the direct output from ASDE processing in addition to the tracks from sensor fusion. This feature is primarily used for evaluating the performance of the scan-to-scan association in ASDE processing and the behavior of sensor fusion's fusion and filtering. The ASDE_Receiver establishes the connection to the merge process of the ASDE processing subsystem, receives the target reports, and after translating the target position from ASDE processing coordinates to system coordinates, passes the data on to each ASDE_Glyph. The ASDE_Glyphs draw a filled circle for each target at the reported position and with an area equal to the reported target area. The targets are drawn in one of two colors depending on whether the target is reported to be low or high confidence. The ASDE_Glyph can also indicate the last three digits of the targets ASDE track number above the filled circle. The ASDE_Glyph stores the relevant information for each target for redrawing or erasing during the next update. The ASDE_Glyph serves as a Listener and a Handler to accept keyboard input to toggle the display of ASDE targets and target numbers in each Window.

7.10.1.10 Snapshot display

As an aid to analyzing the performance of its projection engine, the surface monitor produces a target snapshot each time a target is updated. The target snapshot indicates the assumed state of the target and a set of segments that represent the results of the acceleration and deceleration projections. These snapshots are received by the SnapshotReceiver, which forwards them to each element in the list of SnapshotGlyphs that it maintains. The SnapshotGlyphs render the snapshot information in each window as it is updated for each target and store the current snapshot for each target for redrawing or erasing as

part of the next update. The SnapshotGlyphs also serve as Listeners and Handlers to accept keyboard input to toggle the display of snapshots in each Window.

7.10.1.11 Surface graph display

When the surface monitor constructs its surface graph, it can generate a data file that indicates the arcs and segments along which targets will be projected. The SurfaceGraphDisp can read this data from the file. It maintains a list of SurfaceGraphGlyphs. Each SurfaceGraphGlyph can draw the arcs and segments stored by the SurfaceGraphDisp on a Canvas. The SurfaceGraphGlyph serves as a Listener and a Handler to accept keyboard input to toggle the display of the surface graph in each Window.

7.10.1.12 Master clock control

When the RSLS system is being operated in playback mode, the RSLS display can be used as an interface to the master clock to control the time rate of the playback. The MasterClockInterface establishes the connection with the master clock. It maintains a list of ClockControlGlyphs from which it receives time-rate-control keyboard inputs and to which it passes time and time rate updates. The ClockControlGlyphs serve as Listeners and Handlers to receive keyboard inputs to double, halve, or freeze the time rate or to set it to unity. Whenever the time rate changes, the ClockControlGlyph that is associated with the primary Window displays messages to the operator to indicate the current time rate. When the time rate is frozen, the current time is also indicated. These messages are removed after ten seconds to avoid unnecessary display clutter.

7.10.2 Significant Problems and Solutions

The primary requirements and design objective for the RSLS display are the following:

- use Xwindows to allow generating displays on a large variety of hardware with Xwindows support including high contrast ratio and resolution grayscale monitors and a high contrast ratio color backlit liquid crystal display attached to a portable IBM-compatible PC
- provide an infrastructure to allow software for rendering new data to be quickly added to the display to give analysts new views of RSLS's operation
- support multiple views of the same information in the form of a primary window with sub-windows
- guarantee that certain high-priority information such as track icons and alert indicators can not be completely obscured by lower-priority information.
- provide acceptable real-time performance, for instance, allow new data to be promptly rendered on the display without redrawing the entire window.

The architecture of the RSLS display software, especially the extensive and flexible core infrastructure, meets these needs to a large extent. Not including the InterViews package and various project-wide library modules, the RSLS display consists of only about 13,000 lines of code. Furthermore, many modules are structurally similar, which has expedited their creation and debugging.

7.10.3 Suggested Improvements

The current user interface for changing display settings and options is based on single keystroke (hotkey) commands. While this interface is very efficient for users who are familiar with the hot-key assignments, a graphical user interface would provide a friendlier alternate interface. Also, the current interface does not support changing the number, size, or position of sub-windows once the display has been initialized.

7.11 MASTER CLOCK

7.11.1 Process Architecture

The master clock is a time server that provides time synchronization for multiple processes. The master clock and its clients generate the 'current-time' using three pieces of information: an operating system base-time, the 'current-time' corresponding to the base-time, and the rate of time passage (time-rate). This approach allows the master clock and its clients to maintain synchronous time at not only a real-time rate, but also at fast and slow-time rates.

The master clock process is event driven. The two types of events are the connection of new clients or messages from existing clients. Clients can connect as either local clients or remote clients. Local clients must be executing on the same machine as the master clock and use a protocol that allows exact time synchronization based on the common system clock. Remote clients can execute on any machine on the network, but they achieve less precise synchronization since they cannot rely on system clock synchronization. Initially, all client processes connect as remote clients. When the master clock detects a new remote client connection, it sends the client a message indicating the name of the machine on which the master clock is executing. If the machine name matches the client's machine name, the client will close the remote connection and re-connect as a local client. When the master clock detects a new local client connection, it sends the local client time synchronization data.

When the master clock is not running in real-time mode, it listens for time or time-rate change messages from its clients. When such a message is received, the master clock sends time synchronization data to the local clients and sends an update required message to the remote clients. The master clock also listens for and responds to update request messages from remote clients. Remote clients generate update request messages when they determine that they can not connect as local clients and when they receive an update required message.

7.11.2 Input and Output Data Description

Table 7.11.1 shows the master clock input messages. Local clients generate the LOCAL_SET message to change the time and/or time-rate. Remote clients generate the UPDATE_REQUEST, SET_TIME, SET_FACTOR, and SET_TIME_AND_FACTOR messages.

TABLE 7.11.1

Master Clock Input Messages

Type	Additional Fields
LOCAL_SET	base-time, corresponding-time, time-rate
UPDATE_REQUEST	none
SET_TIME	current-time
SET_FACTOR	time-rate
SET_TIME_AND_FACTOR	current-time, time-rate

Table 7.11.2 shows the master clock output messages. Master clock sends a SERVER_HOSTNAME message to each remote client when the client connects. The LOCAL_UPDATE message is sent to local clients when they first connect or when the current-time or time-rate changes. UPDATE_DATA messages are sent to remote clients as a signal to them that an update is required and to individual remote clients in response to an UPDATE_REQUEST message.

TABLE 7.11.2

Master Clock Output Messages

Type	Additional Fields
SERVER_HOSTNAME	machine name as an ASCII text string
LOCAL_UPDATE	base-time, corresponding-time, time-rate
UPDATE_DATA	current-time, time-rate

7.11.3 Suggested Improvements

The mechanism for synchronizing clients on remote machines is not as robust or accurate as it could be. A better scheme would use multiple messages to avoid the potential error induced by the delayed delivery of a single message. More accuracy could be achieved by measuring round-trip message times and compensating for the estimated message latency.

8. PERFORMANCE OF THE ASDE-X SURVEILLANCE SUBSYSTEM

During the RSLS system demonstrations at Logan Airport beginning in October 1992, a number of design refinements were made to improve performance. In August 1993 an evaluation was undertaken to assess the performance of the system as of that time. This chapter summarizes the results of the performance assessment of the ASDE-X surveillance system by itself. The next chapter addresses the whole system including safety functions together with surveillance.

Ideally the ASDE-X radar surveillance system would produce one target report per scan for each aircraft within the region of interest, with the track number of each aircraft remaining constant as the aircraft moves over its complete path. The same would be true for ground vehicles. Also there would be no false tracks. Thus the many runway lights and other bright spots in the radar images, including multipath effects, would be suppressed so that they do not form tracks. Furthermore this would continue to be true when it rains or snows.

A practical system cannot achieve this ideal, but instead has a surveillance reliability not equal to 100 percent and a false track rate not equal to zero. We have conducted an analysis of data from the demonstration equipment at Logan airport to quantitatively determine these surveillance performance figures and to assess surveillance accuracy.

8.1 EXAMPLES OF SURVEILLANCE QUALITY

Before presenting statistical measures of performance, we begin with some specific data in order to present concrete ideas as to the types of imperfections that can occur. We begin by selecting a data-set arbitrarily, the only criterion being that it should be a busy period on a clear weather day. The subject data was recorded on 12 August 1993, beginning at a busy time in the afternoon, at 4:20 PM. At that time, aircraft were landing on runways 4R and 4L, and taking off from runways 4R, 4L, and 9. While this radar data was being recorded, a log was kept of the aircraft types.

The data analysis began by displaying the surveillance tracks, limiting attention to "established tracks." These are the tracks that are declared to be high-confidence tracks by the surveillance processing. Whereas the system also includes lower confidence tracks, only established tracks are used in the safety subsystem to activate runway status lights.

Although surveillance in the demonstration equipment at Logan airport is provided by both the ASDE radar and the ASR-ATCRBS radar, this section focuses on just ASDE. Thus the analysis here is applied to data originating only from the ASDE radar.

Each track displayed consists of target reports, which occur nominally once per scan. Looking at the display of tracks, we clearly recognize the well defined tracks of aircraft landing and taking off. For example, Figure 8-1 shows the superposition of the first 20 scans. To the analyst, this display evolves as a function of time, from which it is evident that the track on runway 9 is moving upward in the picture, and the tracks on runways 4R and 4L are moving toward the left. Velocities are indicated by the spacings between target reports. Thus it is evident that the aircraft on runway 9 is taking off, the aircraft on runway 4R is landing, and the aircraft on runway 4L is taking off. Also an aircraft has taxied out to runway 9 to begin a takeoff, with another aircraft behind it, and other aircraft are taxiing, some outward to

takeoff and some inward toward the terminal buildings. In this display, each target report is plotted as a circle whose area indicates the area of the reported radar target.

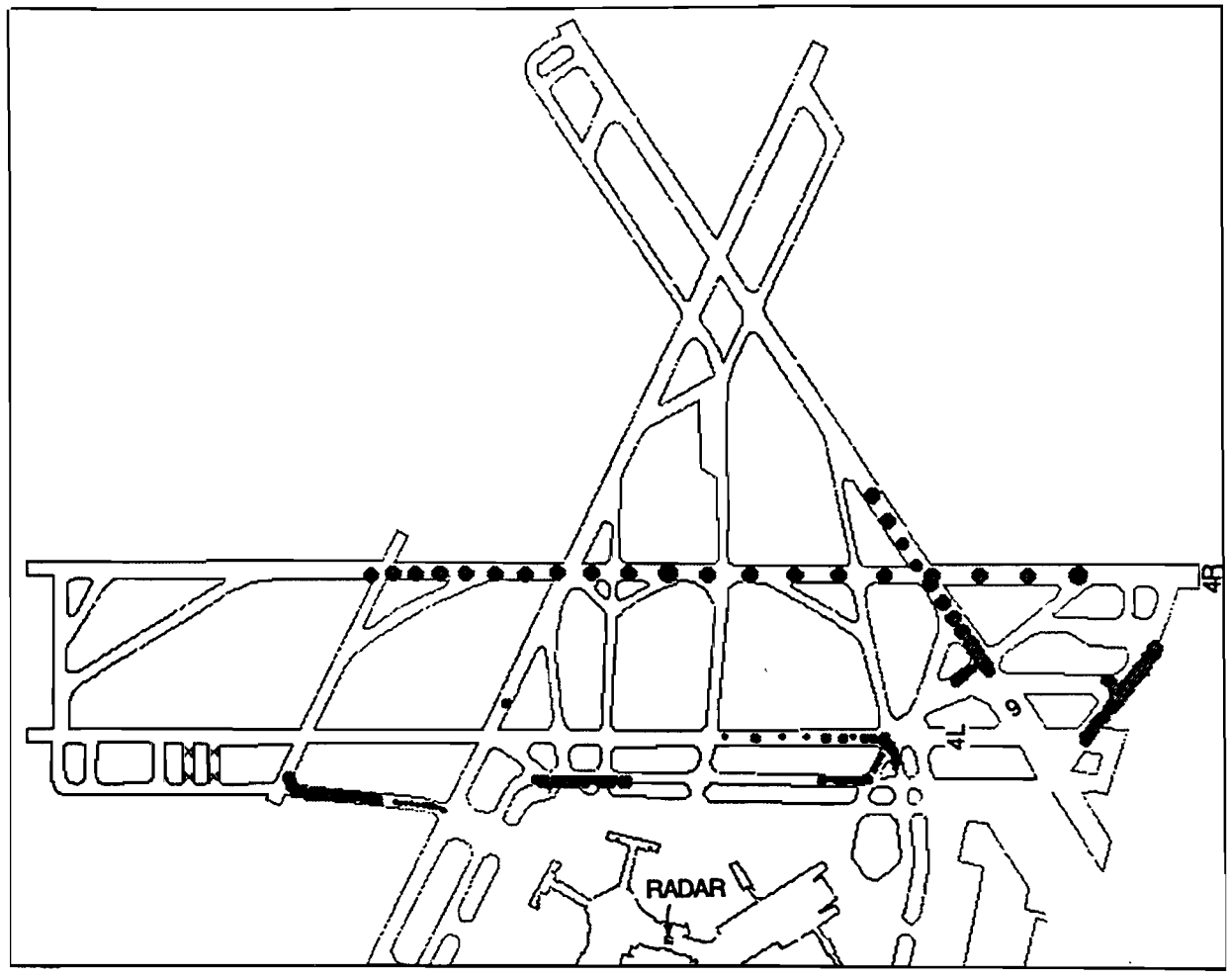


Figure 8-1. Display of tracks from the ASDE-X radar.

The written log indicates that the runway 9 takeoff was a Boeing 727, the runway 4R landing was a Boeing 737, and the runway 4L takeoff was a smaller commuter aircraft with twin propeller engines.

The track numbers can readily be determined by a display option in which track number appears next to the current target report. It becomes clear, for example, that the aircraft landing on runway 4R has track number 108672. Knowing this, we can then display this track by itself, which is shown in Figure 8-2. Seeing the whole track indicates that after landing, the aircraft turned off the runway onto taxiway R, then crossed runway 4L, then followed taxiway N, crossing runways 33R and 33L, until reaching the terminal buildings. The aircraft remained in track during this motion, with a fixed track number. The surveillance performance for this aircraft is seen to be quite good, which is typical.

8.2 SURVEILLANCE RELIABILITY

To obtain a quantitative estimate of surveillance reliability, we have examined a number of tracks in this manner. Beginning with a plot of all established tracks, we identify track numbers for landing and taking-off aircraft, and then display these tracks one-by-one to determine any surveillance gaps. A computer program is also used to check whether a target report is generated on every scan.

In determining surveillance reliability, it is necessary to define the "region of interest." Experience in studying this data indicates that surveillance is less reliable near the terminal buildings. This is to be expected because some terminal buildings obstruct the radar view, and because the multipath environment tends to be more severe near the terminal buildings. On the other hand, in a runway status light application, surveillance is mainly required on the runways and nearby. For these reasons, we have defined the surveillance region of interest for this analysis to begin between the outer taxiway and runway 4L, as illustrated in Figure 8-2.

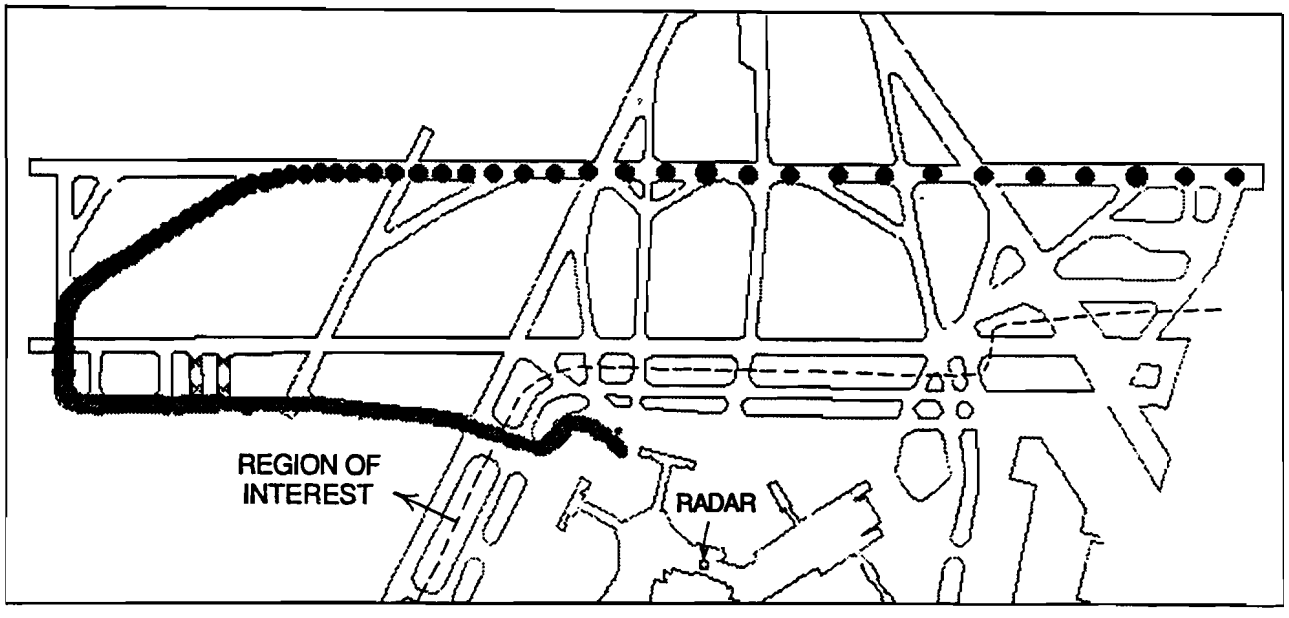


Figure 8-2. Boeing 737 landing (track 108672).

The region of interest must also be defined with respect to landing aircraft that appear in surveillance before they reach the runway. Similarly, taking off aircraft climb to an altitude above the coverage of the radar antenna. Because this analysis focuses on ASDE surveillance, we have limited it to surveillance on the surface . Thus for a landing aircraft, the region of interest in this analysis begins at the runway threshold. For an aircraft taking off, the aircraft is considered to be within the region of interest until the point at which the track disappears, provided that the path remains within the horizontal boundaries of the runway.

