AD-A204 719

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

DISTRIBUTED SENSOR NETWORKS

FINAL REPORT
TO THE
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

30 SEPTEMBER 1986

Approved for public release; distribution unlimited.

LEXINGTON MASSACHUSETTS

1/2

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the Department of the Air Force under Contract F19628-85-C-0002.

The views and conclusions contained in this document are those of the contractor and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the United States Government.

The ESD Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Hugh L. Southall

Hugh L. Southall, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

ABSTRACT

This is the final Lincoln Laboratory technical report for the DARPA Distributed Sensor Networks (DSN) project. The report reviews the DSN concept and documents the accomplishments, technical results, and lessons learned during the project. Topics covered include: acoustic direction finding, distributed tracking algorithms, hardware and software elements of an experimental test bed, network communication and self-location, multisite data-integration, and experimental demonstrations of tracking low-flying aircraft using multiple distributed acoustic and TV sensors.

TABLE OF CONTENTS

Abstract iii
List of Illustrations vii
List of Tables xi

1. EXECUTIVE SUMMARY 1

2. INTRODUCTION TO DISTRIBUTED SENSOR NETWORKS 7

3. TEST BED 13
   3.1 Introduction 13
   3.2 Test-Bed Hardware 13
   3.3 Test-Bed Software Systems 25

4. ACOUSTIC ARRAY PROCESSING ALGORITHMS 31
   4.1 Introduction 31
   4.2 Theoretical Basis for Wideband Array Method 33
   4.3 Wideband Algorithm for DSN Planar Arrays 35
   4.4 Experimental Results 41

5. TV SENSOR ALGORITHMS 47
   5.1 Introduction 47
   5.2 Target Selection and Camera Control 48
   5.3 Image Processing 55
   5.4 Target Detection 55
   5.5 Experimental Performance 58

6. DISTRIBUTED TRACKING ALGORITHMS 61
   6.1 Introduction 61
   6.2 Bayesian Model-Based Distributed Tracking Algorithm 61
      6.2.1 Overview 61
      6.2.2 Basic Notation and the Target Dynamic Model 64
      6.2.3 Position Track Updating with TV Azimuth Measurements 67
      6.2.4 Position Track Updating with Acoustic Azimuth Measurements 74
      6.2.5 Combining Local and Foreign Position Tracks 80
      6.2.6 Position Track Broadcast Policies 85
      6.2.7 Acoustic Azimuth Tracking 93
      6.2.8 Acoustic Position Track Initiation 96
      6.2.9 Azimuth Track Broadcast Policies 101
      6.2.10 Tracking Multiple Aircraft 101
      6.2.11 Possible Improvements 104
   6.3 Delayed Acoustic Tracking 106

7. DISTRIBUTED SURVEILLANCE SOFTWARE 113

7.1 Introduction 113

7.2 Functional Elements 115

7.2.1 Tracking Algorithm 116

7.2.2 Other Functional Elements 119

7.3 Process Groups 121

7.4 Messages 126

7.5 Using the UIP to Run an Experiment 129

8. COMMUNICATION SUBSYSTEM 137

8.1 Introduction 137

8.2 Communication Hardware 138

8.3 Distributed Surveillance Communication Software 142

8.4 Communication Network Technology Research 145

9. REAL-TIME TEST-BED EXPERIMENTS 149

9.1 Introduction 149

9.2 Multisensor Distributed Tracking Experiments 149

9.3 System Integration and Checkout Tests 156

10. ALGORITHM AND SOFTWARE DEVELOPMENT TOOLS 157

10.1 Introduction 157

10.2 Acoustic Data Simulator 158

10.3 Network Simulator 161

10.4 The Test Bed as an Acoustic DSN Simulator 165

10.5 The Test Bed as an Acoustic and TV DSN Simulator 171

10.6 Symbolic Information Flow Simulation 175

10.7 Software Development Tools 177

11. DISTRIBUTED SURVEILLANCE SYSTEM DEVELOPMENT LESSONS 181

12. SELF-LOCATION ALGORITHM STUDIES 185

12.1 Introduction 185

12.2 Location Algorithm 187

12.3 Internodal Range Measurement 193

13. MULTISITE DATA-INTEGRATION STUDIES 197

13.1 Introduction 197

13.2 Multisite Integration Algorithms 197

13.3 User Display and Multisite Integration Examples 200

13.4 Discussion 209

14. KNOWLEDGE-BASED SYSTEM DIAGNOSIS 213

15. REMAINING RESEARCH AND DEVELOPMENT 219

ACKNOWLEDGEMENTS 221

REFERENCES 223

LIST OF ILLUSTRATIONS

Figure No.		Page
2.1	Deployment Options for Distributed Sensor Networks	11
3.1	Lincoln Laboratory DSN Test-Bed System	14
3.2	One of Three Experimental Vehicles Used for Field Deployment of DSN Test-Bed Elements	14
3.3	Microphone with Preamplifier Attached to Wooden Block for Ground-Level Field Deployment. 2-in Windscreen Covers Microphone Head	15
3.4	Field Deployment of a Nine-Microphone Array	16
3.5	Weather Protected TV Camera Installed on Remote Control Mount	17
3.6	Standard Nodal Computer (SNC) Used in the DSN Test Bed	18
3.7	Standard Acoustic Node for the DSN Test Bed	20
3.8	Acoustic Signal Processing Subsystem (SPS)	21
3.9	Standard TV Node for the DSN Test Bed	22
3.10	Gateway Node for the DSN Test Bed	22
3.11	Acoustic/Tracking Node and Gateway Node Installed Within Experimental Vehicle	24
3.12	Node Run Time System (NRTS) Application Programming Interface	25
3.13	NRTS Organization	26
3.14	Hierarchical Structure of the File Oriented Input-Output System	26
4.1	Delay and Sum Beamforming	31
4.2	Spectral Mapping Property of the Zero-Delay Wavenumber Spectrum	35
4.3	Schematic Representation of Wideband Array Algorithm	36
4.4	Comparison of Wideband and Frequency-Wavenumber Methods	40
4.5	Tri-Delta Microphone Array Configuration	42
4.6	Acoustic Power as a Function of Bearing for Two Targets	44
4.7	Azimuth Estimates as a Function of Time	46
5.1	TV Subsystem Elements and Tracker Interactions	47

Figure No.		Page
5.2	Complete Camera Pointing Algorithm	48
5.3	Examples of Camera Pointing Policies	50
5.4	Selection of Pointing Algorithms	51
5.5	Elevation Control Algorithm	52
5.6	Target Detection Probability of TV Subsystem	54
5.7	TV Subsystem Target-Detection Algorithm	56
5.8	Camera Control During Live Experiment	58
6.1	Organization of the Distributed Tracking Algorithm	62
6.2	Updating a Target Track Using a TV Measurement: Conceptual Steps	69
6.3	Real-Time Helicopter Tracking with a Single TV Subsystem	73
6.4	Updating a Track with an Acoustic Measurement: Conceptual Steps	75
6.5	Real-Time Helicopter Tracking with a Single Acoustic Array	79
6.6	Information Diagram Illustrating Serious Information Loss when Only Total Tracks Are Broadcast	83
6.7	Information Diagram Illustrating Minor Information Loss Situation	84
6.8	Symbolic Information Flow Diagram Illustrating Alerting	85
6.9	Illustration of Track Broadcast Sequence for a Single Aircraft	88
6.10	Helicopter Tracking Example for Minimal Communication Policy	90
6.11	Helicopter Tracking Example for Maximum Communication Policy	91
6.12	Illustration of Location Using Possible Position Curves	108
6.13	Illustration of Location Using the Reflection Method	109
7.1	Functional Organization of Tracking Software	114
7.2	Tracking Process Group	122
7.3	Broadcast Input Filter Process Group	124
8.1	Test-Bed Communication Subsystems	137
8.2	Microwave Ring Used to Interconnect Test-Bed Ethernets	140
8.3	Organization of Radio Channel into Slots with Discrete Packet Start Times Within Each Slot	147

Figure No.		Page
9.1	Experimental Test Bed for DSN Demonstrations	150
9.2	Demonstration Scenario for Distributed Mixed Sensor Tracking	150
9.3	Real-Time Helicopter Track Display Obtained by Tracking with Both Acoustic and TV Nodes	153
9.4	Local TV Display with Remote TV Image Overlay	155
10.1	Elements of the SPS Output Simulation Software	160
10.2	Elements of the Acoustic Sensor and Signal Processing Model	162
10.3	Organization of Application Software	163
10.4	Organization of Test-Bed Simulator Software	164
10.5	Procedures for Using Simulated Data and Logging Performance	166
10.6	Simulated Track of Two Helicopters Through Six Nodes	167
10.7	Duplicate Tracks Obtained During the Simulated Flight of a Single Helicopter	170
10.8	TV Simulation Module for Real-Time Experimentation	172
10.9	Simulated Combined TV/Acoustic Tracking Scenario	173
10.10	Position Track and Error Comparisons	174
12.1	Network That Cannot be Configured by Triangulation	186
12.2	Network Layout and Measurement Options	189
12.3	Sparsely Connected Network Used for Self-Location Tests	192
12.4	Illustration of Strong Late Multipath Arrival	194
12.5	Determination of Range Using Time-Stamped Messages	196
13.1	Example of a Situation Display	201
13.2	Situation Display Example at a Later Time and Zoomed In	205
13.3	Situation Display Showing a Communication Failure	207
13.4	Illustration of Communication and Integration Interrelation	211
14.1	Sample Output from Diagnosis System	216

LIST OF TABLES

Table No.		Page
2.1	Distribution Option for Surveillance System Functions	8
12.1	Algorithm Performance for Network in Figure 12.3	191
14.1	Equal-Resolution Operator	217

DISTRIBUTED SENSOR NETWORKS

1. EXECUTIVE SUMMARY

The overall DARPA Distributed Sensor Networks (DSN) program involved several research organizations and was aimed at developing distributed target surveillance and tracking methods for systems employing multiple spatially distributed sensors and processing resources. Such systems would be made up of sensors, data bases, and processors distributed throughout an area and interconnected by an appropriate digital data communication system. The working hypothesis of the program was that through netting and distributed processing, the information from many sensors could be combined to yield effective surveillance systems. The overall concept called for a mix of sensor types as well as geographically distributed sensors.