TABLE 8.1

Surveillance Performance for the First 20 Aircraft

Time (EDT)			aircraft	operation	RW	track no.	scans	misses	swap?	coasts
4	20	40	B727	takeoff	9	103306	169	0		0
4	20	45	twin prop	takeoff	4L	107744	33	0		0
4	21	30	twin prop	landing	4L	109299	20	0		0
4	21	30	twin prop	takeoff	9	107417	109	10		0
4	22	19	B727	landing	4R	110026	162	0		0
4	22	35	twin prop	takeoff	4L	109892	36	13		0
4	22	40	not logged	takeoff	9	106282	86	0		0
4	23	15	twin prop	landing	4L	110570	33	0		0
4	23	15	B727	takeoff	4R	108078	122	0		0
4	24	20	Cessna 421	landing	4R	111293	130	0		1
4	24	20	twin prop	landing	4L	111600	38	0		0
4	24	13	Shorts 360	takeoff	9	107887	90	0		0
4	25	22	twin prop	landing	4R	112307	89	0		0
4	25	40	B737	takeoff	9	106736	94	0		0
4	26	15	Shorts 360	takeoff	4L	109994	37	0	trk swap	2
4	26	24	DC9	landing	4R	113142	99	1		0
4	26	50	twin prop	landing	4L	113239	47	0		0
4	27	0	twin prop	takeoff	9	109083	105	0	trk swap	2
4	27	30	B737	landing	4R	113803	165	0		0
4	28	23	twin prop	takeoff	9	109549	110	0	trk swap	5
						Totals:	1771	24		10
						Fraction:		0.0135		0.0057
						Reliability (%):		98.645		99.428

This analysis procedure was applied to the first 20 aircraft that landed or took-off in this set of recorded data. The results are summarized in Table 8.1. During this period, the average rate of landings and takeoffs was 2.5 operations per minute, which is quite busy. The results indicate that most of the aircraft were tracked regularly and completely while they moved through the surveillance region.

Track Flaws. The individual surveillance flaws seen in Table 8.1 can be described more specifically as follows. An aircraft taking off from runway 9 (track 107417) experienced a track drop shortly before entering the runway. Subsequently, after a gap of 10 scans, a new track was established for

this aircraft. A similar condition occurred for track 109892, which was a takeoff from runway 4L. For an aircraft landing on runway 4R (track 113142), the track was late in becoming established by one scan. Also a track swap among three aircraft occurred when these aircraft were in close proximity in the vicinity of the runway 4L threshold. This period was analyzed in detail in an attempt to assign tracking misses. Although all three aircraft had changes in track number, and all three tracks had coasts, there were consistently 3 tracks corresponding to the 3 aircraft. We have adopted the following assignment of flaws: no drops in surveillance, three cases of track number change, and coasts in all three tracks as shown in the table.

Coasts. In addition to the flaws at the tracking level, some coasts occurred, which means that a target report was not issued in a particular scan but the track was still active. There was one other coast in addition to the coasts associated with the track swap. It was a small aircraft, a Cessna 421, landing on runway 4R. Its radar image was quite small and it disappeared for one scan. This is a relatively minor flaw, because the aircraft was being reported regularly before and after the 1-scan coast, and always with the same track number. Thus there was not much uncertainty as to the location of the aircraft at any time.

Percentages. The results in Table 8-1 can be combined to estimate the surveillance reliability. For each aircraft analyzed, the time it entered the region of interest and the time it exited were determined. The difference was the total exposure—that is, the number of scans in which surveillance was assessed. Altogether the total is 1771 aircraft-scans. The subset of this number in which the surveillance system did not generate a track was also counted, and these flaws total 24 aircraft-scans. Thus the overall surveillance reliability was 98.6 percent at the track level. This is an estimate of the system's probability of having an aircraft in track: P(aircraft in track) = 98.6 percent.

Coasts can also be expressed as a percentage. For an aircraft that is in track but may experience coasts, we define the update reliability as the percentage of all track-scans in which a target report was generated. The result for this set of data is: P(report) = 99.4 percent.

In summary, the surveillance analysis described above was formulated under the following conditions. Only surveillance data from the ASDE radar was used, limiting attention to established tracks. The targets of interest were aircraft that either landed or took-off, and they were assessed for the whole time the aircraft was within the region of interest, whether on a taxiway or a runway. The resulting reliabilities for generating a track and generating a report are as follows:

P(aircraft in track) = 98.6 percent

P(report) = 99.4 percent

An alternative view of surveillance reliability is provided from an analysis of the full system, including runway status lights together with radar surveillance. In such an analysis the surveillance is only assessed in situations where a runway status light would normally have been turned on by the presence of aircraft. For this purpose, the analysis of the full system, which is described in the section that follows, has been adapted to provide results that assess surveillance.

Specifically, we have studied the performance of takeoff-hold lights, limiting attention to situations in which a light would normally be turned on, but was not because of the absence of a surveillance track on an aircraft. Table 8.2 gives the results of this analysis applied to four data sets, each about 1 hour in

duration. This is a large quantity of data including many different aircraft and many situations. The overall result is that the radar surveillance successfully supported the takeoff-hold lights 96.6 percent of the time.

TABLE 8.2

Surveillance Performance Based on Runway Takeoff Hold Lights

date	start time	conditions	runway	opportunities (sec)	misses (sec)
3/11	14:50	VMC	27	315	7
			33L/G	146	28
			33L	51	0
3/13	9:45	snow	9	390	0
3/31	19:15	VMC	4L	320	0
			9	261	5
4/21	17:30	VMC	22L	491	0
			22R	433	44
			22R Int.	79	0
Totals:				2486	84
Ratio:					0.034
Reliability (%)					96.6

This result can be compared to the surveillance-only result given above, provided we take account of the fact that two aircraft must both be in track for a given takeoff-hold light to be illuminated. If the probability of successful surveillance of one aircraft is denoted P1, then, according to the most straightforward model, the probability of successfully having two given aircraft in track is $P2 = P1^2$. Thus having measured P2 = 0.966, we can calculate P1 = 0.983. This result from analysis of the full system is in good agreement with the surveillance-only result given above.

In summary, the reliability of the radar surveillance included in the demonstration equipment at Logan airport has been assessed and found to be high although not perfect. The reliability for having a given aircraft in track at a given time is about 98 to 99 percent.

Finally, it should be mentioned that as these results were obtained, the specific flaws in surveillance that were observed suggested to the analysts some ideas for further refining the design and improving performance. More specifically, most of the observed flaws in surveillance reliability were associated with the multipath rejection algorithms. These algorithms generally achieve a low false track rate at the expense of some decrease in surveillance reliability. More sophisticated multipath rejection logic is now envisioned that promises to further increase surveillance reliability while also reducing the false track rate (described in Section 3.4).

8.3 FALSE TRACK RATE

Experience with the demonstration equipment at Logan Airport indicates that false tracks occasionally occur, primarily as a result of multipath (mechanism described in Chapter 3). On the other hand, the multipath elimination functions that are now included in the system are effective in making the false track rate among established tracks quite low. We have undertaken an analysis to quantitatively determine the rate of false tracks.

This analysis begins with the same set of data discussed in Section 8.2, which was recorded on 12 August 1993 during a busy period in the afternoon in clear weather. Having identified the track numbers for aircraft that are landing or taking off, we can eliminate their tracks and observe what other tracks were generated. These other tracks are then examined in a display of the form shown in Figure 8-1. The results show some tracks on runways and taxiways, which may be either vehicles or false tracks. To determine whether or not they are vehicles, these tracks were displayed individually.

Specifically, working with the 12 August data beginning at 4:20 PM, a total of 86 aircraft tracks were deleted during a 33 minute period. The display of the remaining established tracks showed that they included a number of vehicles moving around the perimeter road and other vehicles in the vicinity of the terminal buildings. Targets that are probably aircraft on the inner and outer taxiways but outside the region of interest also appear. This is reasonable because no attempt was made to delete aircraft unless they landed or took off during the time period studied. Targets that are probably boats in the harbor also appear. Limiting attention to the runways and taxiways within the region of interest, we only see two tracks (after the deletion of the known aircraft tracks). Examination of these in detail indicates that they are both false tracks.

The first false track appeared on runway 4R, near the intersection with Taxiway P. It was an established track briefly, including one target report, after which it coasted and was dropped. Looking at the low-confidence track that preceded the established track, we recognize a false track pattern that has been seen before. An aircraft was taking off from runway 4L, and was in-track. The false target appeared at the same azimuth as the aircraft target, except slightly to the rear, which is consistent with the phenomenon of reflection from the tail of the aircraft. We believe that this is a type of false target in which the radar signal reflects first from the tail of the aircraft, then reflects from Terminal B or a parked aircraft back to the first aircraft, and then reflects back to the radar. The result is that the false target appears at the same azimuth as the tail of the aircraft, but at a longer range as a consequence of the longer path.

The other false track appeared on runway 4R near the intersection with taxiway R. The duration of the false track included 4 target reports, after which it coasted and was dropped. This track also occurred consistently out-range from an aircraft that was in-track. The aircraft had landed on runway 4R and was beginning to turn onto taxiway R when the false track appeared. We believe that the mechanism causing this false track was a reflection from the aircraft to a nearby object such as the glide-slope antenna that is mounted there, then back to the aircraft, and then back to the radar.

The results of the false track analysis can be summarized as follows. In a period of 33 minutes, or 1130 scans, the number of false established tracks within the region of interest was 2. Both were of short duration, much shorter than the tracks of aircraft. The false track rate is about 4 per hour. Given that the total length of runways and taxiways examined in this analysis is about 21 KM, the false track rate can be

expressed as 0.17 per hour per Km. This form of the rate may be more useful to projecting performance to different airports.

8.4 SURVEILLANCE ACCURACY

To assess the accuracy of the ASDE-X radar surveillance subsystem, we have analyzed the reported centroid positions with respect to the smoothness of the track and consistency with the probable motion of the aircraft (see also Appendix D).

Time periods when aircraft are taxiing, and experiencing low values of acceleration, provide good opportunities for analyzing surveillance accuracy. For example, the left-hand plot in Figure 8-3 shows track 113142, which is the landing of a DC9 on runway 4R. After decelerating, the aircraft turns onto taxiway Y, and then onto runway 33R for taxiing to the terminal buildings. One way to look at the jitter in the reported centroids is to form the predicted position for each new scan based on the two preceding reports and then to compare the new position report with the prediction. The prediction is given by

$$x_{p3} = 2\,x_2 - x_1$$

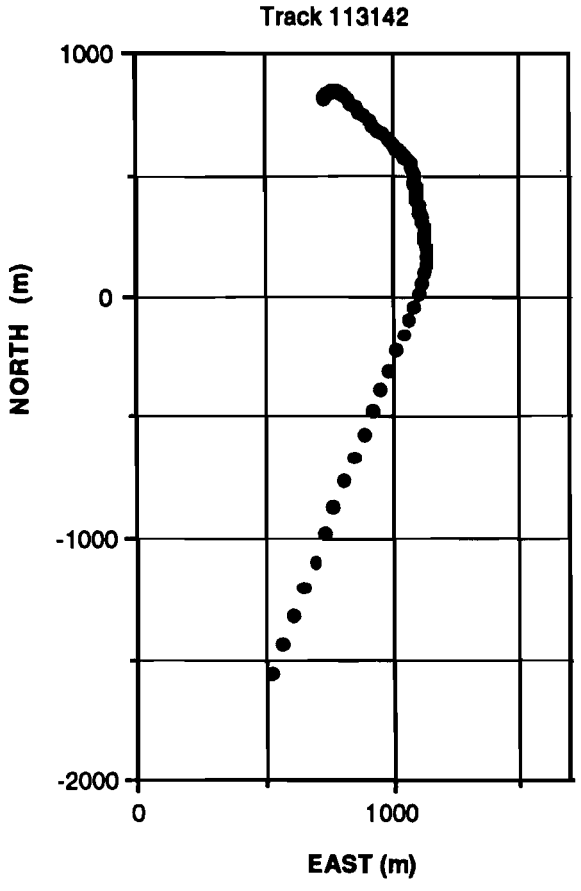
where time index 3 represents the new scan and 1 and 2 are the preceding scans. This formula gives one of the two coordinates of the target report. The other coordinate has the same form. The comparison between the prediction and the new measurement is given by the residual

$$\text{residual} = x_3 - x_{p3}$$

Figure 8-3 shows the resulting residuals for this track, for both the x and y coordinates (here taken to be the east and north coordinates of the target reports). We see that the residuals exhibit jitter that is both positive and negative. The average value of the residuals is approximately zero, except when the aircraft was decelerating strongly. Thus if we limit attention to the regions when the aircraft was not accelerating or decelerating strongly, we can assess surveillance jitter from the magnitude of these residuals, specifically the root-mean-square (rms) value. From the plots in Figure 8-3 it can be seen that the strong decelerating of landing ends about scan 6660, and thereafter the aircraft taxis at approximately constant speed over taxiway P and runway 33R. During this period, the rms value of residuals is calculated to be: 2.3 meters for the east coordinate and 2.6 meters for the north coordinate.

The rms value of the residuals is computed separately for the x and y coordinates. The rms value of residuals (denoted σ_r) can be used to infer the amount of jitter in the radar centroids (rms value denoted σ_c), by using the following relationship, which is derived in Appendix D.

$$\sigma_c = \sigma_r / \sqrt{6}$$



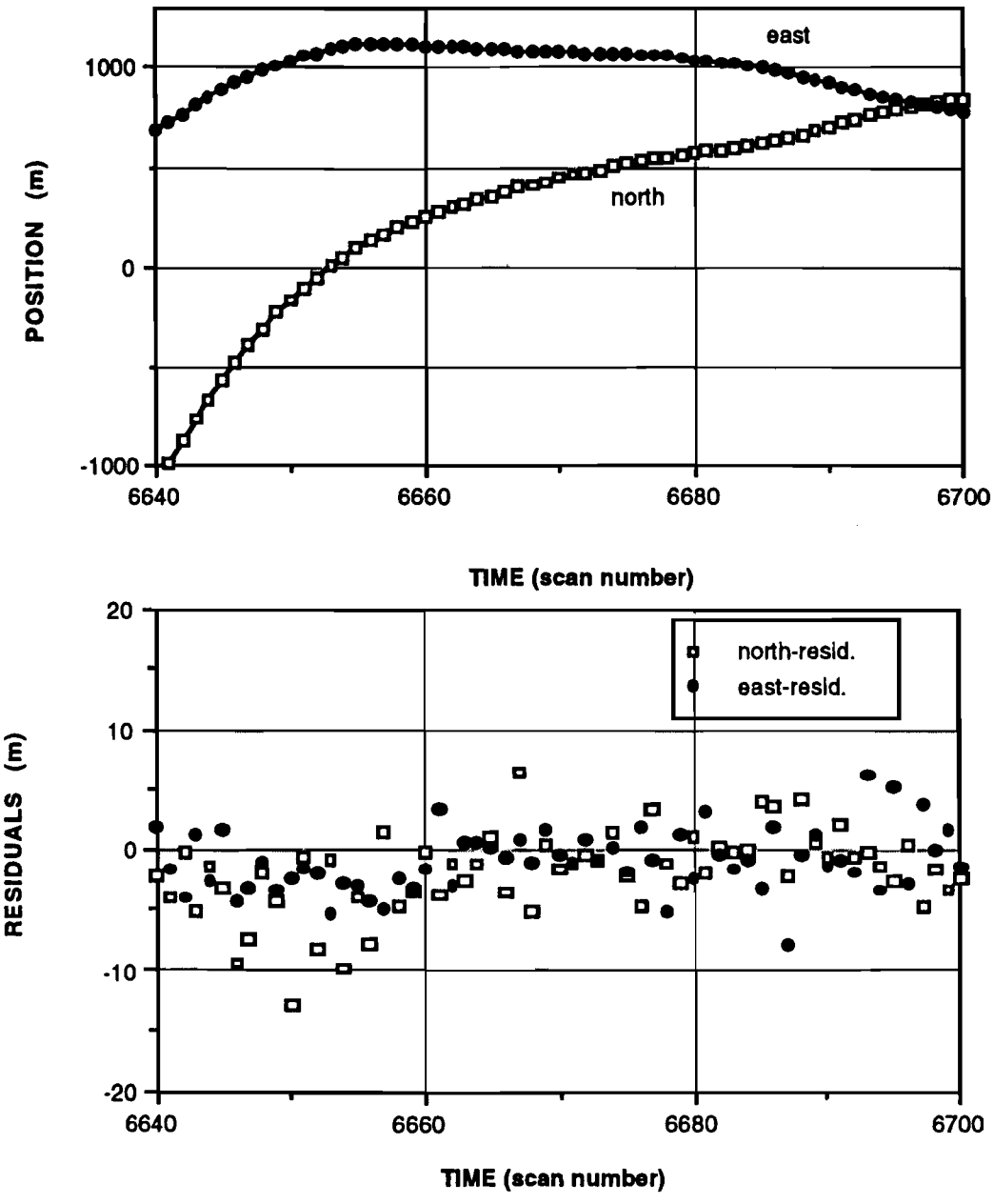


Figure 8-3. Analysis of jitter in track 113142.

This is based on the concept of the effective noise level of the surveillance system. We define the effective noise level of the surveillance system to be the level of a white noise (a model in which centroid jitter is independent from scan to scan) that would produce the same value of σ_r. This definition is appropriate for ASDE surveillance in applications in which scan-to-scan trends are used for determining velocities of aircraft.

Given the σ_r values from above, the resulting effective jitter levels are

σ_c = 0.9 meters, east
1.1 meters, north

It is interesting to see that the jitter in target centroid positions is so small. It is much smaller than the dimensions of an aircraft, and is even smaller than the resolution of the radar (which for this track is about 6 meters in range by 12 meters in cross range). This small level of jitter is beneficial because it will allow a determination of aircraft velocities and possibly accelerations with a useful level of accuracy.

This example is typical of results see for other aircraft, as can be seen in quantitative results for other aircraft tracks presented below. Results of this analysis appear to be not significantly dependent of the type or size of the aircraft.

Effects of Acceleration. Even when an aircraft is taxiing, it is accelerating somewhat, and these accelerations contribute to σ_r. This is undesirable in the assessment of centroid jitter, so we would like to limit attention to periods of low acceleration. Analysis in Appendix D indicates that when acceleration is non-zero it affects σ_r according to the relationship:

$$\text{rms(residuals)} = \sqrt{6\,\sigma_c^2 + a^2}$$

where acceleration, a, is given in units of distance per scan squared. If σ_c is approximately 1 meter, as in the above example, then to keep the acceleration contribution less than 10 percent of the total, accelerations should be less than 1.3 meters per scan squared. That is, the analysis should be limited to time periods when the acceleration is smaller than this.

The ASDE-X jitter analysis applies these principles as follows: select a track for analysis and determine the time periods in which accelerations were not more than 1.3 meters per scan squared. For track 113142, this occurred during scans 6662 to 6682. The rms value of the residuals is then calculated over this period. From these values, the jitter in centroids is calculated. The results are

residuals,	σ_r =	1.8 m, east
		2.6 m, north
centroid jitter	σ_c =	0.73 m, east
		1.1 m, north

This analysis has been applied to other aircraft tracks in this data set, yielding the following results:

track	110026	Boeing 727 after landing on 4R	σ_c = 2.0 m, east 1.2 m, north
track	113803	Boeing 737 after landing on 4R	σ_c = 1.0 m, east 1.3 m, north
track	111293	Cessna 421 after landing on 4R	σ_c = 0.8 m, east 0.5 m, north

Altogether these results indicate that surveillance tracks from the ASDE-X radar have relatively low jitter. Rms values of jitter in the centroids are in the range 0.5 to 2 m rms. This is quite small relative to the dimensions of the aircraft, and is smaller than the radar resolution. In addition to reporting the location of the aircraft, the tracks are sufficiently smooth that the aircraft velocities can be determined to a useful level of accuracy.

Finally it should be added that in addition to jitter in the centroids, a bias may also be expected to some degree. Such biases would result from, among other things, the fact that the radar illuminates the aircraft from one side, and receives returns primarily from the parts of the aircraft on that side. The bias would vary with time and with type of aircraft. We see from surveillance displays, as in Figure 8-1, that the biases are not large relative to the width of a runway and to the ordering among aircraft on a particular way. Other than that qualitative observation, the assessment given here addresses only jitter, because of its relevance in the use of the data for estimating aircraft velocity and acceleration.

9. PERFORMANCE OF THE RUNWAY STATUS LIGHT SYSTEM

9.1 INTRODUCTION

This performance assessment of the Runway Status Light System (RSLS) at Logan Airport is a quantitative evaluation intended to describe the system's performance in a way that is useful to those most directly concerned, that is, pilots, tower controllers, and surface-vehicle operators. They would want to know if the lights - were they installed - would improve safety and situational awareness, and if they would interfere with the normal flow of traffic.

The safety logic algorithms that control the lights are intended to prevent critical conflicts on the runways. They do this (in a complete system with lights on the airfield) primarily by preventing runway incursions. The algorithms do not, however, aim to enforce rules, nor do they detect rule infractions per se. An attempt has been made, in the design of the algorithms, to avoid reliance on knowledge of normal operations in predicting the evolution of scenarios, so as not to compromise the effectiveness of the safety logic in the kinds of abnormal situations that sometimes lead to incidents and accidents.

The assessment reflects the performance of the RSLS in its current form. Additional development remains to be done, so the results presented here should not be interpreted as an indication of the system's ultimate performance and capabilities. They can, however, be viewed as a lower-bound performance estimate.

The fast-paced development of the RSLS demanded strict prioritization of the work, and some software tasks had to be deferred. One of the specific items that has yet to be implemented is the safety-logic algorithm to handle land-and-hold-short operations. The surveillance, tracking, sensor fusion, and safety logic software is not in final form, and the parameters and other adjustable entities represent best initial estimates, to be further tuned on the basis of operational experience. Such a tuning effort is required to realize the system's full potential. Finally, the system as it currently exists has certain hardware limitations. These are likely to be eliminated in an operational system, resulting in better surveillance.

The current embodiment of the RSLS, with no presence either in the tower cab or on the field, makes possible an intuitive, qualitative assessment of the system's effectiveness and transparency: one indication of a well-tuned, effective, and transparent safety system is a traffic display where the traffic flows as if the pilots and surface-vehicle operators could see and were responding to the lights. It is clear from extensive observation of the interplay between the lights and live traffic on the RSLS display that the system already works quite well by this criterion, with very few anomalies.

The observed anomalous events will be discussed and interpreted from the limited perspective of the safety logic and the operation of the runway status lights. No attempt will be made to interpret the events in operational terms, as the complexity of airport operations makes this impossible without full knowledge of the circumstances.

9.2 PURPOSE AND SCOPE

The main purpose of this performance assessment is to provide preliminary quantitative data on the system's end-to-end performance. The performance of the RSLS is evaluated in terms of four performance measures, designed to gauge the system's ability to enhance safety while remaining transparent to normal operations. These measures are Missed Detections, False Alarms, Instances of Interference, and Light Infringements. The first and last address the system's effectiveness, the other two its transparency. The

first three are collectively referred to as light anomalies. All four will be defined and discussed in the next section.

Many other performance measures can be defined for the RSLS, some related to the safety logic, others to tracking and sensor fusion, surveillance, and surveillance processing. These will not be considered explicitly in this assessment, since they are not of direct concern to the end user. In those instances where flaws in surveillance, processing, tracking, prediction algorithms, etc., do manifest themselves as observed light anomalies, they will be identified and described as appropriate. But, in general, a complete evaluation of the performance of the individual functional system elements is considered to be a part of the description of the respective elements and therefore is not attempted here.

Human factors are beyond the scope of the present assessment. Clearly, human-factors concerns are very important in a system such as the RSLS, but they can only be addressed in connection with a more mature system that interacts as intended with the users. Several system features, such as airfield lights, tower-cab displays, and controller interfaces, that can be expected to figure prominently in a human-factors study, are not yet implemented.

9.3 PERFORMANCE MEASURES

The four performance measures are defined below. The definitions involve the concept of a "light threshold." This is a location or line near the light, beyond which the light state would not be clearly discernible from the cockpit, and which therefore should not be crossed when moving towards an illuminated light.

9.3.1 Definitions

Missed Detection: A missed detection is a failure of a runway status light to illuminate as it should, as judged from the intent of the safety logic and the state of traffic on the airport and in the immediate airspace.

False Alarm: A false alarm is a status-light illumination that does not reflect the real state of traffic on the airport and in the immediate airspace.

Interference: Interference occurs when a status light is on, in accordance with the safety logic rules and the state of traffic on the airport and in the immediate airspace, while an apparently safe operation is underway that leads to light threshold crossing within a specified time.

Light Infringement: A light infringement results when an aircraft or surface vehicle advances beyond the light threshold while the light is illuminated in accordance with the intent of the safety logic and the state of traffic on the airport and in the immediate airspace, and it appears unsafe to do so.

9.3.2 Discussion

The performance measures address the system's effectiveness and transparency. They can also in general be related to either the surveillance and tracking function or the safety logic function. Table 9.1 illustrates these relationships.

TABLE 9.1

Performance Measure Relationships

	Effectiveness	Transparency
Surv. & Tracking	Missed Detection	False Alarm
Safety Logic	Light Infringement	Interference

Table 9.1 is intended to serve as a conceptual aid that classifies the performance measures in a general, overall sense. An overly rigid interpretation should, however, be avoided. The surveillance and operational environment of a major airport is very complex and cannot easily be reduced to ultimate simplicity. Still, if it is understood that borderline cases and the occasional crossover may be encountered, the table places the measures in a useful perspective. For example, the primary safety logic performance measures are interference and light infringements, although the safety logic could also be responsible for an occasional missed detection. Low interference is a requirement for a transparent system, while the ultimate proof of the safety logic's effectiveness is its proper operation in a potentially unsafe situation, as demonstrated by a light infringement.

The performance measures will be discussed briefly and illustrated with the aid of examples. Further discussion will be deferred to Section 9.5, where these issues will come up again in connection with the data analysis. The narrative will for simplicity refer to aircraft but the arguments apply equally to surface vehicles, where appropriate.

A missed detection may be defined in a narrow or broad sense. The narrow definition applies to a failure to detect a developing conflict situation: this requires both that the light was off when it should have been on, and that a conflict developed. Since runway conflicts are exceedingly rare events, we do not expect to see evidence of this type of missed detection in the limited amount of traffic used for this assessment. The broad definition is a missed detection of a potential conflict situation; this merely requires that the light was off when it should have been on, and that the traffic picture was such that this failure could have permitted a conflict to develop. The broad definition is used here. There is an even broader definition of a missed detection, namely, that the light was off when it should have been on, regardless of any potential for conflict. This type of missed detection is only associated with the runway-entrance lights (RELs). It will be discussed below under the designation Type 2 REL missed detection.

A missed detection of a condition that should have activated a takeoff-hold light (THL) could be caused by either a track drop of an aircraft in or about to enter the light's arming region, or drop of a target in or about to enter the activation region, or a failure to track or project crossing high-speed traffic properly. It could also in principle, but very rarely in practice, be due to multiple causes. An untracked aircraft holding in position in an arming region is one of the more likely causes of missed detections, since such a situation may persist for a while and possibly give rise to several missed detections.

A missed detection of a condition that should have activated a runway-entrance light could be caused by a track drop of an aircraft engaged in high-speed operations, such as arrival, landing, landing rollout, or takeoff. Such a drop may give rise to a light outage ranging in duration from very brief to tens of seconds. Only the longer outages could possibly have a direct safety impact, since crews and aircraft could not respond to very brief outages. Another cause of missed detections is ambiguity with regard to the landing runway: approaches are flown at Logan that sometimes make it difficult for the safety logic to automatically determine the landing runway from surveillance alone. The logic may be late in making the

determination, with the result that the runway-entrance lights near the approach end of the landing runway illuminate later than desired.

Runway entrance lights illuminate whenever activated, regardless of whether anybody is in a position to see them. This is unlike the takeoff-hold lights, which must be armed as well as activated before they illuminate. The armer is generally in a position to see the illuminated takeoff-hold light, since the arming region is defined specifically to include the holding positions of aircraft positioned for takeoff. With the runway-entrance lights it is necessary to make a distinction between observed and unobserved illuminations. The observed illuminations are called Type 1, the unobserved Type 2. This assessment counts only Type 1 anomalies. This is consistent with the focus of the assessment, which is on the interaction between the lights and pilots or surface-vehicle operators.

Most missed detections are due to surveillance processing or tracking difficulties or prediction ambiguities that require knowledge of controller, pilot or operator intent. In specific airport regions, missed detections can be traced to surveillance limitations such as shadowing by buildings or minimum-range settings in the terminal radar receiver that are not fundamental in nature. It is of course also possible that a missed detection might arise from a failure of the safety logic algorithms or software to accurately reflect the intent of the safety logic. Such problems are rare; they will if necessary be discussed in Section 9.5.

A false alarm is an illumination of a status light that does not reflect real traffic. A false illumination of a takeoff-hold light might come about due to a false activation track and a real arming track, or a false arming track and a false or real activation track. The alarm caused by false activation is called a Type 1 false alarm, an alarm caused by false arming is called a Type 2 false alarm. They differ in that the crew in the arming region is in a position to observe a Type 1 false alarm, whereas there is generally no one around to observe a Type 2 false alarm (or, if there is someone nearby, the observer is likely to be less concerned). The Type 1 false alarm is therefore the more significant and the only type to be considered in this assessment. The definition of a Type 1 false alarm makes no distinction on the basis of the activity or identity of the arming region occupant. It could, for example, be an aircraft holding in position or crossing the runway slowly, or it could be a maintenance vehicle in an arming region on an inactive runway.

Note that if lights were installed on the airport surface, a Type 1 false alarm might actually cause interference by interrupting or delaying a departure or by prompting a pilot to request verification of a clearance. We nevertheless make the distinction between Type 1 false alarm and interference, since it is an important one for the purposes of this assessment.

A false REL illumination could be caused by a false high-speed track on approach or on the runway, due to multipath or misassociation. A surface vehicle traveling at high speed on the runway might also activate the runway-entrance lights, but this would not constitute a false alarm, since the traffic was real; likewise with a helicopter approaching the helipad but erroneously identified as a circling approach to 4L. Such instances may or may not be classified as interference, depending on the circumstances. Only observed false REL illuminations will be counted, reflecting the point of view that instances where a runway-entrance light illuminates an empty scene are of no consequence to the system's end users. By analogy with the takeoff-hold lights, an observed false REL illumination will be called a Type 1 false alarm and an unobserved one a Type 2 false alarm.

Interference, according to the definition, is caused by real traffic. It comes about mostly when the safety logic does not take full account of the operational realities of the airport environment, or where the safety logic is handicapped by its lack of knowledge of pilot and controller intent. An example is a runway crossing in front of aircraft on landing rollout. The safety logic's state machine currently does not

have a landing-rollout state, and this causes the runway-entrance lights to illuminate too far ahead of the aircraft, with the result that the lights interfere with safe crossing operations further down the runway. Another potential interference situation arises when an aircraft uses a runway to taxi to another runway for departure. If this aircraft enters an arming region at taxi speed when the runway ahead is, or is predicted to be, occupied, the takeoff-hold light will turn on even though the taxiing crew has no intention of taking off. Whether this would be interpreted as interference in an operational situation is unclear at this point.

Interference by a runway-entrance light may be due to misprojection of airborne traffic, leading the safety logic to conclude that a landing is imminent on a given runway when it in fact is not. But occasionally the runway-entrance light is on due to an approach that develops into a late sidestep to another runway. This is not classified as interference, since the projection was correct up to the moment the sidestep was initiated.

A Light Infringement differs from interference mainly in that the light infringement results in an apparently unsafe situation. It is difficult to determine from the recorded surveillance data if safety was in fact compromised, since no record was made of tower channel exchanges during the data collection, nor were video recordings made. This makes it impossible to offer a definitive assessment of apparently unsafe traffic situations. No light infringement was seen in the data. This was not unexpected, since such events are exceedingly rare. A search for light infringements was nevertheless part of this assessment, since such an event would provide direct proof of the potential contribution of the RSLS to runway safety.

9.3.3 Normalization

To be meaningful, the anomaly counts must be presented in a way that takes the traffic volume into account, such as, for instance, false alarms per hundred operations. This normalization scheme is simple and easy to understand, since it in effect gives a measure of the number of anomalies to be expected in a given amount of time (at Logan, 100 operations correspond to about an hour of traffic under average conditions).

Suggesting that there is a simple relationship between anomalies per hundred operations and anomalies per hour amounts to a statement about linearity. It is probably not true that the average number of anomalies per operation is completely insensitive to the density of the traffic in the underlying data. Since the assessment looks for observed anomalies, the events counted call for one light activator and one observer, that is, they imply interaction between two agents and thus a potential density dependence. (The same is, appropriately, true for the problem the RSLS is designed to address, namely, runway incursions.) If such quadratic effects are present, they were not discernible in the data used for this assessment, perhaps in part because of the limited size of the data sample, but also because the data were intentionally collected under traffic conditions that were representative of typical Logan traffic conditions, which means that more data were collected in average conditions than under extremely low and extremely high traffic loads.

Since an assumption of linearity appears to be warranted in the context of this assessment, the results will be presented in Section 9.5 in terms of anomalies per hundred operations. But such a normalization fails to distinguish between long and short anomaly events, counting all with equal weight. It therefore gives no clue to another important performance indicator, viz., the fraction of time that crews operating on the airport surface might expect to be confronted with a light anomaly. This is called the anomaly fraction. The results will also be presented in terms of the anomaly fraction, so as to take account of the duration of the anomalies.

Whether presented in terms of anomalies per hundred operations or in terms of the anomaly fraction, the assessment results imply a survey of the entire airport. That is, "five anomalies per hundred operations" means that five anomalies are, on the average, observed by the "pilot population" on the airport in the course of one hundred operations. One hundred operations represent approximately one hour under average traffic conditions at Logan. Thus, we may say that five anomalies are experienced, airport-wide, in the course of one average hour. An individual crew can also expect to encounter five anomalies in the course of one hundred operations but, in general, considerably less than five anomalies in one hour of cumulative taxi time. Likewise, if the anomaly fraction is 1%, this means that the pilot population will, on the average, see anomalies totaling (or an anomaly lasting) 36 seconds in the span of an hour. An individual crew will be less affected.

9.4 DATA COLLECTION

The RSLS was in active development through February 1993, and it was necessary to defer data collection for the performance assessment until the radar processing and tracking software stabilized sufficiently that the assessment could be viewed as a valid point-in-time description. The data were gathered in a brief but concentrated data collection effort during March and April of 1993; reflecting predominantly late-winter and spring conditions. In spite of the short period available for data recording, all major operational configurations and most common weather and traffic conditions were successfully captured. Altogether, more than 5000 operations were recorded. A smaller but well-balanced subset of approximately 800 operations is used for this assessment.

9.4.1 Logan Airport Operational Configurations

Logan Airport is usually operated in one of four major operational configurations, the choice dictated primarily by wind conditions, but also by ceiling and visibility. Other, more restrictive, configurations are employed in special circumstances, notably during snow removal and runway maintenance operations. The major configurations are described briefly below, with reference to the airport map of Figure 1-9, repeated here as Figure 9-1 for convenience. Deviations from the usage indicated may occur upon pilot request and at the discretion of the local controller.

4/9 Configuration: Arrivals to Runway 4R, departures from Runway 9, arrivals and departures to/from Runway 4L, occasional departures from Runways 4R and 15R. This configuration is used when the wind is from the NE or E, as is commonly the case during the winter and spring months, or during deteriorating weather conditions.

22/27 Configuration: Arrivals to Runways 22L and 27, departures from Runway 22R, occasional departures from Runway 22L, occasional arrivals to Runway 22R. This is generally a summer configuration, used when the wind is from the SW.

33/27 Configuration: Arrivals to Runways 33L, 27, and 33R, departures from Runway 27. Non-jet departures from Runway 33L at Taxiway G and occasional full-length jet departures from Runway 33L. This is generally a winter configuration, used when the wind is from the NW.

15/9 Configuration: Arrivals to Runways 15R and 15L, departures from Runway 9, occasional departures from Runway 15R. This configuration, used in SE airflow, is seen less often than the other three.