The Lincoln Laboratory program emphasized the development of an experimental test bed to demonstrate DSN concepts. Surveillance and tracking of subsonic low-flying aircraft using ground based acoustic and imaging sensors was selected as the focus of the system development and demonstrations. Small arrays of microphones providing directional information were employed as acoustic sensors, and visible wavelength TV cameras were used as imaging sensors in the test bed.

A primary objective of the Lincoln Laboratory DSN effort was to prove the feasibility of distributed surveillance systems by demonstrating real-time distributed tracking of low flying aircraft. This was accomplished. The real-time acoustic and TV tracking experiments described in Section 9 constitute the primary feasibility demonstration. They involved real-time helicopter tracking using an experimental system composed of six sensor/processor nodes deployed in the vicinity of Hanscom Field.

The major part of the effort was the development and test-bed implementation of distributed algorithms to demonstrate distributed tracking. Most of this report deals with these topics. Algorithms and test-bed systems are described and discussed in detail. Experimental results, involving simulated as well as real data, are included to illustrate concepts and validate algorithms. Other topics that are covered include network self-location and multi-site data integration.

The test-bed tracking system is implemented using autonomous cooperative processes operating at each of the nodes. Once started in operation, each tracking process interacts with those at other nodes through discrete data messages, with no external control. Each node performs tracking with whatever data are available to it: data derived from attached sensors and data obtained through messages from other nodes. This structure has resulted in a tracking system that easily adapts to changes in the number and type of sensor/processor nodes in the network. When a node fails the only impact on the overall system is the degradation in tracking performance due to the loss of the sensor at the node.

Real-time distributed tracking demonstrations required the development of suitable algorithms. Tracking algorithms were developed to: (1) perform sensor independent tracking; (2) track aircraft with small microphone arrays; and (3) perform tracking with a mix of sensor types, including acoustic and TV sensors.

The DSN tracking algorithms depend upon well understood estimation methods. They can easily be adapted to accommodate new sensor types by the addition of sensor specific Kalman Filters or Extended Kalman Filters for each new type. Also, they provide estimates of present target positions despite acoustic propagation delays and automatically utilize the data from any number of DSN nodes. These algorithms replace less general acoustic tracking algorithms that were developed during the early stages of the DSN project. The approach for the earlier algorithms was to minimize assumptions about target dynamics. This resulted in time-delayed position estimates, ineffective use of the data from the sites nearest to the target, and provided no obvious way to combine acoustic with non-acoustic tracks. The initial algorithms also provided no theoretical model for utilizing data. The new algorithms solved these problems by modeling target dynamics and using the models as the basis for tracking in terms of present target position, independent of acoustic propagation delays. For tracking purposes the targets are now modeled as constant velocity objects subject to random accelerations.

There are several tracking algorithm components. The azimuth tracking algorithm is a Kalman Filter that processes acoustic azimuth measurements into azimuth tracks. The filter assumes constant angular motion with random angular acceleration and forms azimuth tracks without reference to the geographical position of the target. The track initiation algorithm forms position tracks using pairs of azimuth tracks from two different nodes. Extended Kalman Filters (EKF) are used to update position tracks with acoustic azimuth measurements or line-of-sight azimuth measurements obtained from TV cameras. All algorithms compensate for acoustic delays when needed so that tracking is in terms of present target position estimates.

The Extended Kalman Filters in a node process only measurements derived from the sensors attached directly to the node. Sensor information from other nodes is indirectly utilized by forming optimal Bayesian combinations of local position tracks with position tracks received in messages from the other nodes. This combining is independent of the sensor types since it deals only with target state and error covariance estimates. The combining algorithm is the mechanism we have used to decompose and distribute the tracking function so each node is concerned only with its own attached sensors.

Other tasks handled by the tracking algorithms include data association, track association, and track maintenance. We elected to implement these functions as simply as possible. In addition we elected to allow each measurement to be associated with only one track. One result is that alternative interpretations of the measurement data are not considered. These are areas in which significant improvements could be made to the algorithms.

Data association is the assignment of new sensor measurements to tracks. This is done by comparing new measurements with tracks and associating a measurement with the first track that yields a good match. The match test is statistical and takes measurement errors and the track errors into account.

Track association identifies a track estimate from one node with the correctly corresponding track estimate at another node based on track identifiers that are assigned when tracks are initiated. During normal system operation tracks are initiated by one or two nodes. For the two node case, the track identification rules are such that both assign the same identifier. Subsequently, other network nodes are alerted to approaching targets and given the correct identifier.

Track maintenance involves deciding when to add or delete a track from the track data base in a node. Tracks are deleted when their estimated errors exceed a threshold. A track can persist (coast) during periods when no new measurements are associated with the track. Without new measurements the estimated errors increase, eventually becoming large enough to cause the track to be deleted. A track generated by the track initiation algorithm is added to the data base when it satisfies a number of conditions based on its estimated error and the geometry of the situation.

The last important issue for the distributed tracking algorit!ıms is what information to exchange between nodes and when to broadcast it. Unprocessed sensor measurements are never broadcast. Azimuth tracks are broadcast when they have not been associated with a position track. This makes them available for position track initiation. Position tracks are broadcast when their estimated errors are small enough and several other conditions are met. Some conditions involve where the estimated track is located relative to the node and to neighboring nodes that are expected to receive the broadcast. The rate at which updated position tracks are broadcast may also be reduced to limit the communication load of the network. All aspects of tracking algorithms are treated in detail in Section 6.

Demonstration of real-time tracking required the development of acoustic signal processing algorithms to detect targets and determine directions. A wideband array processing algorithm was developed for this purpose and was implemented and used for real time tracking demonstrations. This algorithm is based upon a new approach to direction finding with arrays. It depends upon the following facts. First, the spatial correlation function of a plane wave is like a long mountain range with the ridge line perpendicular to the direction from which the signal is arriving. This fact does not depend upon the temporal bandwidth of the plane wave. But the two dimensional spatial Fourier transform of a mountain range is another long mountain range, but this time passing through the origin of the two dimensional spatial frequency space and rotated by ninety degrees to be oriented in the direction of the arriving signal. The mountain ranges for independent plane waves arriving from different directions are simply added together. The wideband algorithm looks for these mountain ranges radiating from the origin. Their directions are target directions and their heights are proportional to signal strength.

The wideband algorithm replaced a more conventional one that was developed and used for initial DSN experimentation. The new algorithm provided more reliable detections and direction finding while requiring fewer computations than the original. Acoustic detection and direction finding algorithms are discussed in Section 4.

Whereas acoustic arrays are appropriate for surveillance and track initiation as well as for tracking, the TV sensors have limited fields of view and are more appropriate for improving tracks than for initiating new ones. The TV cameras were developed to function as autonomous cued measurement resources. They operate as follows. The TV subsystem receives position track estimates and calculates which target it is most likely to be able to detect. It then slews to where that target is predicted to be and captures two frames of data which are used to attempt to detect the target on the basis of motion in the image frame. If there is a detection, then, using the known orientation of the camera and the position of the detection in the image frames, the target azimuth is estimated and sent to the distributed tracking system which integrates it with all other measurements. This is a closed loop system in which the TV uses target position tracks to select and find targets and produce measurements that subsequently improve tracks. The algorithms used by TV subsystems to select targets, control the TV subsystem, detect targets and provide azimuth measurements to the tracker are described in Section 5.

A major part of the Lincoln Laboratory DSN effort was the development of the test-bed system. It contains eleven nodes interconnected by Ethernets and radio communication links. Every node contains from one to three single board microcomputers that perform tracking and communication functions. Six of the nodes also contain acoustic subsystems consisting of a small microphone array and the electronics required to collect and process acoustic data in real time. Some nodes are installed in vehicles for field deployment. Two of the nodes are dedicated to TV functions. The test bed also includes microwave communication equipment that allows the test bed to be split into three physically separated parts, an essential capability for experiments involving mobile nodes separated by more than about a kilometer. Two UNIX workstations and a VAX computer constitute the remainer of the system. They are employed for software development, experiment control, and other analysis functions.