Over a period of one year, one can expect to encounter each of the first three configurations approximately 30% of the time, the fourth approximately 10% of the time.

9.4.2 Data

The data samples, or blocks, used for this assessment are listed below by time of day, regardless of date and configuration. Thirty percent were recorded in IMC and 70% in VMC. There are approximately three hours each of the 4/9 and 22/27 configurations and two hours each of the other two. The data are thus slightly overweighted in the 15/9 configuration, at the expense of the 33/27 configuration, when compared to the annual statistics. Most hours of the operational day are represented, but early morning and late evening and a period around noon are missing. All in all, the data represent a brief but well-balanced glimpse of operations at Logan Airport.

Data Block	Time Interval (local time)	Config.	IMC?	Net Time (minutes)	Operations
9	7:48-8:44	4/9	yes	56	70
10	9:03-10:02	22/27		56	87
7	9:44-11:00	15/9	yes	70	55
2	10:33-11:36	33/27		60	94
4	13:12-14:12	15/9		52	63
5	14:50-16:10	33/27		80	99
1	15:59-17:14	4/9	yes	75	91
3	17:34-18:34	22/27		60	103
6	19:15-20:20	4/9		58	91
8	19:51-20:47	22/27		54	75

These data blocks capture weather conditions ranging from sunny and calm to foggy, to rainy and windy, to the early part of the historic blizzard of March 13, 1993. The traffic intensity ranges from moderate to heavy, approaching 110 operations per hour during parts of Data Block 3.

The discrepancy between net block time and the total block interval is due to tape errors in the recorded raw ASDE-X data, which made an occasional short segment of the recorded data unusable for the analysis.

The data collection sessions did not include tower channel voice recording, nor were video recordings made of the airport traffic, but notes were made as time allowed when events of potential interest to the safety logic were observed. Although sufficient information was collected to permit a thorough assessment of the technical performance of the runway status lights, the lack of a complete record of the operations precludes definitive interpretation of the recorded events from an operational point of view.

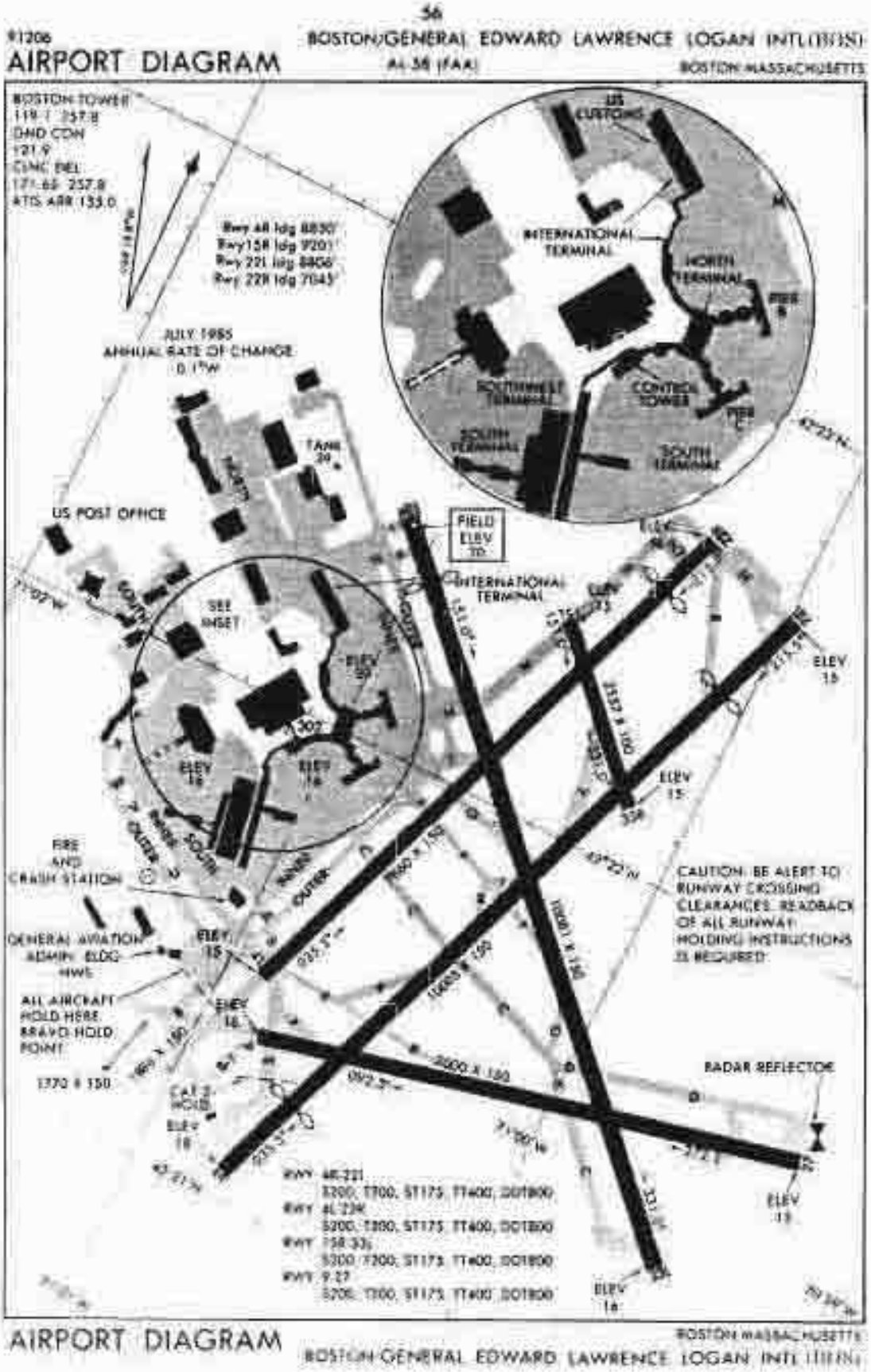


Figure 9-1. Boston Logan airport runway and taxiway map.

9.5 DATA ANALYSIS

The 10 data blocks of Section 9.4 were analyzed to identify all light anomalies that satisfied a prescribed set of inclusion criteria. These criteria, or counting rules, are described in Section 9.5.1. This is followed by the assessment results, presented for each configuration individually in Sections 9.5.2 through 9.5.5. These sections, besides identifying the anomalies, also discuss the apparent causes and possible fixes. The descriptions of causes serve to illustrate the complexity of the airport surveillance environment and the variety of challenges presented to the RSLS by the physical and operational environment at Logan airport. The descriptions also serve to identify areas of high payoff in future development efforts.

To simplify the presentation, the narrative will generally be presented as if the lights were actually installed on the field. That is, the discussions refer to lights illuminating "in front of aircraft," "causing interference," etc. It must be kept in mind, however, that there are currently no runway status lights at Logan. Neither field nor tower operations at Logan are in any way affected by the RSLS in its present form.

9.5.1 Counting Rules

The counting conventions are consistent with the assessment's objective, which is to characterize the system from the user's perspective. That means answering questions about how the RSLS will affect pilots during ground operations. The main task of the assessment is to find answers to the following two questions:

1) How many light anomalies will be observed in an average hour of operations at Logan, by any and all crews operating on the surface?

2) What is the likelihood that, at any given moment, some crew will experience a missed detection or a false alarm?

Answering these questions requires not only an assessment of the operation of the lights themselves, but also an evaluation of the probability that an anomaly will be seen. The second piece of information can only be obtained by observing the interplay between the lights and the traffic. It can not be determined by assessing the functional elements (radar processing and tracking, sensor fusion, or safety logic) of the RSLS in isolation.

This is not to say that evaluating the individual functional elements is inappropriate; on the contrary, these more fundamental modes of evaluation are at least as important as the evaluation presented here, if the goal is a thorough understanding of the performance of the RSLS. But they cannot provide the "bottom-line" answer, and for that reason they properly belong with the individual functional elements of the system, and not in an end-to-end analysis such as the present one.

In keeping with the intent to evaluate the system from the user's perspective, the assessment counts only Type 1 false alarms and missed detections, that is, anomalies that would have been observable from the cockpit. This is a straight-forward concept for the takeoff-hold lights: it simply requires that an aircraft is positioned in the light's arming region. Although in principle not much more complicated for the runway-entrance lights, the definition is slightly more involved, since there is no well defined region in which to look for an observer. The assessment identifies observed REL false alarms and missed detections by the following three criteria: to be counted as a Type 1 (observed) REL missed detection or false alarm, an aircraft 1) must be within a certain distance of the light during the anomaly, 2) it must be

first in line, and 3) it must eventually enter the runway at that location. The first two criteria address the question of whether the pilot is likely to be looking at the light, the second whether he is likely to be concerned with it. These criteria may suggest that the identification of Type 1 REL anomalies is a difficult process, fraught with uncertainty, but as a practical matter, the flow of taxi traffic is such that little ambiguity arises.

A long anomaly will tend to be more of a concern than a short one. A short anomaly, such as a one-scan light outage, may of course be a cause for concern for the pilot - if it is noticed - depending on the circumstances. But this is mainly a human factors question and thus outside the scope of this assessment. In terms of direct consequences, clearly neither crew nor aircraft can respond to such a short outage. The assessment recognizes this fact by treating one-and-two scan false alarms and missed detections differently from the longer ones. All false alarms and missed detections are of course undesirable and should be eliminated as much as possible in future development efforts.

Instances of interference are, by definition, all observed. According to the definition in Section 9.3, interference requires actual or imminent crossing of the light threshold while the light is on. All instances of interference are counted, with no distinction attempted on the basis of duration, except for a qualitative judgment about severity.

When surface vehicles cause light anomalies that affect aircraft, they are counted as any other. REL anomalies that affect surface vehicles are also counted. But the operation of takeoff-hold lights that are armed by, for instance, service vehicles parked on an inactive runway is not assessed. The affected takeoff-hold light sometimes illuminates due to normal airport traffic, but such illuminations are considered inconsequential since a takeoff by the surface vehicle is out of the question.

9.5.2 The 4/9 Configuration

Data Blocks 1, 6, and 9 were recorded in the 4/9 configuration, Data Block 6 in VMC, the other two in IMC. The net duration was 189 minutes. There were 115 arrivals and 137 departures, for an average traffic load of 80 operations per hour. Table 9.2 summarizes the three data blocks individually as well as the combined results for this configuration.

Generally, in the 4/9 configuration, most departures are from Runways 9 and 4L, with arrivals to Runway 4R and, conditions permitting, 4L. It is a challenging configuration for the RSLS, for several reasons. First, most of the arrivals land on Runway 4R and must cross Runway 4L inbound on their way to the gate. Likewise, many of the departures on Runway 9 taxi out via Taxiway S, crossing Runway 4L outbound at the approach end. Careful tuning of the safety logic is required to avoid interference, especially in heavy traffic under VMC. Second, Taxiway S can become very congested, both in the vicinity of Runway 4L and between Runways 4L and 9. This, along with the radial orientation of the taxiway with respect to the radar, leads to a challenging surveillance environment, in that shadowing effects add to the difficulties of separating and tracking the closely spaced aircraft. Third, there is sometimes a profusion of multipath returns along the first half of Runway 4L and the first quarter of Runway 9, especially near the 4L/S and 9/S intersections. This increases the probability of track drop by misassociation with multipath, in areas already made challenging by the orientation of the taxiway and traffic congestion. Despite these difficulties, Table 9.2 indicates very good performance, especially in Data Blocks 6 and 9.

TABLE 9.2

Light Anomalies for Configuration 4/9 at Logan

Data Block	Conditions	Duration (min)	Arrivals	Departures	Ops/hr	Light Anomalies MD*	FA*	I	
1	IMC	75	39	52	73	6	0	0	THL
						0	0	3	REL
6	VMC	58	46	45	94	1	0	0	THL
						1	0	1	REL
9	IMC	56	30	40	75	1	1	0	THL
						0	0	0	REL
Totals or Averages	—	189	115	137	80	8	1	0	THL
						1	0	4	REL

* Nine missed detections and four false alarms of one or two scan duration are not included in the table.

9.5.2.1 Takeoff-hold Light Performance

There were 86 departures from Runway 9, 41 from Runway 4L, four from Runway 4R, five from Runway 15R, and one from Runway 33L. All were full-length departures, except for one that was made from Runway 4R at Taxiway S.

There were two false alarms and 10 missed detections in Data Block 1, one false alarm and one missed detection in Data Block 6, and two false alarms and three missed detections in Data Block 9. There was no interference.

False Alarms

The five false alarms were all brief: three of one-scan duration and one each of two and three-scan duration. All were caused by false (multipath) targets appearing briefly on Runways 4L and 4R in front of aircraft positioned for departure. The one-and-two-scan false alarms are judged too brief to have operational impact and are therefore excluded from Table 9.2. The longest of the five false alarms is included, since, at three scans (five seconds), it would more likely have been a cause of concern for the crew. The multipath rejection algorithms are still being refined, and it is expected that the number of false tracks caused by multipath will be reduced by more sophisticated multipath filters than those currently in use. This will reduce the incidence of false alarms caused by multipath.

Missed Detections

All but one of the fourteen missed detections occurred on Runway 9, the remaining one on 4L. Nine (all brief, lasting between one and three scans) were caused by difficulties in tracking arrivals to Runway 4R (which intersects Runway 9) over the approach-light pier that extends into the water off the approach end of Runway 4R. This pier represents a difficult high-clutter surveillance environment where further improvements are needed in both surveillance and tracking algorithms. Compounding the difficulties is

the fact that ASR returns from aircraft within approximately 0.8 nm of the ASR are usually suppressed and therefore not available to the RSLS. This surveillance limitation is not fundamental and could be eliminated in a future system at Logan. It is, however, a feature of the current system, and it necessitates coasting ARTS targets in the surveillance gap -- in particular, those on approach to Runway 4R -- through the gap. Sometimes the position error accumulated during this coast period is sufficiently large to preclude proper fusion with the ASDE return once inside the region of ASDE coverage. The result is an occasional (usually brief) track drop and THL deactivation.

The remaining five missed detections, lasting between four and 32 scans, were, unlike the nine just described, due to THL disarming, caused by track drop while holding in position on the runway. Four occurred on Runway 9 and one on Runway 4L. In at least two cases (including the 32-scan outage) the problem appears to have been misassociation with multipath. Such misassociations present a difficult surveillance and tracking problem and a high-priority item in future development work. A recovery feature may be added to the tracking software that will recognize when such a misassociation has occurred and attempt to re-establish the broken track.

The six one-and-two-scan outages are too brief for crews and aircraft to respond to them and are therefore excluded from Table 9.2. The eight longer outages are included, even though, at three scans, it can be argued that the three shortest of them were also too short for the crew and aircraft to respond and move past the takeoff-hold light before it was reactivated (assuming, of course, that they were erroneously cleared or started to take off without clearance).

It is not possible to evaluate the performance of the takeoff-hold light for the five departures from the Runway 15R full-length departure location, due to inadequate radar surveillance: the location of the X-band radar on the roof of the old tower building places it just barely high enough to see over Terminal E. The result is that even large aircraft enter the runway very soon after emerging from the radar shadow and thus are not in track until after they begin their takeoff roll. The resulting missed detections are not counted since they are an unavoidable consequence of the suboptimal positioning of the radar.

9.5.2.2 Runway-entrance Light Performance

There were 95 arrivals to Runway 4R and 20 to Runway 4L, in addition to the above-mentioned departure activity. There were four missed detections and four instances of interference. There was no false alarm.

Interference

Three of the four instances of interference occurred in Data Block 1. They were all associated with a surface vehicle doing a sweep of Runway 22R. It entered the runway at Taxiway N while the runway-entrance lights were still on due to a 4L departure. This aircraft was at a safe altitude and the runway-entrance lights would ordinarily have been off. There was not yet an altitude report for this aircraft, however, (perhaps because the crew had not turned the transponder on yet) so the safety logic had not declared the aircraft safely airborne.

After crossing the lights, the vehicle proceeded down the runway, reaching a speed and acceleration sufficient to be declared a departure and illuminating the runway-entrance lights. Two aircraft crossed against the lights, a safe distance in front of the vehicle. Further tuning of the state transition thresholds in the safety logic will reduce the chances that a speeding surface vehicle is misidentified as a departing

aircraft, but such misidentifications probably cannot be eliminated altogether, without some way to positively identify vehicles that are likely to be engaged in high-speed operations on the runways.

The fourth instance of interference occurred in Data Block 6. It involved a 4L departure and an inbound crossing at Taxiway N. In this case the airborne aircraft did report its altitude, but the altitude threshold used by the safety logic to deactivate the runway-entrance lights was set too high, causing the lights to turn off late. The interference was mild.

Missed Detections

Two of the missed detections occurred in Data Block 6 at the intersection of Taxiway S and Runway 4L. They were brief, lasting one and three scans. Both were associated with curved approaches to 4L: the tracking filter applied in sensor fusion to the ARTS returns caused the track to deviate to the outside of the turn, resulting, ultimately, in a failure to project the arrival onto the runway. This occurred just before the ASDE had the aircraft in track, hence the short duration of the outages. An adjustment to the filter gains and/or the ASDE coverage map will alleviate this problem. The other two missed detections, one each in Data Blocks 1 and 9 and both lasting two scans, were caused by late declarations of departures on Runway 4L, resulting in delayed activation of the runway-entrance light at the intersection with Taxiway C. As discussed above, only the one missed detection longer than two scans is included in the anomaly count of Table 9.2.

9.5.3 The 22/27 Configuration

Data Blocks 3, 8, and 10 were recorded in the 22/27 configuration, all in VMC. Net duration was 170 minutes. There were 119 arrivals and 146 departures, for an average traffic load of 94 operations per hour. Table 9.3 summarizes the three data blocks individually as well as the combined results for this configuration.

TABLE 9.3

Light Anomalies for Configuration 22/27 at Logan

Data Block	Conditions	Duration (min)	Arrivals	Departures	Ops/hr	Light Anomalies MD*	FA*	I	
3	VMC	60	47	56	103	5	0	0	THL
						0	0	0	REL
8	VMC	54	39	36	83	1	0	1	THL
						0	0	1	REL
10	VMC	56	33	54	93	1	1	0	THL
						0	0	0	REL
Totals or Averages	—	170	119	146	94	7	1	1	THL
						0	0	1	REL

* Two missed detections and seven false alarms of one or two scan duration are not included in the table.

Generally, in the 22/27 configuration, most departures are from the full-length position of Runways 22R and 22L, with arrivals to Runways 27 and 22L. As with the 4/9 configuration, it is a challenging configuration for the RSLS. Most of the arrivals land on Runways 27 and 22L and must cross the primary departure Runway, 22R, on their way to the gate. The majority cross at Taxiways E and S. Careful tuning of the safety logic is required to avoid interference at these busy crossing points. Table 9.3 indicates very little interference and, except for a number of missed detections in Data Block 3, excellent performance overall.

The potential of the RSLS to protect against runway incursions without causing interference can be seen clearly in Data Block 3: there were 44 departures from Runway 22R in a span of 60 minutes, while 34 arrivals crossed inbound at Taxiways E and S. Between them, these 34 crews saw a total of 57 REL illuminations. None experienced interference.

9.5.3.1 Takeoff-hold Light Performance

There were 128 departures from Runway 22R and 16 departures from Runway 22L, as well as two departures from Runway 15R in Data Block 8.

The takeoff-hold lights performed well, especially in Blocks 8 and 10, less so in Data Block 3. There was one instance of interference in Data Block 8. There were six false alarms, three in Data Block 3, one in Data Block 8, and two in Data Block 10, all but one of short duration. There were seven missed detections, lasting from three to 37 scans, five in Data Block 3 and one each in Data Blocks 8 and 10.

Interference

The one case of interference occurred on Runway 22R and was caused by strong headwind: the previous departure was climbing out steeply, at a ground speed of 68 kts and an altitude of 600 ft over Taxiway Q. The safety logic misclassified this aircraft as a departure abort and did not deactivate the takeoff-hold light as it would ordinarily have done - under VFR conditions - when a safe separation was achieved. Modification of the safety logic's state-transition rules is expected to reduce the likelihood of such misclassifications.

In addition to the one case of interference in Data Block 8, there were three instances of THL interference on Runway 22L in Data Block 3, caused by land-and-hold-short operations on Runway 27. The safety logic currently has no provision for handling this condition, so these instances of interference will not be counted. Future additions to the safety logic will include a special algorithm to be invoked when land-and-hold-short operations are in effect, to shorten the duration of THL activation on the intersecting departure runway and thus eliminate the interference with departure operations. This algorithm will include information on the type or category of the arriving aircraft as well as provision for controller input, so as not to reduce the effectiveness of the takeoff-hold light when the landing aircraft is not expected to hold short.

False Alarms

Of the six false alarms, four were due to multipath appearing on Runway 22R while an aircraft was in position for departure. Three of these were brief (one and two scans), as is usually the case when multipath is involved, but one, in Data Block 10, lasted 10 scans. The multipath rejection algorithms are still being refined, and it is expected that the number of false tracks caused by multipath will be reduced

by more sophisticated multipath filters than those currently in use. This will reduce the incidence of false alarms caused by multipath.

Another false alarm was due to a false high-speed track that appeared over the water, presumably caused by radar returns from the choppy sea. It was misidentified as approaching traffic and projected to Runway 4L, illuminating the takeoff-hold light for two scans in front of an aircraft holding in position at the opposite end of the runway. Modifications to the safety logic may be implemented that would delay slightly the projection of certain suspicious tracks that appear over the water with no ASR counterpart. This should eliminate such wave-action effects with minimal reduction in system effectiveness.

The last false alarm was caused by a fusion failure during departure roll, which resulted in a one-scan false illumination of the takeoff-hold light in front of the next departure, which was holding in position. The five one-and-two-scan false alarms are judged too brief to have operational impact and are therefore excluded from Table 9.3. Only the 10-scan false alarm is included.

Missed Detections

The seven missed detections, lasting between three and 37 scans, were all long enough to be included in the anomaly count in Table 9.3. The longest was a 37-scan outage in Data Block 8 that resulted when an aircraft in position for departure on Runway 22R suffered a track drop and thus disarmed the takeoff-hold light. During this time the aircraft that had been ahead of it in the departure queue was returning to the gate, taxiing down the runway. This resulted in a rather long missed detection. There were three other missed detections caused by track drops while in position on Runway 22R; they lasted three, six, and eight scans. The problem of occasional track drops in the vicinity of the intersection of Runway 22R and Taxiway N is under study. It may in part be related to deficiencies in one of the track-drop recovery features. Correcting this problem has high priority.

Two other missed detections, in Data Blocks 3 and 10, occurred when small aircraft suffered track drops while waiting to cross Runway 22R at Taxiway E. They crossed while not in track, thus failing to activate the Runway 22R takeoff-hold lights and causing missed detections of eight and three scan duration. Failed reacquisition was the cause in both cases; the bad-drop reacquisition algorithm will be improved.

The last missed detection, of estimated duration 11 scans, occurred in Data Block 3: an arrival to Runway 27 failed to activate the takeoff-hold light on the intersecting Runway 22L as early as it should have. The problem has been traced to the safety logic software and a fix has been identified.

It is not possible to evaluate takeoff-hold-light performance for the two departures from the Runway 15R full-length departure location due to surveillance limitations, as mentioned in Section 5.2.1. The resulting missed detections are not counted since they are an unavoidable consequence of the suboptimal positioning of the radar.

9.5.3.2 Runway-entrance Light Performance

There were 67 arrivals to Runway 27, 50 to Runway 22L, and two to Runway 22R, in addition to the above-mentioned departure activity. The runway-entrance lights gave rise to one instance of interference in Data Block 8, and there were two brief false alarms and one brief missed detection in Data Block 3. No REL anomaly was seen in Data Block 10.

Interference

The one case of interference occurred at the intersection of Runway 22R and Taxiway S: an aircraft was on landing rollout on Runway 22R and still moving fast enough to activate the runway-entrance lights at Taxiway S when a prior arrival crossed inbound. There was no danger in this situation. Planned enhancements to the safety logic include the addition of a landing-rollout state, which would have the effect of reducing the likelihood of interference in such situations. Additional tuning of the REL activation time for the landing state would be an alternative, but less fundamental, solution.

False Alarms

The two false alarms, lasting one and two scans, occurred at Taxiways E and S on Runway 22R. They were caused by false tracks due to wave action in the harbor channel, as described above in connection with the takeoff-hold lights.

Missed Detection

The one-scan missed detection occurred at the intersection of Runway 27 and Taxiway C. It was caused by a brief velocity error suffered by an arrival to Runway 27, which caused the runway-entrance light at 27/C to go out momentarily. This missed detection as well as the two false alarms are judged too brief to have operational impact and are therefore excluded from Table 9.3.

9.5.4 The 33/27 Configuration

Data Blocks 2 and 5 were recorded in the 33/27 configuration, both in VMC. Net duration was 140 minutes. There were 113 arrivals and 80 departures, for an average traffic load of 83 operations per hour. Table 9.4 summarizes the two data blocks individually as well as the combined results for this configuration.

Generally, in the 33/27 configuration, most departures are from the full-length position of Runway 27 and from Runway 33L at Taxiway G, with most arrivals to Runways 33L and 27. Data Block 2 provides a good example of 33/27 operations: a significant number of arrival and departure operations were conducted on both of the intersecting runways 27 and 33L, in addition to a small number of arrivals to Runway 33R. The traffic was heavy, reaching 100 operations per hour at its peak. Many of the runway status light channels were very active, demonstrating their potential contribution to safety under conditions of heavy and complex traffic. For instance, there were 16 departures from Runway 33L at Taxiway G. Thirteen of these saw a total of 24 REL illuminations and 11 saw illuminated takeoff-hold lights. Nine of the departures saw both kinds of lights illuminated between the time they stopped on Taxiway G short of Runway 33L and the time they departed. Only one departure out of the 16 took off without light involvement. Yet, there was only one case of mild interference.

9.5.4.1 Takeoff-hold Light Performance

There were 37 full-length departures from Runway 27 and two departures from 27 at C, as well as eight full-length departures from Runway 33L and 33 departures from 33L at G. The takeoff-hold lights performed very well. There was one instance of interference in Data Block 2 and four instances of missed detections in Data Block 5.

TABLE 9.4

Light Anomalies for Configuration 33/27 at Logan

Data Block	Conditions	Duration (min)	Arrivals	Departures	Ops/hr	Light Anomalies MD*	FA*	I	
2	VMC	60	57	37	94	0	0	1	THL
						0	0	5	REL
5	VMC	80	56	43	74	3	0	0	THL
						0	1	1	REL
Totals or Averages	—	140	113	80	83	3	0	1	THL
						0	1	6	REL

* One missed detection of one scan duration is not included in the table.

Interference

The one case of interference occurred on Runway 33L at G, when the takeoff-hold light failed to anticipate that a prior arrival was vacating the runway. An extension to the safety logic's anticipation features is planned, to permit earlier deactivation of the takeoff-hold light in similar situations and reduce the likelihood of such interference.

Missed Detections

Of the four missed detections, one was a 4-scan outage of the takeoff-hold light at the full-length departure position on Runway 27 and one was a one-scan outage. Both were caused by difficulties in tracking arrivals to the intersecting Runway 33L over the approach-light pier that extends into the water off the approach end of Runway 33L. This pier represents a difficult high-clutter surveillance environment where further improvements are needed in both surveillance and tracking algorithms.

The one-scan outage is far too brief for crews and aircraft to respond to it and is therefore excluded from the missed-detection count in Table 9.4. The longer of the two outages is included even though, at four scans (7 seconds), it can be argued that it was also too short for the crew and aircraft to respond and move past the takeoff-hold light before it was reactivated (assuming, of course, that they were erroneously cleared or started to take off without clearance).

The remaining two missed detections, lasting six and 10 scans, respectively, occurred on Runway 33L at G. They were both caused by track drops suffered by traffic crossing Runway 33L at Taxiway N: two aircraft that had previously landed on Runway 33R were apparently misassociated with multipath returns as they waited to cross Runway 33L and subsequently crossed while not in track. These aircraft, temporarily invisible to the safety logic, failed to activate the takeoff-hold light lights which therefore did not illuminate as they should have in front of aircraft holding in position on Runway 33L at G. Planned enhancements to the scan-to-scan association algorithms include a misassociation recovery feature that would lessen the negative impact of such misassociations.

9.5.4.2 Runway-entrance Light Performance

There were 16 arrivals to Runway 27, 16 to 33R, and 81 to 33L, in addition to the above-mentioned departure activity. The heavy activity on Runway 33L resulted in extremely active runway-entrance lights on this runway, especially beyond the Taxiway F turnoff, where the channels were activated not only by the arrivals but also by intersection departures from Taxiway G in addition to those from the full-length position. There were at least 57 REL activations along this segment of 33L in Data Block 2 and 65 in Data Block 5 -- that is, almost one per minute on the average.

There were five instances of interference in Data Block 2 and one in Data Block 5, as well as one false alarm in Data Block 5.

Interference

The case of interference seen in Data Block 5 occurred when an aircraft that had landed on Runway 33R was crossing Runway 33L at Taxiway N. An aircraft on landing rollout on Runway 33L was still moving fast enough to activate the runway-entrance lights at Taxiway N, causing interference with the crossing aircraft. There was no danger in this situation and the interference was mild. Planned improvements to the safety logic include the addition of a landing-rollout state, which would have the effect of reducing the likelihood of interference in such situations. Additional tuning of the REL activation time for the landing state would be an alternative, but less fundamental, solution.

Of the five cases of REL interference seen in Data Block 2, two were of the type just described. The other three were on Runway 27, caused by misprojections of 33R arrivals to Runway 27. These misprojections were brief and the interference mild. It may prove difficult to eliminate this kind of interference completely, since 33R arrivals often appear lined up with Runway 27 until quite late in the approach. The approach logic for the 33/27 configuration will undergo additional tuning, however, to reduce the frequency of such misprojections. Provisions for controller input will also be considered.

False Alarms

The one false alarm occurred on Runway 4L at Taxiway S: one of only two arrivals to Runway 27 in Data Block 5 was taxiing in on Taxiway X when the Runway 4L runway-entrance lights illuminated erroneously for 8 scans. The reason was an erroneously coasted track in the vicinity of the GA ramp that was misidentified as an arrival to 4L. Modifications to the tracker's coast feature are expected to reduce the likelihood of such false alarms.

9.5.5 The 15/9 Configuration

Data Blocks 4 and 7 were recorded in the 15/9 configuration, the first in VMC, the second in IMC. Net duration was 122 minutes. There were 61 arrivals and 57 departures, for an average traffic load of 58 operations per hour. Data Block 7 was recorded during the major blizzard of March 13, 1993. The storm intensified during the recording period, the visibility deteriorating to about one mile and the wind increasing to 15 kts. As would be expected, radar surveillance was not significantly affected by the snow, but the 15-kt ESE wind caused considerable wave action in the harbor channel adjacent to the airport and this did produce a number of short-lived high-speed false tracks over the water. These false tracks did not have a significant effect on the system, which performed very well overall. Table 9.5 summarizes the two data blocks individually as well as the combined results for this configuration. Note that the pilots operating on the airport during Block 7, in increasingly severe blizzard conditions, would have found the runway-status lights performing flawlessly -- they would have experienced no missed detection, no false alarm, and no interference.

TABLE 9.5

Light Anomalies for Configuration 15/9 at Logan

Data Block	Conditions	Duration (min)	Arrivals	Departures	Ops/hr	Light Anomalies			
						MD	FA	I	
4	VMC	52	27	36	73	1	0	0	THL
						0	0	2	REL
7	IMC	70	34	21	47	0	0	0	THL
						0	0	0	REL
Totals or Averages	—	122	61	57	58	1	0	0	THL
						0	0	2	REL

In the 15/9 configuration, Runway 9 is the primary departure runway, with additional departures from Runway 15R. Runways 15R and 15L are used for arrivals. When conditions and aircraft category permit, aircraft landing on Runway 15R are issued clearances to land on 15R and to hold short of the Runway 9 intersection; under these conditions takeoff clearances may be issued for Runway 9 while the landing aircraft on Runway 15R is still rolling out. This was the case in Data Block 4. The safety logic does not yet include a land-and-hold-short algorithm, so some interference resulted, as discussed below.

9.5.5.1 Takeoff-hold Light Performance

There were 49 departures from Runway 9 and eight from Runway 15R. The takeoff-hold light on Runway 9 performed well. There was one 4-scan missed detection in Data Block 4, when an aircraft moved into position on Runway 9 while not in track (thus failing to arm the takeoff-hold light), while the previous departure was still on its takeoff roll. There was no anomaly in Data Block 7.

It is not possible to evaluate the performance of the takeoff-hold light at the Runway 15R full-length departure location due to inadequate radar surveillance, as mentioned in Section 9.5.2.1. The resulting missed detections are not counted since they are an unavoidable consequence of the suboptimal positioning of the radar.

In addition to the one missed detection in Data Block 4, there were six instances of THL interference on Runway 9, caused by land-and-hold-short operations on Runway 15R. As mentioned in Section 9.5.3.1, the safety logic currently has no provision for handling this condition, so these instances of interference will not be counted.

9.5.5.2 Runway-entrance Light Performance

There were 57 arrivals to Runway 15R and four to Runway 15L, in addition to the above-mentioned departure activity. There were only two instances of interference, both in Data Block 4 and both due to the same cause: a surface vehicle was moving along Runway 22R, with sufficient speed and acceleration to be declared a departure, thus illuminating the runway-entrance lights along the runway. Two aircraft crossed 22R against the lights, one inbound on Taxiway C, the other outbound on Taxiway S. There was

no danger in these crossings. Tuning of the safety logic's state transition criteria may lessen the likelihood that surface vehicles activate the runway-entrance lights, but such misactivations probably cannot be eliminated altogether, without some way to positively identify vehicles that are likely to be engaged in high-speed operations on the runways.

There was no observed REL anomaly in Data Block 7, but false high-speed tracks caused by radar returns from the choppy seas appeared over the water and on seven occasions were misidentified as approaching traffic and projected to Runways 4L, 4R, or 9, illuminating runway-entrance lights briefly. Most of these anomalies lasted only one or two scans, none gave rise to an observed false alarm. Modifications to the safety logic may be implemented that would delay slightly the projection of certain suspicious tracks that appear over the water with no ASR counterpart. This should eliminate most of these wave-action effects with minimal reduction in system effectiveness.

9.5.6 Synthesis

The observed light anomalies are shown in Figure 9-2 for the four operational configurations at Logan. The anomaly counts are taken from Tables 9.2 through 9.5 and are presented as anomalies per 100 operations. As in the tables, missed detections and false alarms of one-and-two scan duration are excluded. 100 operations correspond to very nearly one hour of Logan traffic (the 1993 average traffic load at Logan is 98 operations per hour). It is apparent that most of the takeoff-hold-light anomalies are missed detections and most of the runway-entrance-light anomalies are interference. This appears to be true regardless of the configuration.

Figure 9-3 shows the anomaly fraction for false alarms and missed detections in the four configurations. The anomaly fractions were arrived at by summing up the durations (measured in scans) of all observed anomalies of these two types in a configuration and dividing the sum by the total number of scans in the data blocks for that configuration. The quantity presented is thus the probability that, on any given scan, an anomaly is affecting someone on the airport surface. The black and gray bars correspond to the data presented in Tables 9.2 through 9.5 and Figure 9-2; the white extensions represent the aggregate durations of the short anomalies excluded from the tables and Figure 9-2. Multiplication of the ordinate by 2000 gives the aggregate duration (in scans) of observed light anomalies in an hour (one hour contains very nearly 2000 scans of the ASDE-X radar).

The four graphs of Figure 9-2 can be combined with weights of 0.3, 0.3, 0.3, and (for the 15/9 configuration) 0.1 to give a prediction of the long-term average performance of the RSLS. These weights correspond to the approximate historical frequency of the configurations. The same can be done for Figure 9-3. The results are shown in Figures 9-4 and 9-5. It is clear that the RSLS performs quite well considering its status as a technology-demonstration system: Figure 9-4 indicates that, averaged over the long term, there is better than a 97% chance that a cockpit crew will not encounter a missed detection of more than two-scan duration during the time they are taxiing in after landing or taxiing out for departure, better than a 99% chance that they will not encounter a false alarm of more than two-scan duration, and about a 98% chance that they will not experience even a mild case of interference. Many of the missed detections and false alarms are of short duration. This means that, when measured in terms of time (as in Figure 9-5) rather than events (as in Figure 9-4), the performance of the RSLS appears even better. For instance, Figure 9-5 shows that only about 1% of the time will some cockpit crew be confronted with a status light in the wrong state. A specific crew will be less affected.