The test bed is a large hardware and software system that required substantial effort to develop and use. Section 3 describes the hardware and the system software. Section 7 describes the application level control and tracking software. Section 8 provides more details about the communication system. Sections 10 and 11 address the software development process and lessons that were learned about the development of such complex distributed systems. We have attempted in these last two sections to distill and document information that we believe may be particularly useful to future implementors of Distributed Surveillance Systems.

Sections 12, 13, and 14 cover problems other than tracking and test-bed development and use.

Section 12 treats the problem of how to estimate the network node locations using range measurements between pairs of nodes. A distributed self-location algorithm is described along with sample results obtained using simulated range measurements. Autonomous portions of the algorithm operate at each node. Each node estimates its position relative to other nearby nodes. The algorithm is iterative and continues to run as long as there are changes in the estimates. The range measurements needed by the algorithm could be obtained using radios to measure the transit time of electromagnetic signals between the nodes. Radio features and protocols to measure ranges, even with unsynchronized nodal clocks, are described.

Section 13 treats multi-site data collection: how to collect tracks from many nodes in a network and how to resolve differences between tracks provided by different nodes for the same target. The track combining algorithm developed for tracking can also be used for combining similar tracks during multi-site data collection; but this solves only part of the multi-site data collection problem. Section 13 discusses other aspects of the problem, especially the close coupling between communication and multi-site data collection issues.

Section 14 summarizes the results of a preliminary investigation of the application of Artificial Intelligence approaches to some acoustic data interpretation problems in an acoustic DSN.

The last Section discusses remaining DSN research and development issues.

2. INTRODUCTION TO DISTRIBUTED SENSOR NETWORKS

Distributed Sensor Networks (DSN) are surveillance and tracking systems that use many geographically distributed sensors. A pure DSN system would be composed of many autonomous sensor/processor nodes interacting to provide surveillance and tracking functions for users. All essential system functions, not just sensing, would be distributed and implemented in the form of interacting autonomous processes. The system would automatically adapt to changes in the number, location, and sensor capabilities of the nodes. It would contain no single points of failure. System performance would be determined only by the capabilities inherent in the set of sensors available to the system. Thus the performance of the system would change incrementally as individual sensor/processor nodes are added or removed.

This is the idealized DSN system concept. It is at one extreme of a range of options. Consider Table 2.1. It singles out sensing, data processing, and control as important surveillance system functions and suggests three possible levels of distribution: non-distributed, partially distributed, and fully distributed. The shaded boxes in the table highlight the emphasis of the Lincoln Laboratory research effort. We concentrated on full distribution for sensing and data processing, but our goals in the area of system control were more modest, emphasizing only partial distribution. All functions in an ideal DSN would be fully distributed.

The lowest level of distribution in Table 2.1 corresponds to traditional surveillance systems, typically radar systems, in which a single large sensor is employed for surveillance of an area. With only one sensor, the issue of the level of distribution of other functions does not arise.

The next level of sensor distribution is when the area covered by a single sensor, perhaps a traditional monostatic radar, is not adequate. It then becomes necessary to replicate and distribute many copies. Each has full surveillance capability, but only within a limited area. Such a distributed sensor system might employ any level of distribution to accomplish other critical functions. But the most natural thing is for both the data processing and control to be partially distributed.

First consider the data processing. Each radar can have associated processors that perform all radar functions. This eliminates the need for unrealistically large communication bandwidths from the sensors to some central processing facility; only tracks need be communicated. Radar

TABLE 2.1

Distribution Options for Surveillance System Functions

Degree of Distribution	System Function		
	Sensing	Data Processing	System Control
More	● Geographic Distribution for Area Coverage ● Multiple Sensors Required for Functionality	● Autonomous Surveillance at Each Sensor Site ● Extensive Multisite Data Combining	● Negotiated Hierarchy
	● Geographic Distribution for Area Coverage ● Multiple Sensors Not Required for Functionality	● Autonomous Surveillance at Each Sensor Site ● Minimal Multisite Data Combining	● Requestor/Server
Less	● One Sensor with Full Area Coverage	● Centralized	● Master/Slave Hierarchy

tracks or detections from multiple sites could be combined to provide improved system performance, but if each radar can be designed to provide adequate performance within its own area and there is limited coverage overlap there is little motivation to do this. Thus the resulting data processing system is distributed but there is little interaction between the sensors. It is because of the logical simplicity of the interactions that we assign only a moderate degree of distribution to such systems.

It is natural for the control for a multiple monostatic radar network to be partially distributed in the same sense as the data processing. Sensor sites can operate quite autonomously, interacting to a limited extent with other sites or with a central network coordinator. Sites may provide data to several users, but are not controlled by any external users.

Computer networks often contain server processes from which remote users can request service. This is similar to the radar network just described except that the server function becomes the primary function whereas local surveillance for specific users is the primary function of the radars. Servers distributed within a computer network constitute a form of distributed control of the network and its resources.

The server process idea can be adapted to distribute the control function in a DSN. Each node can respond to a community of users, agreeing to be controlled by any member rather than by a fixed master. This is slightly more distributed than site-by-site autonomous operation. It provides for multiple masters, but does not necessarily address how to coordinate them or how to select only one. This is the form of distributed control that we have adopted for the DSN test bed. Nodes act as servers. They start, stop, change parameters, provide data, etc., all under external command with little concern for the source of the commands. Thus several users and network controllers can operate simultaneously within the network. However, users are responsible for avoiding conflicts. This approach, with some modifications such as the use of priorities and software locks to avoid conflicts, could also be used to control more operational DSN systems without the need to fully solve some difficult problems associated with more completely distributed control. For example, a higher level of distribution of control might require that nodes interact with each other to adaptively design a control hierarchy. How to do this, and if this is the right approach are unanswered questions.

For sensing and data processing we concentrated upon the highest level of distribution. The highest level of sensor distribution is required when a single sensor cannot cover the entire area of interest and, in addition, the data from several geographically dispersed sensors are required to perform the surveillance function, even in a local area. Underwater surveillance systems employing networks of hydrophones are fully distributed in this sense. Networks of multistatic radars would also be fully distributed.

We worked on an atmospheric version of the underwater surveillance problem: the detection and tracking of aircraft using networks of geographically distributed microphone arrays. The range of the individual microphones is limited and measurements from two or more geographically separated sites are required to confidently locate and track an aircraft. We also elected to include imaging sensors, TV cameras, which have similar characteristics. This allowed us to investigate how to use complementary sensor types and to demonstrate that our results have general applicability.

The full range of distributed data processing options is theoretically available when the sensing function is fully distributed, although fully centralized data processing is not likely to be practical. For example consider a system of many small acoustic arrays. Except for small networks, completely centralized processing is impractical because of the large communication capacity required to transmit all the sensor data to a central site. A substantial amount of data reduction is required at each sensor site to keep communication requirements within reasonable bounds. For acoustic arrays this could be done by signal processing, detection and direction finding algorithms at each sensor site that convert raw data rates of several tens of kilobits per second per site to only a few hundred bits per second per site. The resulting data rates make it practical to transmit the reduced data to a central site for final processing for much larger networks. Such a processing system would be only slightly more distributed than a network of autonomous monostatic radars. Very large systems might employ several "centralized" processing sites which might interact with each other in the same way as autonomous monostatic radars might interact. But the most general solution is to fully distribute the data processing function.

Our view of fully distributed surveillance is that a processor should be co-located with each sensor, and the processor should perform surveillance of the area within which the co-located sensor can detect targets. Each site should transmit information to nearby sites and each site should expect to receive information from its neighbor sites. We have concentrated our efforts upon this level of data processing distribution.

DSN systems could be deployed in many different geometries and used for many different purposes. The geometries include thin barriers for early warning, thick barriers as part of barrier defense systems, small systems for local area or point defense systems, or large systems for wide area coverage. Figure 2.1 illustrates some deployment options. Each dot represents a sensor/processor site. The separation between sites would be sensor and application specific. A system using acoustic sensors might have separations of only a few kilometers whereas a system of radars might have much larger separations. A typical system would have sensor spacing