It should be understood that none of the anomalies seen in this assessment represents a direct safety hazard. This is because one of the fundamental design constraints of the safety logic was that it should not

cause a dangerous situation. The most immediate effect of the anomalies is a brief and localized suspension of the protection that the RSLS normally provides.

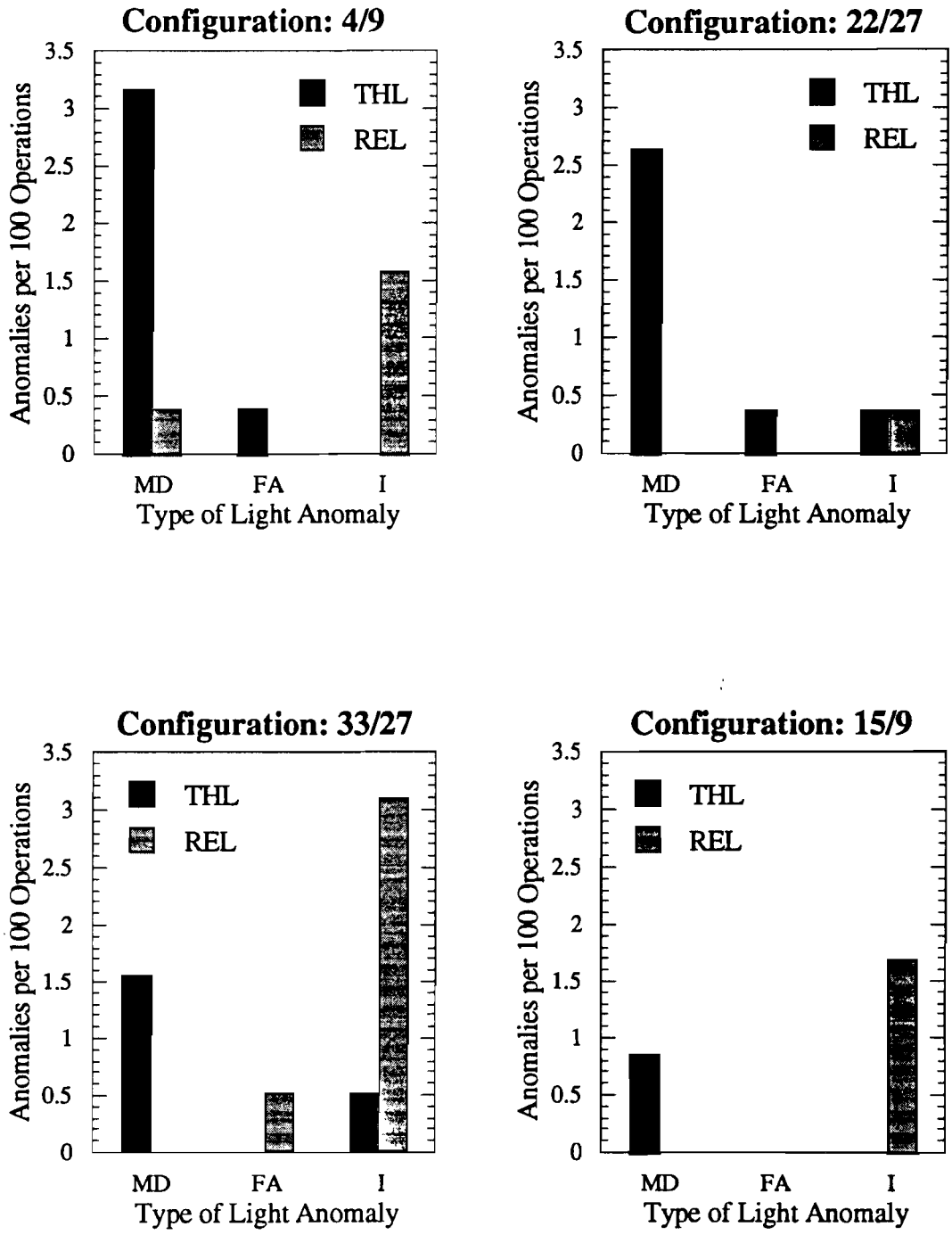


Figure 9-2. Observed light anomalies.

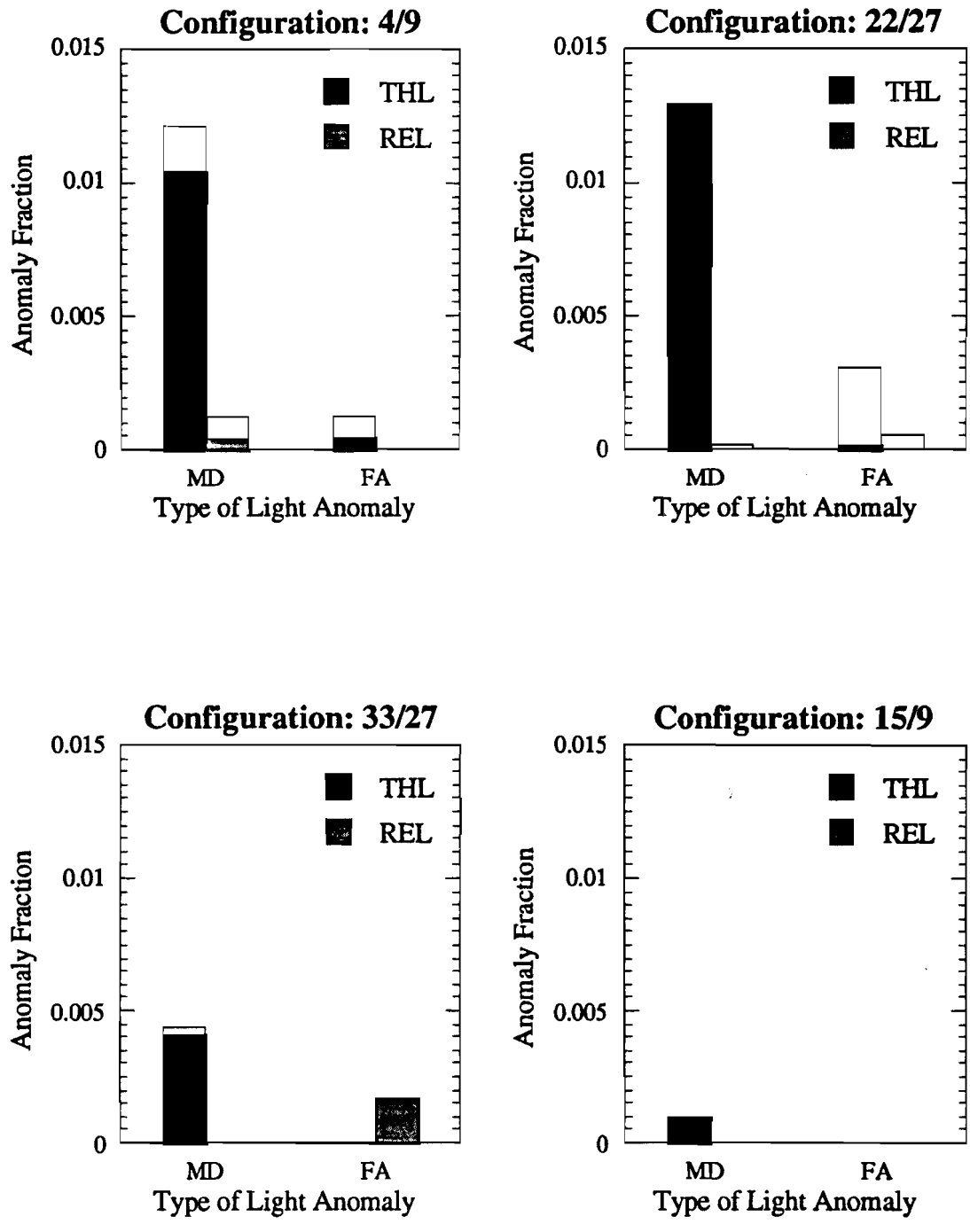


Figure 9-3. Anomaly fraction for false alarms and missed detections in the four configurations.

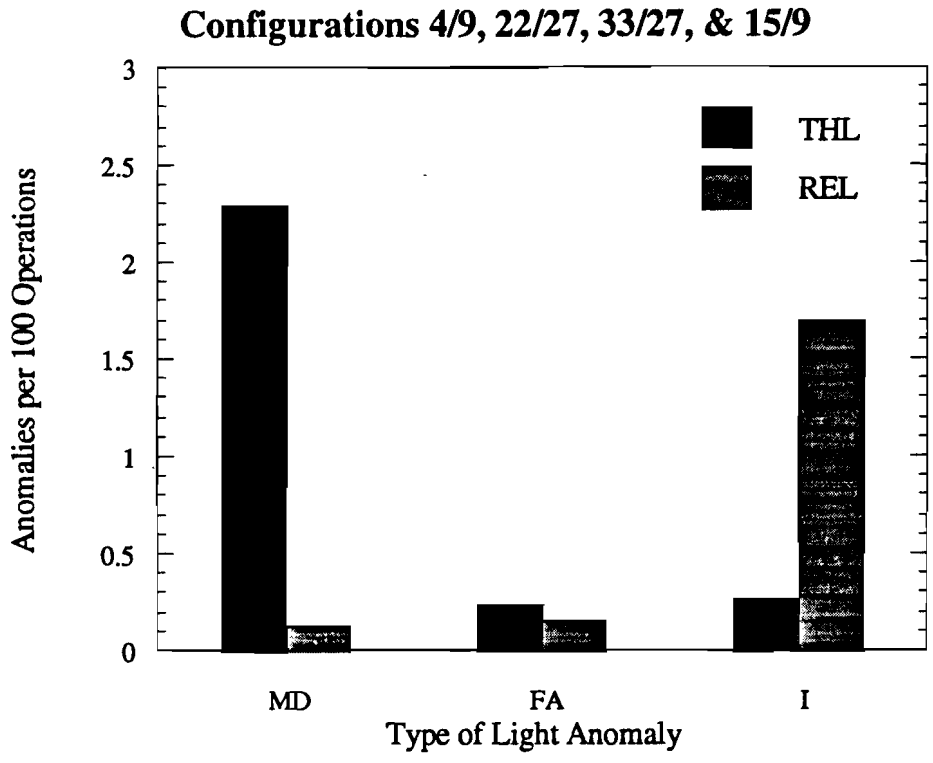


Figure 9.4. Observed light anomalies, composite.

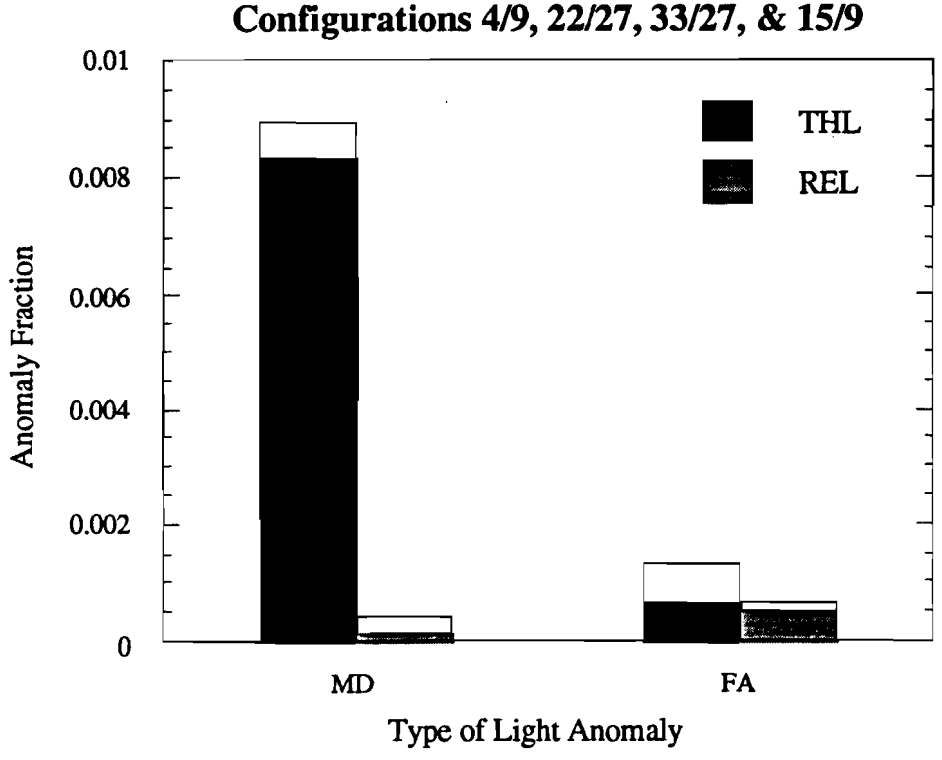


Figure 9-5. Anomaly fraction, composite.

Improvements

The RSLS performs well in its present state. But it can and should be made much better. Improvement efforts should be focused mainly on THL missed detections and REL interference. The fixes should, as far as possible, be generic. That is, custom-tailoring to a specific event should be avoided, in favor of general applicability. Similarly, location-specific fixes should be avoided, unless this is clearly appropriate. Adherence to these guidelines will ensure that the lessons learned in the present assessment have maximum leverage.

Most THL missed detections are due to one of the following three causes:

- Track drop while holding in position on the runway. This disarms the takeoff-hold light and prevents it from turning on. The resulting missed detections are sometimes of long duration (10 events averaged 12 scans each). The major trouble spots are the approach ends of Runways 9 and 22R; there are also track drops on Runway 4L at Taxiway S. These track drops appear to be mainly due to misassociations with multipath returns or deficiencies in the track-drop recovery algorithm.
- Track drop while taxiing across a runway. This results in failure to activate the takeoff-hold light, so that it does not turn on as it should if armed. The missed detections tend to be of moderate duration (four events averaged 7 scans each). The major trouble spots seem to be the intersections of Runway 4L/22R with Taxiways S and E and the intersection of Runway 33L/15R with Taxiway N. These are locations that are rich in multipath returns.
- Track drops while on final approach to a runway that intersects the currently active departure runway. This sometimes deactivates or prevents activation of an armed takeoff-hold light on the intersecting runway, causing it to turn off or, at times, come on late. The resulting missed detections are usually brief and probably of little direct operational consequence (11 events averaged 2 scans each). The major trouble spots are Runways 4R and 33L. Both are runways with approach-light piers, which present a difficult high-clutter surveillance environment.

THL false alarms are generally caused by multipath on the runway; they are usually of short duration (of nine such events, one lasted 10 scans, the other eight averaged 1.5 scans each).

Both THL missed detections and false alarms mainly reflect shortcomings in surveillance processing and tracking, and, to a lesser extent, sensor fusion. High-leverage improvements would center on clutter rejection, scan-to-scan association, and multipath rejection, as well as the track-drop recovery features. But the fixes should involve the safety logic as well, since a more robust safety logic will be less sensitive to surveillance and tracking imperfections.

Most REL interference is due to one of the following three causes:

- An aircraft rolling out after landing turns on the runway-entrance lights too far ahead and causes interference with crossing traffic.
- An accelerating, high-speed surface vehicle on the runway is mistaken for an aircraft on takeoff roll, thus activating the runway-entrance lights and causing interference with crossing traffic.
- Arrival traffic is projected to the wrong runway, causing erroneous REL activation. This occurs mostly on Runway 27, in the 33/27 configuration. It is a result of the approaches sometimes flown by aircraft landing on Runway 33R, which makes it difficult to determine the intended landing runway from surveillance alone.

The observed cases of interference were all due to the safety logic. The majority can be corrected either with parameter tuning or with algorithmic modifications.

9.6 SUMMARY AND CONCLUSIONS

The purpose of this performance assessment has been to provide preliminary quantitative data on the end-to-end performance of the RSLS, as it would be experienced by the system's users.

The assessment is based on approximately 10 hours of live Logan traffic data, recorded during March and April of 1993, comprising more than 800 operations. The data provide a brief but well-balanced glimpse of operations at Logan. Most hours of the operational day are represented, as are all major operational configurations and a variety of weather conditions.

The RSLS is a technology demonstration system. As such, it performs very well: a cockpit crew could expect to encounter a runway status light in an incorrect state only once in 18 operations. Half of these events would be brief, lasting less than 4 seconds. Interference would be experienced only once in 50 operations. Much of the interference would be mild. Overall, it is expected that the observed anomalies would have a negligible effect on airport capacity.

Although the RSLS performs well in its present state of development, it can be made better. An order-of-magnitude improvement in the system's end-to-end performance is a realistic goal that can be achieved by subjecting all functional elements to further algorithmic development and tuning. Such an effort would reduce the average number of anomalies experienced by pilots to approximately 10 per day. We believe that this level of performance is necessary to ensure success in an operational evaluation. High-payoff improvements and enhancements to the RSLS are outlined below.

None of the anomalies identified in this assessment, whether classified as missed detection, false alarm, or interference, would have presented a direct safety hazard. This is a consequence of one of the fundamental design constraints of the safety logic, which is that it should not cause a dangerous situation. The direct effect of a missed detection, for example, is a brief and localized suspension of the protection that the RSLS normally provides.

Improvements

Efforts to improve the performance of the RSLS should be focused mainly on missed detections that affect the takeoff-hold lights and on interference caused by the runway-entrance lights. Missed detections are mostly due to surveillance processing and tracking difficulties, interference mostly to the safety-logic. Much of the interference can be eliminated with simple tuning, some of the interference calls for algorithmic enhancements. Additional instances of interference occur in situations that are currently not covered by the safety logic: time constraints during the development process necessitated strict prioritization of the development work, and some critical algorithms have yet to be implemented. The high-payoff tasks that remain include:

- Improvements in surveillance processing and scan-to-scan association, especially in regions that are rich in multipath. This will reduce THL missed detections.
- Improvements in multipath rejection. This will reduce the number of false alarms.