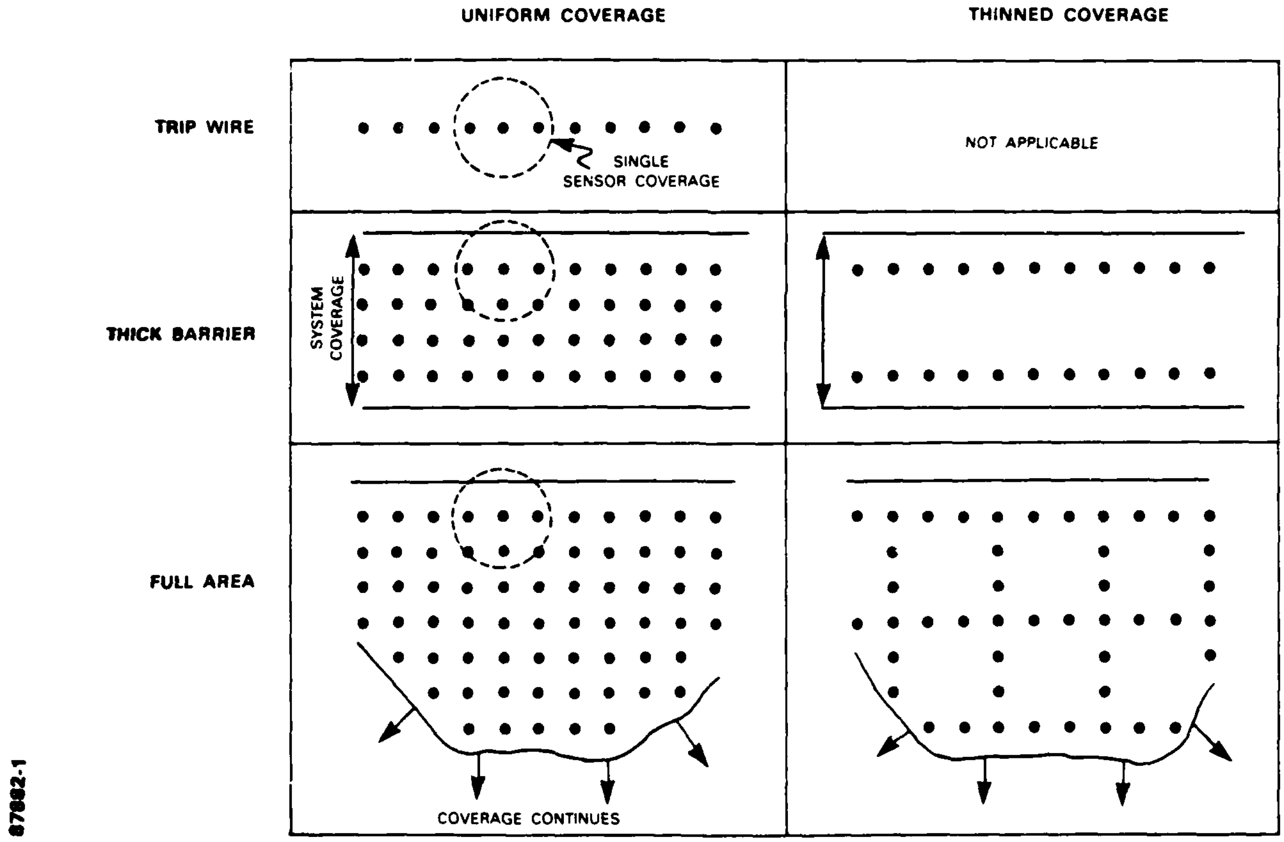


Figure 2.1. Deployment options for Distributed Sensor Networks.

adjusted to be slightly smaller than the reliable detection range of each sensor. This provides complete cost-effective coverage with no holes and with some redundancy that will provide robust system operation in the face of nodal failures.

Figure 2.1 also shows thinned deployments which might be used for some applications to reduce system cost. The idea is to recognize that continuous coverage may not be required and to exploit this fact to reduce the number of elements in the system. For example consider the case of a thick barrier that acts as an early warning and cueing system for other sensors or weapon systems. A first line of sensors could serve to provide a pre-alert for a localized area along a long front. The second line could then be used for more specific cueing. The important point to note is that DSN systems are not restricted to wide area systems with many nodes uniformly distributed. They can be designed for use in many different applications.

3. TEST BED

3.1. Introduction

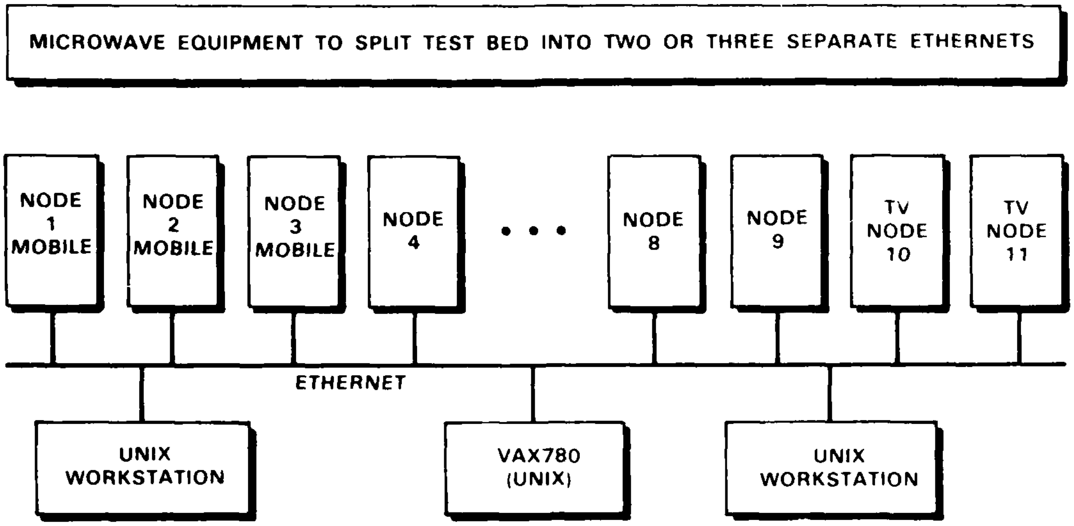
The Lincoln Laboratory DSN test bed consists of hardware and software systems intended to support the development and demonstration of distributed tracking algorithms. It can be used in several different modes and can be split into smaller test beds that can be independently used for different purposes. The modes include data acquisition mode, real-time mode with prerecorded or simulated data, and real-time mode with live targets. These modes have all been used extensively during the project. The parallel use of different parts of the test bed has also been a useful feature. In addition, the test bed includes general purpose computers, primarily a VAX 11/780, for data analysis, algorithm development, and software development.

Section 3.2 describes the test-bed hardware and Section 3.3 describes the system software with emphasis on the basic Nodal Run Time System (NRTS) that supports tracking and communication functions. Application level distributed surveillance software is covered separately in Section 7. Section 8 provides additional information about the communication hardware and software. Section 10 contains additional information about the software development process and tools.

3.2. Test-Bed Hardware

Figure 3.1 is a block diagram of the entire DSN test bed. It consists of 11 separate computer systems identified as nodes, three general purpose computers (a VAX and two UNIX workstations) and communication equipment (Ethernet and microwave radios) to interconnect the elements. Some of this equipment is installed within a DSN laboratory area and some of it is installed in three vehicle-mounted mobile enclosures. Except during field operations all nodes and other computers are interconnected by a single physical Ethernet. In this configuration the vehicles are parked adjacent to the laboratory.

The test-bed vehicles, one of which is shown in Figure 3.2, are used to remotely deploy nodes for experiments. When they are deployed for data collection they operate independently without internodal communication. For live distributed-tracking experiments, the remotely deployed mobile systems are interconnected with each other and with the laboratory portion of



NOTE:

(A) NODES 1-3 IN FIELD DEPLOYABLE TRUCKS
(B) NODES 1-6 HAVE ATTACHED ACOUSTIC ARRAYS
(C) NODES 7-9 CAN BE USED AS INTERNET GATEWAYS
(D) NODES 1-9 CAN BE USED FOR TRACKING
(E) TV CAN BE INSTALLED IN MOBILE UNIT
(F) WORKSTATION CAN BE INSTALLED IN MOBILE UNIT

Figure 3.1. Lincoln Laboratory DSN test-bed system.

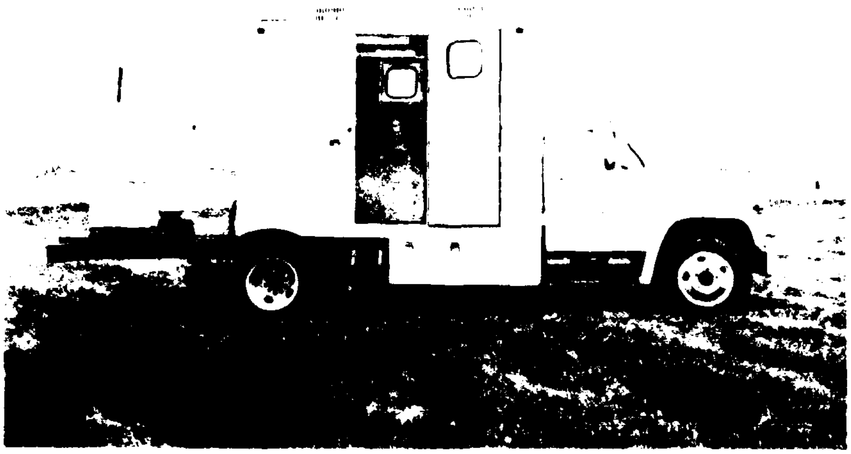
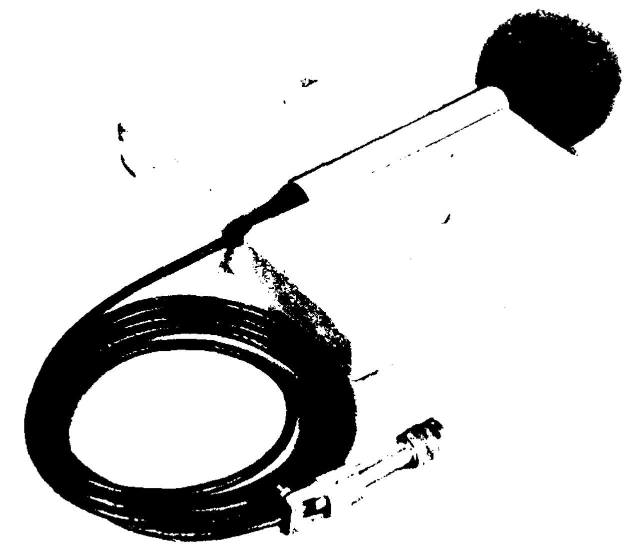


Figure 3.2. One of three experimental vehicles used for field deployment of DSN test-bed elements. Box at the rear contains an acoustically quieted generator.

the test bed by commercial microwave equipment. Each remotely deployed vehicle contains a short internal Ethernet that interconnects equipment within the vehicle enclosure. To maintain communication with the laboratory portion of the test bed, the vehicles are deployed within line-of-sight to microwave antennas installed on the roof of the DSN laboratory. The sites have usually been within 3 km of the laboratory although the communication system will operate up to 10-km ranges.