- More robust track initiation and tracking on approach to Runways 4R and 33L. Both are runways with approach-light piers, which present a difficult high-clutter surveillance environment. More robust sensor fusion and safety logic projection algorithms will also help.
- Addition of a landing-rollout state in the safety logic.
- Addition of a land-and-hold-short algorithm in the safety logic.
- Tuning of the safety logic by adjusting parameters and regional boundaries to match the airport's operational patterns.

In addition to these improvements to the RSLS, there is a need for comprehensive, validated software for performing automatic performance assessment of the system. A general, flexible, and efficient assessment capability is required to evaluate and quantify the effects of both algorithmic changes and tuning. Such an assessment capability is also needed to perform the final assessment that must be carried out prior to installing status lights on the airport surface for operational evaluation.

APPENDIX A
DETAILED OUTLINE OF THE SAFETY-LOGIC ALGORITHMS

Note: The symbol "&&" means logical "and." The symbol "||" means logical "or."

A.1 TARGET-STATE MACHINE

A.1.1 Definition of target states

Name	Meaning (not definition)
Target states:	
STP	stopped
TAX	taxi
ARR	arrival
LDG	landing
DEP	departure
DBT	departure abort
LBT	landing abort: go-around after entering airport region, touch-&-go after entering airport region, missed approach after entering airport region, or not landing safely (e.g., going off the end of the runway)
UNK	unknown intent
Pseudostates:	
NONE	none, i.e., no track. This is used as a source and sink of tracks.
ANY	Any state.

A.1.2 Transition parameters

v_x- minimum speed to remain in state x

v_x+ maximum speed to remain in state x

Definition: A **propagation cell** is a polygon around a portion of a runway or taxiway used for projection and notification.

Note: Parameters are defined for the target unless they are overridden by a propagation cell or a runway. For example: 1) v_tax+ for arming regions at full-length-takeoff locations may be smaller than at intersection-takeoff locations; 2) v_ldg- may be larger on high-speed taxiways than at regular taxiways; 3) closed areas may have their own sets of parameters.

The following inequalities must be satisfied:

$0 < v\_tax- < v\_stp+ < v\_tax+ < v\_ldg+\_initial$

$v\_tax- < v\_ldg-$

$v\_tax- < v\_dbt- < v\_dep- < v\_tax+$

v_tax- < v_lbt- < v_tax+

v_unk_arr > v_arr_unk

a_dep > 0

a_dbt < 0

a_ldg < 0

The following is the algorithm for automatically adjusting parameters to maintain inequalities when one parameter is changed:

Case 1: Let p be the parameter that's decreased. Let p_large be the largest parameter that is > p but should be < p. Set p_large = (1. - delta_param)p. Repeat until inequality is satisfied.

Case 2: Let p be the parameter that's increased. Let p_small be the smallest parameter that is < p but should be > p. Set p_small = (1. + delta_param)p. Repeat until inequality is satisfied.

For example: suppose the required inequality is: a < b < c < d < e < f.

Case 1: Suppose e is decreased so that e < c < d. Set d = .95e, and set c = .95d.

Case 2: Suppose b is increased so that c < d < b. Set c = 1.05b, and set d = 1.05c.

A.1.3 Target-direction determination

Note: OPP is defined only for active runways, high-speed taxiways associated with those active runways, and other taxiways whose direction is defined for some other reason. Thus, OPP gets redefined with each configuration change. For runways, OPP is defined with respect to a given runway.

A.1.3.1 Target-direction states

Name	Meaning (not definition)
NRM	normal direction on a given runway or taxiway
OPP	opposite direction on a given runway or taxiway
CRS	crossing on a given runway or taxiway

A.1.3.2 Target-direction-state determination

Note: The angle theta is measured from the normal direction of the surface segment to the current heading estimate of the target (i.e., 0 ≤ theta ≤ 180 degrees).

Note: New targets are initially in the CRS state. Also, the default values assume the following symmetry:

CRS	NRM	theta < theta_nrm - theta_hyst_direc
CRS	OPP	theta > 180 - theta_nrm + theta_hyst_direc
NRM	CRS	theta > theta_nrm + theta_hyst_direc

OPP CRS theta < 180 - theta_nrm - theta_hyst_direc

The angles have a default value that is the same for all runways and taxiways but can be overridden for a specified propagation cell.

A.1.4 Definition of "inbound/outbound/midbound"

1. The **airport region** is a convex polygon around the airport that includes the entire movement area. This polygon should not extend too far beyond the movement area.
2. The angle **theta** is the difference between the current heading estimate of the target and the heading of a line drawn from the target through the center of the circle circumscribing the airport region (i.e., $0 \leq$ theta ≤ 180 degrees).
3. If the target is outside the circumscribing circle, the angle **phi** is determined by the following two rays: 1) a line drawn from the target through the center of the circle circumscribing the airport region, and 2) a line drawn from the target tangent to the circle circumscribing the airport region. If the target is inside the circumscribing circle but outside the airport region, then **phi** is set equal to 90 degrees.
4. A target is in the **airport approach zone (AAZ)** if it satisfies the following conditions: outside airport region

The parameters are:

d_inout (some miles)

hyst_inout (the hysteresis fraction of phi; a constant)

The possible target states are nobound, inbound, midbound, and outbound.

If the target satisfies the following conditions, then state = nobound: !(valid velocity) || outside AAZ.

Else, the following state-machine transitions are defined, always using the target's present heading (direction of its velocity vector):

From	To	When
nobound	midbound	valid velocity && inside AAZ
midbound	inbound	theta < phi
midbound	outbound	theta > 180 - phi
inbound	midbound	theta > phi + (hyst_inout)phi
outbound	midbound	theta < 180 - phi - (hyst_inout)phi

A.1.5 Other definitions

1.5.1 "approach area" for a runway: a polygon (e.g., trapezoid) out from the runway threshold. (**Note:** These polygons are functions of location.)

1.5.2 "a target is on approach to a particular runway:" [(centroid of target is in the approach area of the runway && target has a feasible path projected across the runway's threshold) || (centroid of target is in a surface propagation cell between the physical end of the runway and a displaced threshold && target is moving in the NRM direction relative to the runway)] && [ARR || LDG || LBT || (UNK && valid velocity && v>v_tax+ && inbound)]

1.5.3 "target has a feasible path projected across the runway's threshold:" there is a path from the target to the runway's threshold whose required transverse acceleration is less than the target's maximum achievable transverse acceleration (i.e., the target can make the turn onto the runway).

1.5.4 "a target is unambiguous:" target is on approach to only one runway

1.5.5 "a target is ambiguous:" target is on approach to more than one runway

1.5.6 "primary runway for an ambiguous target:" the runway with the lowest required transverse acceleration (i.e., the "straightest" of the approaches)

1.5.7 "a target is along a runway:" centroid of target is in a surface propagation cell && not on approach && target is moving in the NRM direction relative to the runway

1.5.8 "a target satisfies the target-runway-alignment conditions for a runway:" target is along the runway || {target is on approach to the runway && [target is unambiguous || (target is ambiguous && ambig_proj_switch = ON for this runway)]}

(**Note:** If ambig_proj_switch = ON, then an ambiguous target can trigger lights and alerts.)

1.5.9 "a target satisfies the approach-bar conditions for a runway" (i.e., is shown on the approach bar): target is on approach to the runway && [target is unambiguous || (target is ambiguous && this runway is the primary runway)]}

A.1.6 Target-state transitions

Note: Where velocity and acceleration are needed, assume they are valid.

From	To	When
NONE	UNK	new track
ANY	NONE	dropped track
UNK	ARR	inbound && v> v_unk_arr
ANY	STP	inside airport region && v<v_tax-
UNK	TAX	inside airport region && v≥v_tax- && v≤v_tax+ (this preferentially puts unknown targets with known velocity in the taxi state)
STP	TAX	v>v_stp+
LDG	TAX	v<v_ldg-
LBT	TAX	inside airport region && v<v_lbt-
ARR	UNK	v<v_arr- || midbound || outbound
LBT	UNK	outside airport region && not outbound
ARR	LDG	target is inside airport region
TAX	DEP	v>v_tax+ && a>a_dep>0

DEP	DBT	inside airport region && {[max(v), over all v while target in DEP state] - v} > delta_v_dbt
DBT	TAX	v<v_dbt-
LDG	LBT	exits airport region || v>v_ldg+(x), where x is location inside airport region (**Note:** The function v_ldg+(x) is defined as follows: there is a default value for v_ldg+(x), depending on location. Location-dependence is implemented by the state machine. The highest-priority object sets the value of v_ldg+. For a target on a runway, let x be its distance from the end of the runway. Let D be the length of the runway. Then for $0 \leq x \leq$ d_ldg+_initial, v_ldg+(x) = sqrt [(v_ldg+_final)^2 + 2 * decel_ldg+ * x], and for d_ldg+_initial $\leq x \leq D$, v_ldg+(x) = v_ldg+_initial, where v_ldg+_final is the minimum velocity at the end of the runway, v_ldg+_initial is the maximum velocity across the threshold, decel_ldg+ is a nominal deceleration, and d_ldg+_initial is the estimated distance from the threshold at which touchdown takes place. As opposed to the other parameters, d_ldg+_initial is not an input but is computed as the following: d_ldg+_initial = [(v_ldg+_initial)^2 - (v_ldg+_final)^2] / (2 * decel_ldg+).)
UNK	LDG	inside airport region && (along a runway || on approach to a runway) && v>v_tax+ && a<a_ldg<0 (**Note:** This transition could misclassify a DBT as a LDG, because prior state is not known. Also, the first condition is added to be conservative.)
UNK	DEP	inside airport region && along a runway && [(v>v_tax+ && a>a_dep>0) || v>v_ldg+(x)] (**Note:** This transition could misclassify a LBT as a DEP, because prior state is not known. Also, the first condition is added to be conservative.)
DEP	UNK	outside the airport region && not outbound

A.2 RUNWAY-STATUS LIGHTS

Note: Logic in bold type is not currently implemented.

A.2.1 Runway-entrance lights

High-level logic rule:

> At entrances to active and inactive runways (not closed runways or closed portions of runways) **and to the stopways at the approach ends of runways 4L and 9**, the runway-entrance lights are RED if the runway is not safe to enter at this intersection and OFF otherwise.

Note: The REL activation region is an area on the runway at the intersection.

Note: Currently, runway-entrance lights at the intersections of two runways (called "runway-intersection lights") are, in general, suppressed; however, there may be exceptions specified for a given configuration. We may eventually have logic for runway-intersection lights that is different from the logic for lights at runway/taxiway intersections; for example, they may be deactivated only when both runways are active, the logic may depend on target speed, or they may show directionality of the target triggering them by turning red on only one side of the entrance to the runway.

Note: whole_rwy_switch(state,runway) = whole-runway switch parameter, and is a function of the target state and the runway. If whole_rwy_switch = ON, then the lights turn red all the way down the runway ahead of the target.

Note: In the prediction engine, if the target's extent is outside the REL activation region, t_min_enter is measured from the FRONT of the target's extent to the nearest end of the REL activation region. If the target's extent is inside the REL activation region, t_min_enter(act_reg) is set equal to zero.

Lower-level logic rules:

2.1.1 "the runway is not safe to enter at this intersection:" there is a target that satisfies the target-runway-alignment conditions for this runway (see 1.5.8) && the target's hot zone overlaps the REL activation region of the RELs at this intersection

2.1.2 rules for ENTERING the condition "the target's hot zone overlaps the REL activation region of the RELs at this intersection:"

(**Note:** The parameter t_min_enter is the minimum time for the target to enter the REL activation region, measured from the front of the target's extent.)

2.1.2.1 TAX || STP || (LDG && v≤v_ldg_rollout) || (UNK && valid velocity && v>v_tax+): t_min_enter < t_state_rel, where "state" = tax, stp, ldg, unk

(**Note:** This is called the t-second rule.)

(**Note:** In future, could use whole_rwy_switch for UNK when it's between t_thresh sec. from threshold up to the time its velocity is < v_ldg_rollout.)

(**Note:** Need to add hysteresis for v_ldg_rollout.)

2.1.2.2 DBT: (t_min_enter < t_dbt_rel) || whole_rwy_switch = ON

2.1.2.3 (LDG && v>v_ldg_rollout) || LBT: (t_min_enter < t_state_rel) || [whole_rwy_switch = ON && (target is across the runway threshold || **target is t_thrshld seconds from the runway threshold**)], where "state" = ldg, lbt

(**Note:** Need to add hysteresis for v_ldg_rollout.)

2.1.2.4 ARR: (t_min_enter < t_state_rel) || (**whole_rwy_switch = ON && target is t_thrshld seconds from the runway threshold**), where "state" = arr, unk

2.1.2.5 DEP: [(t_min_enter < t_dep_rel) || whole_rwy_switch = ON] && the target is not tagged as the special altitude case

2.1.2.6 !(UNK && valid velocity && v>v_tax+) || NONE: no hot zone

2.1.3 special altitude case:

2.1.3.1 rule to tag target as special altitude case: valid altitude && alt > alt_dep_rel

2.1.3.2 rule to clear target as special altitude case: target is no longer DEP (i.e., target changes state) (**Note:** This condition precludes need for hysteresis.)

2.1.4 rules for LEAVING the condition "the target's hot zone overlaps the REL activation region of the RELs at this intersection:" (**Note:** The parameter t_max_exit is the maximum time for the target to exit the REL activation region, measured from the back of the target's extent. The condition "t_max_exit < t_antic_sep_rel" is for anticipated separation.)

2.1.4.1 if the condition was originally entered because of the t-second rule: (t_max_exit < t_antic_sep_rel) || [target's extent is outside the REL activation region && (t_min_enter ≥ t_state_rel + delta_t_state_rel), where "state" = state when lights were triggered] || target is dropped || (target is DEP && is tagged as the special altitude case) (**Note:** The second condition is hysteresis.)

2.1.4.2 if the condition was originally entered because whole_rwy_switch = ON: (target changed to a state where the whole_rwy_switch condition does not apply) || (t_max_exit < t_antic_sep_rel) || target is dropped || (target is DEP && is tagged as the special altitude case)

A.2.2 Takeoff-hold lights

High-level logic rule:

For active or inactive runways (not closed runways), the takeoff-hold lights are RED if there is a target [call it target A] that is (in position for takeoff || starting its takeoff) at this location && the runway is NOT safe for takeoff at this location, and are OFF otherwise.

Definitions:

1. arming region: a polygon that defines the area where a target is declared in position for takeoff or starting its takeoff

2. THL activation region: a polygon that defines an area used for determining whether the runway is safe for takeoff (e.g., the area downstream from the lights to the end of the runway) (**Note:** Currently, the stopways at the approach end of runways 4L and 9 will not be included in the THL activation region.)

3. straight-line path: the projected path along the runway that the target is on

4. intersection window: a polygon at a runway/runway intersection.

Comment: Suppose there is another target (call it B) whose propagation cell path includes at least one propagation cell in the takeoff-hold-lights' THL activation region. This propagation cell notifies the takeoff-hold lights about B. Essentially, the runway is "not safe for takeoff" if A might collide with B if A started to take off. More specifically, the runway is "not safe for takeoff" if B is in the THL activation region and not leaving "soon," or B is a high-speed target on an intersecting runway and is predicted to enter the THL activation region at an intersection before A can take off and pass that intersection, or B is a taxiing target predicted to enter the THL activation region.

2.2.1.a "A is in position for takeoff at this location:" A is STP && centroid of A is in the arming region && [prev_ldg_switch = OFF || (prev_ldg_switch = ON && A was not previously LDG)] (**Note:** The previous-landing switch can be specified on a light-by-light basis for each configuration.)

2.2.1.b "A is starting its takeoff at this location:"

1. rules for entering this condition: (A is TAX || DEP) && centroid of A has just entered the arming region && the difference between its heading and the heading of the runway is less than or equal to the angle theta_pos_hold && v ≤ v_pos_hold && [prev_ldg_switch = OFF || (prev_ldg_switch = ON && A was not previously LDG)] (**Note:** The previous-landing switch can be specified on a light-by-light basis for each configuration.)

2. rule for leaving this condition: centroid of A leaves the arming region

(**Note:** For arming regions at intersection-takeoff locations, it might be desirable to have v_pos_hold smaller than at full-length-takeoff locations, so that aircraft taxiing down the runway do not trigger the lights unnecessarily. Choosing this parameter is an example of the tradeoff between effectiveness and nuisance.)

Note: In what follows below, "window" refers to the intersection window.

Note: Assume A would start to take off according to the DEP acceleration model.

Note: The statements that B is in, or not in, a region means that B's *centroid* is in, or not in, a region.

2.2.2 "runway is not safe for takeoff at this location:" B satisfies the following conditions (depending on its target state)

2.2.2.1 B is STP || [UNK && !(valid velocity && v>v_tax+)]: B is not in the arming region && B is in the THL activation region

2.2.2.2 B is TAX: B is not in the arming region && [(B is in the THL activation region **&& if A started to take off, A could reach B before B definitely will exit the THL activation region) || (B is not in the THL activation region && B definitely will enter the THL activation region)**]

2.2.2.3 B is [LDG || ARR || LBT || (UNK && valid velocity && v>v_tax+)]: B is not in the arming region && {[B satisfies the target-runway-alignment conditions for A's opposite runway && (B is on approach to A's opposite runway || B is in the THL activation region)] || (B is along A's runway && B is in the THL activation region) || [(B satisfies the target-runway-alignment conditions for an intersecting runway || **the stopways at the approach ends of runways 4L and 9)** && A and B could be in the intersection window simultaneously if A started to take off]}

2.2.2.4 B is DBT: B is not in the arming region && {(B is along A's opposite runway && B is in the THL activation region) || (B is along A's runway && B is in the THL activation region) || [(B is along an intersecting runway || **the stopways at the approach ends of runways 4L and 9)** && A and B could be in the intersection window simultaneously if A started to take off]}

2.2.2.5 B is DEP: (same as 2.2.2.4) && B is not tagged as the special DEP case

Note: If B is [DEP || LDG || DBT || LBT || (UNK && valid velocity && v>v_tax+)] on a TAXIWAY and also in the THL activation region, the lights will be OFF. This is okay for now, because B has enough velocity so that it will definitely leave the THL activation region before A could enter the window.