It is also possible to configure the vehicles to perform live distributed-tracking experiments while completely separated from the DSN laboratory, although this has not been done. It requires that one of the UNIX workstations be moved from the laboratory to one of the vehicles. In this configuration the vehicles could be deployed anywhere, with separations of up to 10 km, and used to perform DSN experiments. The workstation would serve to control the experiment and display results.

Microphones and TV cameras are used as the sensing elements. Figure 3.3 shows one of the microphones, a standard GenRad laboratory quality electret microphone with a flat frequency

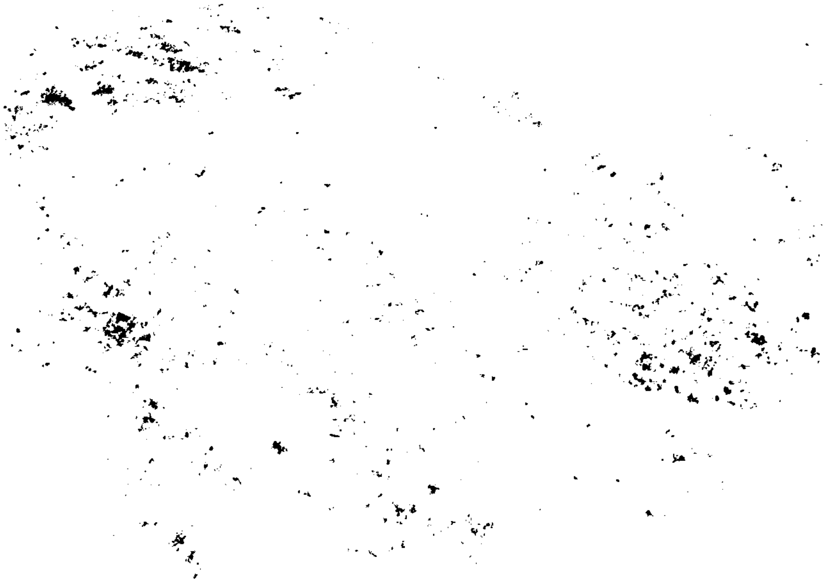


104059-3

Figure 3.3. Microphone with preamplifier attached to wooden block for ground-level field deployment 2-in windscreen covers microphone head.

response from 20 Hz to 20 kHz, with a foam wind screen attached. The microphone is attached to an 8-in-square wooden block with spikes attached to hold it in place in the ground when it is deployed. Figure 3.4 shows nine such microphones deployed in a grassy area as a small acoustic array that will be used for aircraft detection and acoustic direction finding. The distance between the most separated microphones in the the photo is 6 m. There are enough microphones to deploy six such arrays at one time.

The microphones require periodic calibration and testing during periods of prolonged use. For many purposes we have found it adequate to permanently locate a small loudspeaker near each array and to use it to generate a test-tone. However, we also individually calibrate each microphone using a laboratory grade calibrator designed for use with the microphones. These procedures have been satisfactory for the experimental test-bed system but better techniques would be needed for an operational system.



104059.4

Figure 3.4. Field deployment of a nine-microphone array.

Figure 3.5 shows one of the two TV cameras. Each camera is enclosed in a waterproof case and the mount is designed for outdoor operation. The mount azimuth and elevation as well as the camera zoom can be remotely controlled.

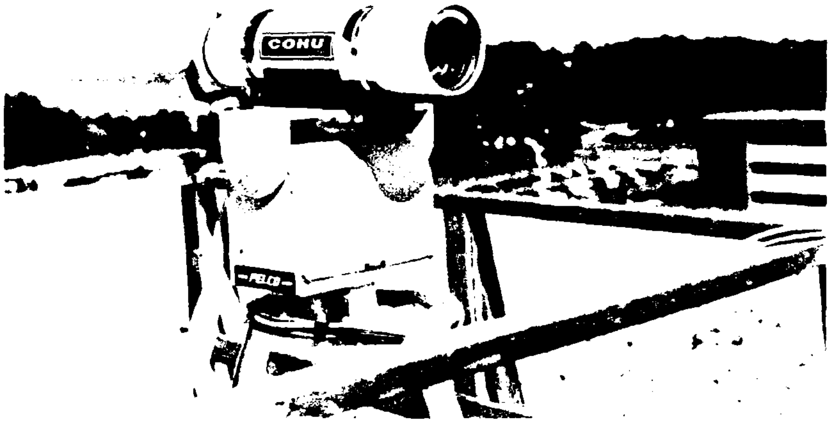


Figure 3.5. Weather protected TV camera installed on remote control mount.

Each of the nodes is built around a basic unit called a Standard Nodal Computer (SNC). The SNC is a small but flexible multiple computer system that uses commercial single board computers on a Multibus. The computers are early versions of the Stanford University Network (SUN) computer boards containing 256 bytes of memory and a Motorola MC68000 processor. The MC68000 boards were manufactured by Pacific Microsystems and installed in rack-mounted SNC units designed and built at Lincoln Laboratory. Figure 3.6 shows one of these SNC units with boards installed. Each SNC unit can accommodate up to three processor boards. It should be noted that commercial hardware and software technology progressed quickly during the DSN project and that standard commercial workstations and software could now be used in place of the less powerful, more customized hardware and software that were developed for the test bed.

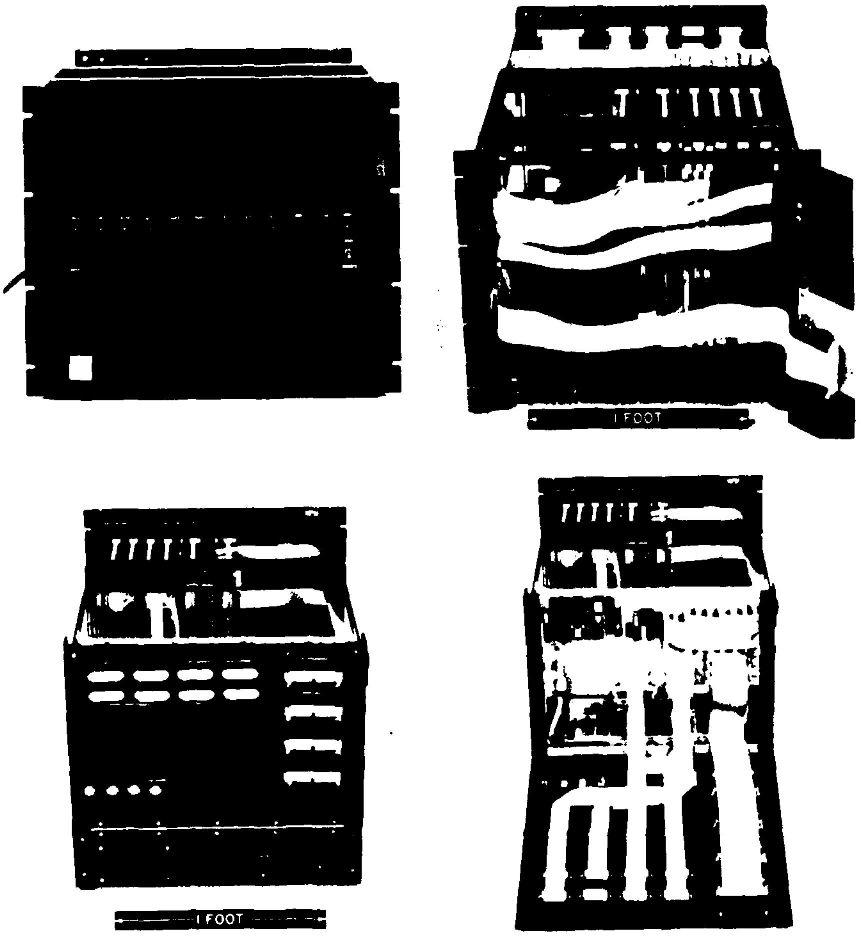


Figure 3.6. Standard Nodal Computer (SNC) used in the DSN test bed.

There are three kinds of nodes in the test bed. These are acoustic/tracking nodes, TV nodes, and gateway nodes. Acoustic/tracking nodes provide acoustic azimuth measurements using small acoustic arrays. They also perform tracking and can be used without an attached acoustic sensor subsystem. When operated without the acoustic subsystem they use simulated acoustic data. TV nodes collect visible TV images and derive measurements of aircraft azimuths from the images.

Even when radios are used to interconnect physically separated parts of the test bed, the communication system still appears as a single logical Ethernet to the acoustic/tracking and TV nodes. This is done by using a few nodes as gateway nodes that transfer data between Ethernets and radios. Only the gateway nodes are aware that the system does not consist of a single Ethernet. One gateway node is required for each physical Ethernet that is interconnected with others by radio. The function of gateway nodes is to provide a specialized communication service that isolates the rest of the nodes from communication complexities.

When the vehicles are not deployed remotely, they are attached directly to the laboratory Ethernet and neither radios nor gateways are required. A single physical Ethernet can be used to interconnect all nodes, and gateway nodes can then be reconfigured to serve as acoustic/tracking nodes.

Figure 3.7 is a block diagram of a standard acoustic node. It contains three MC68000 single board computers, additional memory and floating point processor boards, an Ethernet interface, a floppy disk system, an acoustic Signal Processing Subsystem (SPS) and some simple reset hardware to expedite remote recovery from catastrophic nodal failures.

The three MC68000 computers are the heart of the system. They handle all tracking and communication tasks. They each contain private memory and operate independently of each other. They interact only by message passing through the separate 512-kB shared memory board. The memory board also serves as the communication buffer between the node and the Ethernet, which is the primary means by which nodes interact with each other. The floating point board provides floating point hardware for all three of the MC68000s.

The floppy disk system on each node has three primary uses. First, nodal software is stored on the disk. Second, the disk can be used to record data during experiments. Third, simulated acoustic detections and direction estimates can be stored on the disk and used as input for controlled experimentation.

A serial port on one of the MC68000 boards is attached to a remote reset device and a 9600-baud line which is attached in turn to either a local console or a remote computer by an RS-232 line. This processor is denoted as P1. The other two are P2 and P3. The remote reset device allows a local or remote operator low level control of the node to reset and reboot it when there are catastrophic failures. The remote reset device simply monitors the line for a special reset code which causes the processor to be reset. With P1 operational, P2, P3 or the SPS can be reset by a secondary reset device.

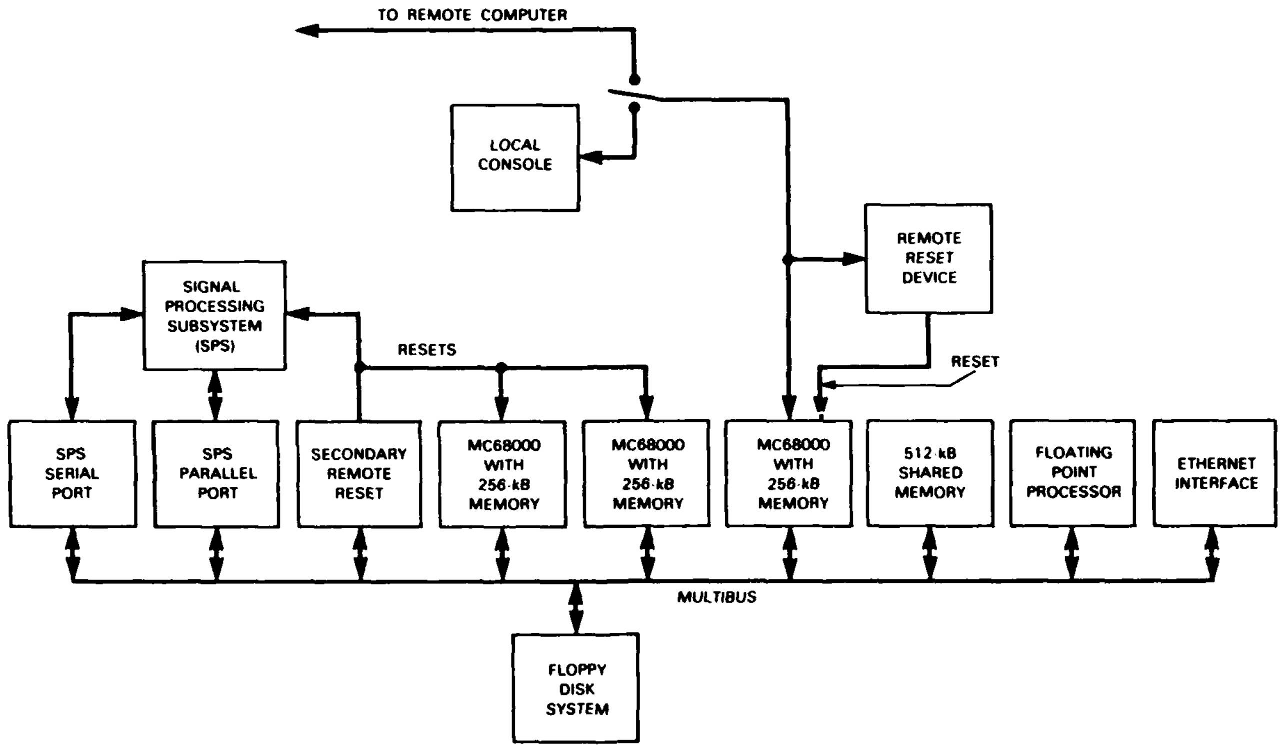


Figure 3.7. Standard acoustic node for the DSN test bed.

87882-4

The SPS communicates with the rest of the system using one serial port and one parallel port. The parallel port delivers acoustic detections, including estimates of source directions and power levels, to the tracking software installed in the MC68000s. The serial port is used for control purposes.

Figure 3.8 provides more details about the Signal Processing System. It is used for data acquisition as well as signal processing. The PDP 11/34 is the intelligent core of the SPS. In data acquisition mode it collects acoustic data from the A/D system and records it on a standard 1600-BPI digital tape. In real-time signal processing mode it obtains digital acoustic data either from the A/D or from tape and uses the FPS-120 array processor to obtain the acoustic detection data required by the tracking system. The disk system is only used for software storage.

Data obtained from the A/D system are time stamped using a time signal, accurate to a few milliseconds, provided by a satellite time clock. Thus data recorded on tape have permanent absolute time stamps. In addition, and this is important, detections provided to the tracking system during real-time experiments are correctly time-stamped. These time stamps are necessary