2.2.3 "A could enter the window before B definitely will exit the window:"

1. rule for entering this condition: B's straight-line deceleration path exits the THL activation region && t_max_exit_B ≥ t_min_enter_A - t_sfty_mrg_arrvl, where t_min_enter_A is measured along A's acceleration path from the front of its extent, and t_max_exit_B is measured along B's deceleration path from the back of its extent

2. rule for leaving this condition: **t_max_exit_B < t_min_enter_A - t_sfty_mrg_arrvl - delta_t_arrvl** (**Note:** This condition is hysteresis.)

2.2.4 "A and B could be in the intersection window simultaneously if A started to take off:" B could enter the window before A could exit the window && A could enter the window before B definitely will exit the window

2.2.5 "B could enter the window before A could exit the window:"

1. rule for entering this condition: t_min_enter_B ≤ t_min_exit_A + t_sfty_mrg_clear, where t_min_exit_A is measured along A's acceleration path from the back of its extent, and t_min_enter_B is measured along B's acceleration path from the front of its extent

2. rule for leaving this condition: **t_min_enter_B > t_min_exit_A + t_sfty_mrg_clear + delta_t_clear** (**Note:** This condition is hysteresis.)

2.2.6 **"if A started to take off, A could reach B before B definitely will exit the THL activation region:"** t_min_reach_A < t_max_exit_B + t_antic_sep_thl, where t_min_reach_A is measured along the DEP acceleration path from the front of A's extent to B's centroid, and t_max_exit_B is measured along B's deceleration path from the front of its extent to the edge of the takeoff-hold-light THL activation region

2.2.7 **"B definitely will enter the THL activation region:"** all B's deceleration paths enter the THL activation region

2.2.8 special DEP case:

2.2.8.1 the rule to tag B as the special DEP case: B is DEP && vfr_switch = ON && B is along A's runway && B's distance from the front of A's arming region > d_dep_thl feet && [(valid altitude && B's altitude > alt_dep_thl) || B's velocity > v_dep_thl]

(**Note:** Altitude is not always available, e.g., when the target is an X-band target and, thus, is not fused.)

2.2.8.2 the rule to clear B as the special DEP case: B is no longer DEP (i.e., B changes its state) (**Note:** This condition means that B stays tagged as the special DEP case when it leaves the runway, and thus is no longer along the runway, but is still in the THL activation region. Thus, the THL will not momentarily flicker on.)

2.2.9 Calculation of t_min_enter_A:

Let d_max be the distance A travels up to a maximum velocity v_max_dep, assuming acceleration accel_nom_dep.

Then d_max = (v_max_dep)**2 / (2 * accel_nom_dep).

Let d_arrive_wndw be the distance to the window from a distance d_front_arming from the front of the arming region.

If d_arrive_wndw ≤ d_max, then t_min_enter_A = sqrt[(2 * d_arrive_wndw) / accel_nom_dep].

Otherwise, t_min_enter_A = (v_max_dep / accel_nom_dep) + [(d_arrive_wndw - d_max) / v_max_dep].

2.2.10 Calculation of t_min_exit_A: same as t_min_enter_A, except replace d_arrive_wndw with d_clear_wndw, the distance to clear the window.

A.3 DOUBLE-TARGET ALERTS (BETWEEN TARGETS A AND B)

Note: Logic in bold type is not currently implemented.

Note: In the case of multiple alerts, there is the capability for a priority scheme for determining which audible message is sent first. However, all octagons will be shown.

A.3.1 Head-on (same runway)

3.1.1 [ARR || LDG || (UNK && valid velocity && v>v_tax+)]/STP:

Note: The parameters t_stp_alert and delta_t_stp_alert are functions of the runway and a parameter called d_thresh. The parameter d_thresh is B's distance from the threshold of A's runway. For a given runway, the functions t_stp_alert and delta_t_stp_alert are step functions.

Option 1 (default):

a) rules for initiating the alert:

A is [ARR || LDG || (UNK && valid velocity && v>v_tax+)] && A satisfies the target-runway-alignment conditions for a runway && B is STP && B's centroid is in the ARR-alert activation region for A's runway && B is downstream from A && [A's centroid is ≤ t_stp_alert seconds from the runway threshold || **(B is between the approach end of the runway and the displaced threshold && A's centroid is ≤ t_stp_alert seconds from the approach end of the runway)]**

b) rules for ending the alert:

A is ![ARR || LDG || (UNK && valid velocity && v>v_tax+)] || B is !STP || B's centroid exits the ARR-alert activation region || (A || B) is dropped || B is not downstream from A || A's centroid is > t_stp_alert + delta_t_stp_alert from the runway threshold (**Note:** The last condition is hysteresis.)

Option 2: same as Option 1, except replace "from the runway threshold" with "from B's centroid"

APPENDIX B

SAFETY-LOGIC PARAMETERS REFERENCED IN CHAPTER 6 AND APPENDIX A

Parameter	Description	Default Value
v_unk_arr	velocity threshold for UNK to ARR transition	70 kts
v_arr-	velocity threshold for ARR to UNK transition	38 kts
v_tax-	lower velocity threshold for TAX	5 kts
v_tax+	upper velocity threshold for TAX	50 kts (override is 37 kts for departure runways: see Appendix C)
v_stp+	upper velocity threshold for STP	6 kts
v_ldg-	lower velocity threshold for LDG	34 kts
a_dep	accel. threshold for TAX to DEP transition	.15 g
a_dbt (not currently used)	accel. threshold for DEP to DBT transition	-.1 g
a_ldg	accel. threshold for UNK to LDG transition	a_dbt = -.1 g
v_dep- (not currently used)	lower velocity threshold for DEP	35 kts
v_dbt-	lower velocity threshold for DBT	currently is v_ldg-
v_dbt+ (not currently used)	upper velocity threshold for DBT	NYI
delta_v_dbt	maximum velocity decrease for DEP->DBT	22 kts
v_lbt-	lower velocity threshold for LBT	currently is v_ldg-
v_ldg+_initial	maximum velocity at threshold for LDG (for LDG to LBT transition)	180 kts
v_ldg+_final	minimum velocity at end of rwy for LDG (for LDG to LBT transition)	35 kts
v_ldg+_approach	maximum velocity in approach area for LDG (for LDG to LBT transition)	NYI (could be 170 kts)
delta_param	parameter adjustment to maintain inequalities of state-transition parameters	.05
t_horizon_dep	look-ahead time for departure	60 sec
t_horizon_arr	look-ahead time for arrival	60 sec
t_horizon_ldg	look-ahead time for landing	60 sec
t_horizon_dbt	look-ahead time for departure abort	60 sec

t_horizon_lbt	look-ahead time for landing abort	60 sec
t_horizon_unk	look-ahead time for unknown	30 sec
t_horizon_tax	look-ahead time for taxi	15 sec
t_horizon_stp	look-ahead time for stopped	0 sec
accel_nom_tax	nominal acceleration for TAX model	.12 g
v_max_tax	maximum TAX speed	35 kts
decel_nom_tax	nominal deceleration for TAX model	-.25 g
accel_nom_dep	nominal acceleration for DEP model	.25 g
v_max_dep	maximum DEP speed	180 kts
decel_nom_dep	nominal deceleration for DEP model	-.3 g
decel_nom_dbt	nominal deceleration for DBT model	-.3 g
decel_nom_ldg	nominal deceleration for LDG model	-.3 g
decel_nom_unk	nominal deceleration for UNK model	-.3 g
theta_nrm	angle threshold for NRM	15 deg
theta_hyst_direc	hysteresis angle for directionality on runway	5 deg
d_inout	distance threshold of airport approach zone	6600 m
hyst_inout	hysteresis fraction of phi for inbound/midbound and midbound/outbound	.1
t_stp_rel	time projected ahead for STP to turn REL on	0 sec
t_tax_rel	time projected ahead for TAX to turn REL on	0 sec
t_arr_rel	time projected ahead for ARR to turn REL on	36 sec (25 at 4L & S)
t_ldg_rel	time projected ahead for LDG to turn REL on	36 sec
t_dep_rel	time projected ahead for DEP to turn REL on	36 sec
t_dbt_rel	time projected ahead for DBT to turn REL on	36 sec
t_lbt_rel	time projected ahead for LBT to turn REL on	36 sec
t_unk_rel	time projected ahead for UNK to turn REL on	36 sec
alt_dep_rel	maximum altitude for DEP to turn REL on	250 ft
delta_t_stp_rel	delta time added to time projected ahead for STP to turn REL off	0 sec
delta_t_tax_rel	delta time added to time projected ahead for TAX to turn REL off	0 sec
delta_t_arr_rel	delta time added to time projected ahead for ARR to turn REL off	6 sec
delta_t_ldg_rel	delta time added to time projected ahead for LDG to turn REL off	2 sec

delta_t_dep_rel	delta time added to time projected ahead for DEP to turn REL off	2 sec
delta_t_dbt_rel	delta time added to time projected ahead for DBT to turn REL off	2 sec
delta_t_lbt_rel	delta time added to time projected ahead for LBT to turn REL off	2 sec
delta_t_unk_rel	delta time added to time projected ahead for UNK to turn REL off	2 sec
t_antic_sep_rel	anticipated separation time to turn off RELs	4.5 sec
v_ldg_rollout	velocity at which LDG becomes a landing rollout	55 kts
t_thrshld	time from rwy threshold that LDG can use whole-runway switch	NYI
whole_rwy_switch	whole-runway-switch parameter: ON means certain target states activate all the REL along the runway ahead of the target, regardless of the predicted times	1 (ON)
ambig_proj_switch	ambiguous-projection switch to be used for targets on approach; if OFF, then an ambiguous target cannot trigger lights or alerts	0 (OFF) (override is ON for certain runways: see Appendix C)
delta_d+	incremental distance for determining far edge of intersection window	100 m
delta_d-	incremental distance for determining near edge of intersection window	100 m
theta_pos_hold	angle threshold to determine in position for takeoff	65 deg
v_pos_hold	velocity threshold to determine in position for takeoff	35 kts
d_dep_thl	distance threshold between targets for special DEP case	2500 ft
v_dep_thl	velocity threshold for downstream target for special DEP case	75 kts
alt_dep_thl	altitude threshold for downstream target for special DEP case	100 ft
t_min_exit_A	minimum time for A to exit path-overlap window	depends on window
t_sfty_mrg_clear	safety margin time for clearing intersection window	5 sec
delta_t_clear	extra safety margin time for clearing intersection window - for hysteresis	NYI
t_min_enter_A	minimum time for A to enter path-overlap window	depends on window
t_sfty_mrg_arrvl	safety margin time for arrival to intersection window	5 sec
delta_t_arrvl	extra safety margin time for arrival to intersection window - for hysteresis	NYI
vfr_switch	VFR switch: ON means the procedures for VFR are in effect	0 (OFF)

prev_ldg_switch	"previously-landed" switch: ON means that a target in an arming region cannot trigger THL if it was previously in the LDG state	1 (ON)
t_stp_alert	threshold activation time for ARR/STP alert	35 sec
delta_t_stp_alert	hysteresis time for ARR/STP alert	4 sec

APPENDIX C

CONFIGURATION-DEPENDENT RUNWAY SETTINGS FOR OVERRIDING THE SAFETY-LOGIC PARAMETERS

Configuration at Boston Logan Airport

	4-9[1]	27-22	33-27	33-27_b	15-9
RELs[2] that can be activated at runway/runway intersections (default is to have RELs always off at runway/runway intersections)	facing 15L/33R @ 4L/22R	facing 15R/33L @ 4L/22R; facing 15R/33L @ 4R/22L	none	none	facing 4R/22L @ 15R/33L
Runway whose REL activation region triggers RELs at Tango taxiway (which is at the intersection of two runways)	4L/22R	4L/22R	15R/33L	15R/33L	15R/33L
Runway whose REL activation region triggers RELs at Juliette taxiway (which is at the intersection of two runways)	4R/22L	4R/22L	15R/33L	15R/33L	15R/33L
Runways where v_tax+ is overridden, i.e., runways used for departures	4L, 4R, 9, 15R	22L, 22R	27, 33L	27, 33L	9, 15R
Runway where ambig_proj_switch is overridden, i.e., where lights and alerts are triggered for ambiguous approaches	4L (instead of 9)	27 (instead of 33R)	none	27 (instead of 33R) [except when ARTS scratchpad says "33R": not yet implemented]	9 (instead of 4L)

[1] There is another configuration called 4-9_dv, which is a variation of the 4-9 configuration for daytime and when VFR conditions are in effect. The only difference between the two configurations is that for 4-9_dv, there is no displaced threshold for runway 4R.

[2] REL = runway-entrance light

APPENDIX D. ANALYSIS OF SURVEILLANCE JITTER

The analysis in this appendix provides more details in support of the analysis of surveillance accuracy given in Section 8.1.4.

Effective Noise Level. Consider a white-noise model, in which surveillance errors are independent from scan to scan, and have a constant standard deviation. In the following discussion the term "noise" is used for the jitter in reported target position. Let

$$x_m = x_t + n_x$$
$$y_m = y_t + n_y$$

where

x_m = measured value of x
x_t= true value of x
n_x = measurement inaccuracy or noise

and the corresponding definitions apply to y. In this model the noises n_x and n_y are independent of each other and independent from scan to scan. Let σ_c = standard deviation of n_x. Then because

$$\text{residual} = x_3 - 2\,x_2 + x_1$$

it follows that the standard deviation of the residuals is

$$\sigma_r = \sqrt{6}\,\sigma_c$$

Therefore if σ_r is determined from measurements, σ_c can be determined from this formula. This applies to the white noise model. It can be applied to the radar data if we speak of an effective noise level. That is, we define the effective noise level of the surveillance system to be the level of a white noise that would produce the same σ_r.

Accelerations. The above analysis applies when acceleration is zero. To determine the amount of accelerations that can be neglected relative to the noise effect, calculate the effect of a pure acceleration on σ_r. Model the aircraft motion as quadratic:

$$x = x_0 + a\,(t-t_0)^2/2$$

where a = acceleration in units of distance per scan squared, t = time in scans, and t_0 is the time when the acceleration begins. A similar equation applies to y. When this model is substituted in the formula for the residual, the result gives the component of the residual contributed by acceleration:

$$\text{residual} = a \text{ (in units of distance per scan squared)}$$

Note that the residual is independent of x_0 and t_0.

Noise Plus Accelerations. In the more general case, when accelerations and jitter are present together, the residual is a random variable having non-zero mean. Its rms value is

$$\text{rms(residual)} = \sqrt{\text{mean}^2 + \text{variance}}$$

$$= \sqrt{a^2 + 6\,\sigma_c^2}$$

APPENDIX E. LIST OF ACRONYMS

AAZ - Airport Approach Zone

ACID - Aircraft Identification

ACP - Azimuth Change Pulse

ADIDS - ARTS Data Interface and Distribution System

ADS - Automatic Dependent Surveillance

AMASS - Airpot Movement Area Saftey System

API - Applications Programming Interface

ARP - Azimuth Reference Pulses

ARR - Arrival

ARTS - Automated Radar Terminal System

ASCII - American Standard Code for Information Interchange

ASDE - Airport Surface Detection Equipment

ASR - Airport Surface Radar

ATCBI - Air Traffic Control Beacon Interrogation

ASTA - Airport Surface Traffic Automation

ASV - Average Square Value

ATCRBS -

ATBC - ARTS Tracks

ATRAIN - ARTS IIIA/TRACS Interface

AUC - ARTS Unfused Tracks

BI (Section 5.9)

CA -

CDR - Continuous Data Recording

CFAR - Constant False Alarm Rate

COTS - Commercial Off The Shelf

CPU - Central Processing Unit

CRS - Crossing (on a given runway or taxiway)

CW - Continuous Wave

D-BRITE - Digital Bright Radar Indicator Tower Equipment

DABT - (TED)

DBM - Display Buffer Memory

DBT - Departure Abort

DCD - Data Carrier Detect

DEC - Digital Equipment Corporation

DEDS - Digital Electronic Display System

DEP - Departure

DOS - Disk Operating System

E/SE - East/South East

EDT - Eastern Daylight Time

FAA - Federal Aviation Administration

FAR - False Alarm Rate

FC - Fused Tracks

GA - General Aviation

GHz - Giga Hertz

HP - Hewlett Packard

ICAO - International Civil Aviation Organization

IF - Inter-Facility

IFC - Interfacility Communications

ILS - Instrument Landing System

IMC - Instrument Meteorological Conditions

IOP - Input/Output Processor

LARS - Lincoln ASDE Recording System

LBT - Landing Abort

LDG - Landing

LSB - Least Significant Bit

LTM - Long Term Memory

MDBM - Multple Display Buffer Memory

MHz - Mega Hertz

MIMD - Multiple Instruction Multiple Data

MIPS - Millions of Instructions Per Second

MSAW - Minimum Safe Altitude Warning

MSB - Most Significant Bit

NA - Not Applicable

NCP - NAS Change Proposal

NIL - Tracks that are to be dropped

N/NE - North/North East

NRM - Normal Direction (on a given runway or taxiway)

NYI - Not Yet Implemented

OPP - Opposite Direction (on a given runway or taxiway)

PC - Personal Computer

PRF - Pulse Repetition Frequency

PRI - Pulse Repetition Interval

RCS - Radar Cross-Section

REL - Runway Entrance Light

RIB - Radar Interface Board

RLT - Run-Length Table

RS-232 -

RSLS - Runway Status Lighting System

RTQC/Test - Real Time Quality Control/Test

RW - (Table 8.1)

SCIP - Serial Communications Interface Processor

SCSI - Small Computer System Interface

SGI - Silicon Graphics Incorporated

SIMD - Single Instruction Multiple Data

SRAP - Sensor Receiver and Processor

S/SW - South/South West

STC - Sensitivity Time Control

STM - Short Term Memory

STP - Stopped

TAX - Taxiing

TBC - (Section 5.6.2)

TC - Fused Tracks

TCA - Terminal Control Area

THL - Takeoff-Hold Light

TI - Texas Instruments

TRACS - Transportable Radar Analysis Computer System

UNK - Unknown Intent

USAF - United States Air Force

VFR - Visual Flight Rules

VLDS - Very Large Data Store

VMC - Visual Meteorological Conditions

VME -

W/NW - West/North West

XTBC - X-band Tracks

XUC - X-band Unfused Tracks