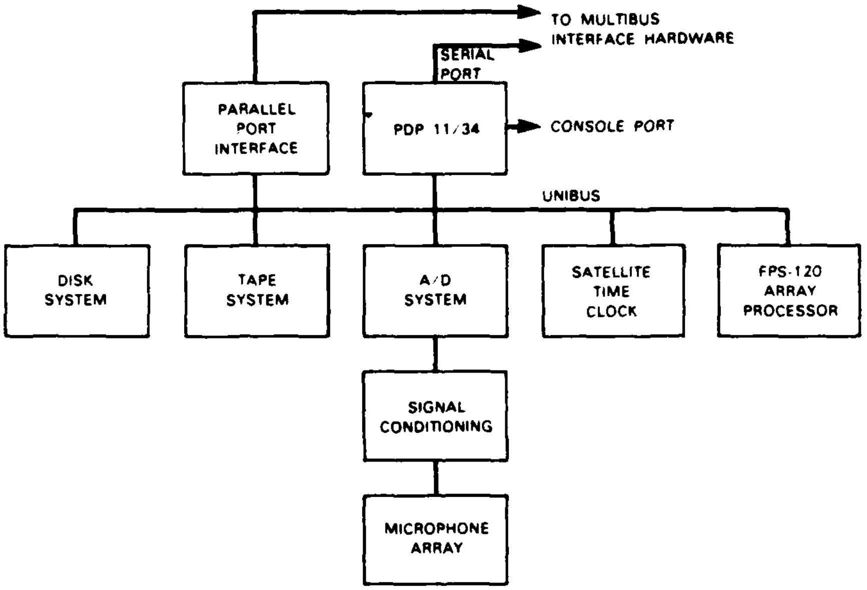
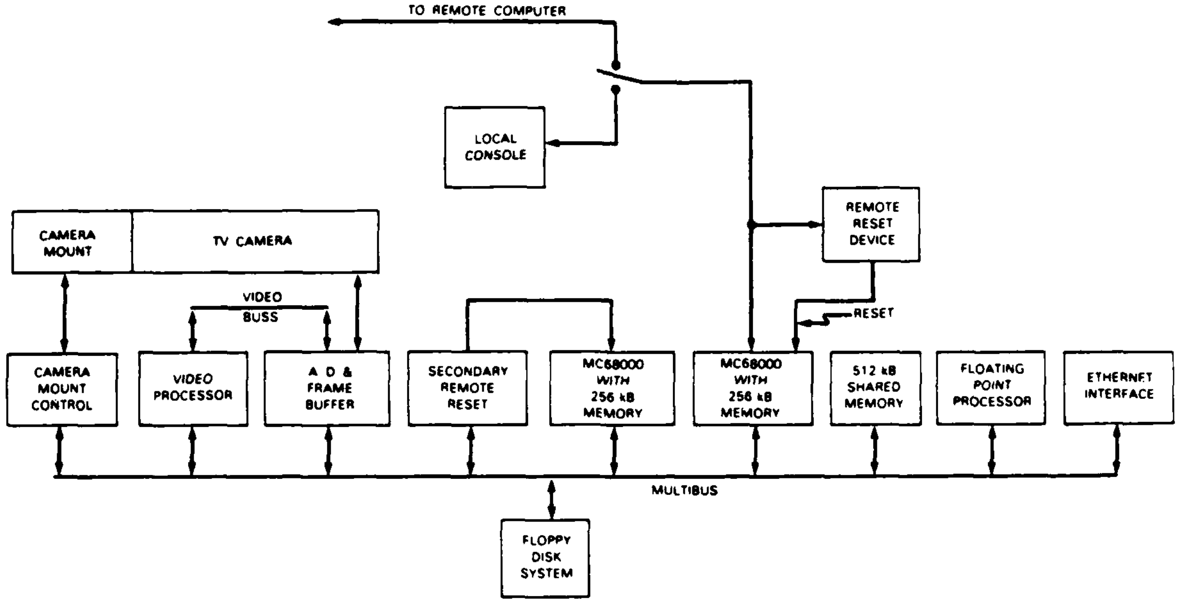
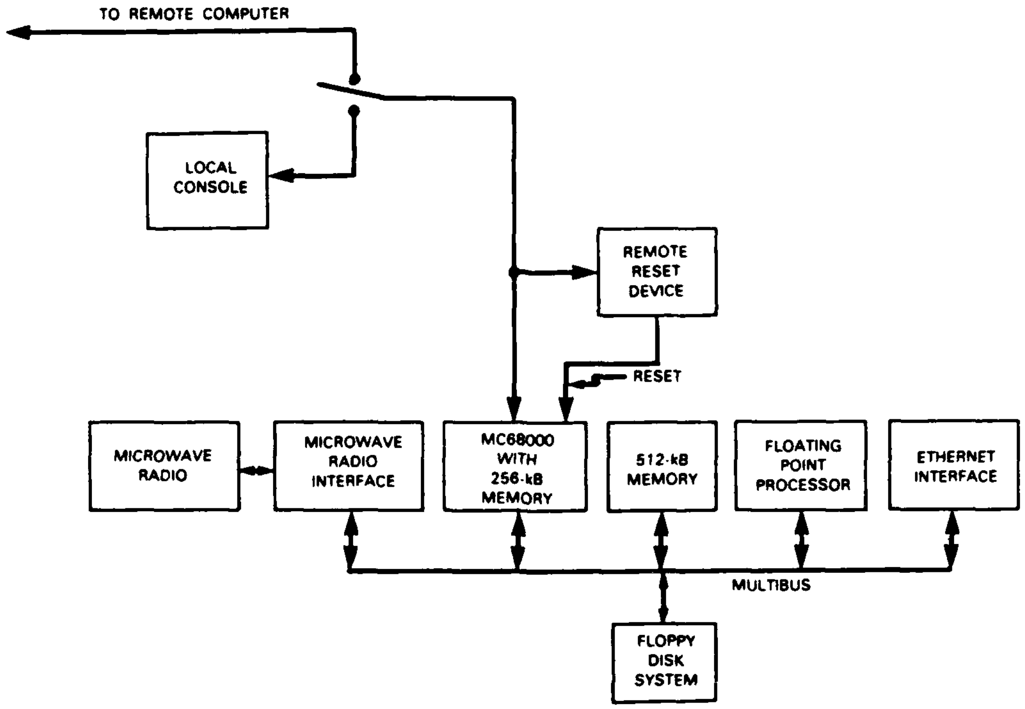


Figure 3.8. Acoustic Signal Processing Subsystem (SPS).



87882-6

Figure 3.9. Standard TV node for the DSN test bed.



87882-7

Figure 3.10. Gateway node for the DSN test bed.

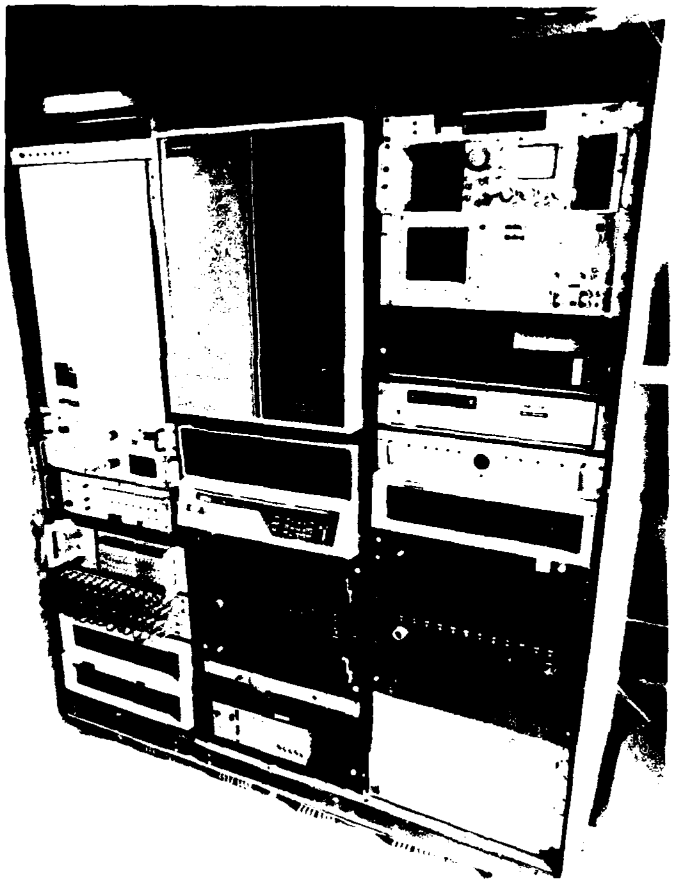
for the distributed system to correctly combine data from several sites. The small antenna used to obtain the signal from a geosynchronous satellite can be seen at the top left of the instrument enclosure in Figure 3.2.

The microphone array can contain up to 12 sensors. Signal conditioning equipment amplifies the microphone outputs and passes them through an anti-alias filter with a 750-Hz cutoff frequency. Signals are then sampled at 2048 samples per second. The A/D system is a "gain ranging" system that provides 95 dB of operating range. Each sample consists of a 14-bit (13 bits plus sign) digitized signal and two additional bits indicating which of four gain values apply. The gain is adjusted for every sample to keep the A/D converter input as large a possible but less than 80% of full scale. This pseudo floating-point sampling method minimizes system gain adjustment problems that might otherwise arise. It does require additional processing to convert the A/D output into a form that is more readily usable. For real-time signal processing this conversion is performed within the FPS-120 array processor. There is no format conversion during data collection. Conversion of the collected data is done as needed when the data are used. Gain ranging for every sample is possible for an acoustic system because the overall data rates are modest, compared with data rates that might be required for other sensor systems such as radars. For example the digitization of 12 acoustic channels at 2048 samples per second involves only about 25,000 conversions per second and a single A/D is used with a multiplexer to do the job.

Figures 3.9 and 3.10 show the other two standard node configurations. They should be compared with the standard acoustic/tracking configuration of Figure 3.7. The difference between the acoustic/tracking and the TV configurations are that the acoustic SPS is replaced by TV-related equipment, and one MC68000 board is removed because two processors are sufficient for all the TV-related functions. The TV interface to the Multibus consists of three functional elements. One is the camera mount control used to control the mount and camera zoom as well as to measure the actual position of the mount and zoom. A second is the A/D and frame buffer unit that serves to digitize and store TV images. The third is the video processor which performs signal processing operations on the digitized TV images. The video processor and frame buffer are interconnected by a private high speed digital bus to avoid excessive Multibus traffic.

The gateway configuration shown in Figure 3.10 is an even more stripped-down version of a standard node. It contains a single MC68000 board, a microwave radio interface, and microwave radio equipment.

Figure 3.11 is a photograph of a complete acoustic/tracking node and a gateway node as installed in one of our vehicles for remote operation. The left rack contains the FPS-120 array processor, acoustic signal conditioning, and A/D equipment. The center rack contains the digital tape drive and PDP 11/34 for the SPS as well as the gateway SNC chassis and T1 multiplexer. At the bottom of the center rack is some of the microwave radio electronics. The right rack contains the acoustic SNC chassis plus some microphone power supplies and ancillary test equipment.



104059-7

Figure 3.11. Acoustic/tracking node and gateway node installed within experimental vehicle.

The disk in the middle of the right rack is for the SPS and the two at the bottom of the left rack are for the two SNC systems. All three racks of equipment are shock mounted to the floor to protect them during transport.

3.3. Test-Bed Software Systems

Three operating systems are used in the Lincoln Laboratory DSN test bed: UNIX, RSX11, and NRTS. NRTS is a system that was developed as part of the DSN project to serve as the operating system for the standard nodes. (A new test-bed system could probably use, with minor modifications, more recent commercial computer workstations, including their operating systems, network communication capabilities and software development tools, to replace the SNC systems and NRTS.) UNIX is the operating system for the general purpose VAX 11/780 and user workstations. It is also the primary environment for software development. RSX11 is a Digital Equipment Corporation real-time operating system that is used by the acoustic SPS systems. Both UNIX and RSX11 are standard and should require no further explanation. The following provides more details about NRTS, primarily from the user's point of view.

Figure 3.12 illustrates the basic user interface in NRTS. There are five kinds of service provided to users as indicated. These are process creation and control, input/output, string manipulation, memory management, and timing and clock-related services. The interface was designed to be similar to that provided by the UNIX operating system. Application programs are written in the C language and all services are accessed by subroutine calls with UNIX-like syntax.

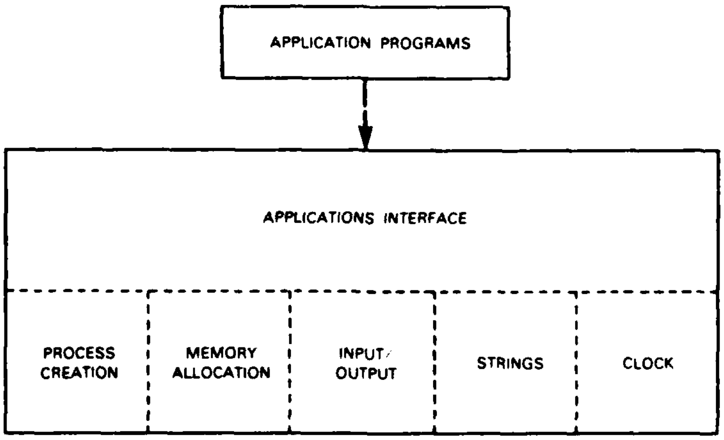


Figure 3.12. Node Run Time System (NRTS) application programming interface.

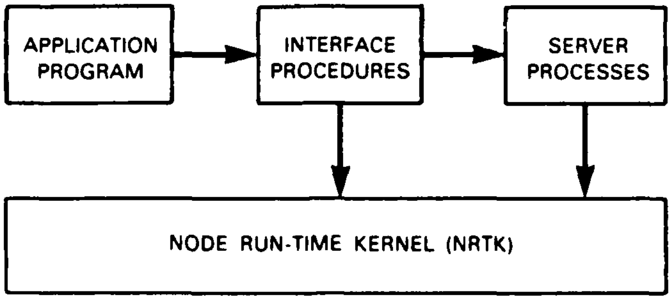


Figure 3.13. NRTS organization.

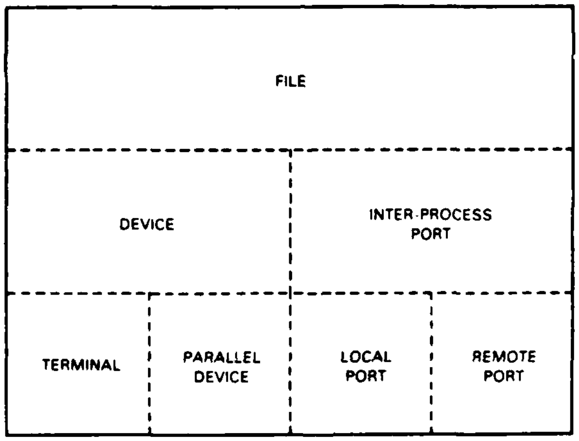


Figure 3.14. Hierarchical structure of the file oriented input-output system.

There are three parts to NRTS: the kernel , the servers , and the interface as indicated in Figure 3.13. NRTK , the nodal run-time kernel, is the core of of the system. It provides facilities for process creation and control, process synchronization, primitive I/O, and basic memory allocation. The servers are processes that serve the applications. These include device servers and the directory server for naming devices and inter-process files. Servers are not accessed directly by the application but through a library of interface subroutines.

NRTS programs consist of a set of load modules with at most one load module per SNC processor. Each module consists of procedures and data. At system initialization a single user process is created. This process is a user-defined procedure that the user has selected as the first process to run. It can perform any task of the user's choosing. It usually initializes user data structures and creates additional user processes. There is a single initialization process for each SNC, independent of how many processors contained in the SNC. It can create processes within the SNC where it is running but not in any other SNC. Aside from the initial main process, no inherent process structure is imposed on the load module. Processes are started and terminated at run-time via user requests.

Process initiation involves three steps. A new process is created using a "fork" system call. Parameters for the process, such as stack size, are set via a separate command and finally, the process is set to run a particular procedure through an "exec" system call. A process terminates when its outermost procedure returns or when it calls "exit." However, except when a problem occurs, most processes in the real-time test-bed environment do not terminate. The NRTS way of handling processes is similar to that of the UNIX operating system.

Although NRTS does not provide a general purpose file system, its I/O system uses the UNIX file model. The I/O system supports devices and inter-process ports as special files as illustrated in the file hierarchy shown in Figure 3.14. Devices include serial devices such as terminals and parallel devices such as floppy disk controllers and the parallel interface between the SNC and SPS. Programs may open, read, and write ports and devices through a library patterned after the UNIX standard I/O library. A simple but essential broadcast facility is provided as well.

Inter-process port files are functionally similar to UNIX pipes. Data are passed through ports on a first-in-first-out basis. Output routines are provided to pass data to a port as discrete messages. Other input routines remove messages from ports and deliver them to the user program. There is a maximum number of messages which may be on the port at a given time. Processes attempting to write to a port when the number of messages is at the maximum are delayed until another process reads from the port. When there are no messages on a port, a reader is delayed until another process writes to the port. A port may have several readers and writers. Message delivery is assured only if the writer and the reader are within the same SNC.

Files in NRTS have UNIX-style pathnames. Each node has its own name tree and the entire network has a name tree as well. The network name tree is composed of the nodal name trees grafted on to a single network level tree whose leaves are the names of the nodes. Full pathnames are always used although names may be given relative to the nodal name tree or the network name tree.

One standard set of files is created when the system is initialized. Processes can then create files with the restriction that a process can create a file only within the node in which it is running. This is a "local" file. Remote files are those on other nodes. The creating process must provide the name of the file and the maximum number of allowed messages if it is a port. Once created, files are opened, closed, read and written in much the same way as in UNIX. Local files can be opened for reading or writing; remote files can be opened for writing only. File open commands return a FILE type variable. This variable is a pointer that identifies the file for read and write routines.

NRTS supports many I/O routines with syntax and semantics identical to UNIX. These are: fgets, fputs, fscanf, fprintf, fread, fwrite, and ungetc. The string format routines sscanf and sprintf are also available. In addition, NRTS provides the procedures fgetc and fputc, which have syntax and semantics identical to the UNIX macros getc and putc.

There are three standard pre-defined FILE variables in each NRTS load module. These are stdin , which is used for input, and stdout and stderr, which are used for output. These are similar to the identically named standard files in UNIX. There are minor complications in their usage due to the distributed nature of the test bed but their purpose is the same as that of the UNIX standard files.

Message broadcasting is an essential NRTS I/O capability. Application software uses it to exchange distributed tracking data. The application programs create ports to receive broadcast messages by creating files in a special directory called "/cast." Reading from such a broadcast port is the same as for any other port. Writing to a broadcast port is done by a routine with syntax and semantics similar to the UNIX fwrite. The arguments include a character string representing the the name of a file in the /cast directory, and the message is sent to all ports in the network with the name /cast/<string>. If there is a local port with that name, the message is placed there; in addition, the message is broadcast to other nodes. Nodes that receive the message place it on a port with the same name if such a port exists. The choice of broadcast message file names is left to the user.

NRTS provides several other services at the applications level. These include facilities for node identification and memory allocation, as well as some clock-related services. Applications may occasionally utilize the inter-process synchronization and communication facilities of semaphores and queues. Node identification routines allow a process to determine the identity of the node in which it is running. Allocation and deallocation of memory are by routines that have identical syntax and semantics to the UNIX procedures for the same purposes. Memory is always allocated from a processor's local memory. The shared memory is not directly available to application programs.

NRTS provides two clocks which differ from the UNIX system clock. One is a system-time clock that starts when NRTS is started and can be read or waited upon for a user-set interval. The other is an experiment time clock. The user can start, stop and set this clock as well as read it or wait until a user-set time. The user can also set the rate at which this clock 'ticks' relative to the system-time clock when it is running. The latter clock is used for most application purposes. In particular, it provides the 'real' time standard for the applications software while an experiment is in progress. The other clock is used when fixed system time delays are needed for some reason.

4. ACOUSTIC ARRAY PROCESSING ALGORITHMS

4.1. Introduction

The use of multiple small microphone arrays to detect and track low-flying subsonic aircraft required the development of detection and direction finding algorithms for the arrays. The sensor array direction finding problem is well known and has been extensively investigated [1-5] although not for the DSN aeroacoustic aircraft surveillance application. The array is usually assumed to be in the far-field of the sources and the signal at the array is modeled as a planewave for each source. Each planewave is characterized by a direction of propagation, a speed, and a temporal frequency spectrum. Sensor array processing often reduces to the estimation of one or more of these planewave parameters from the space-time wavefield sampled by the array.

The most common detection and direction finding method for sensor arrays is "beamforming"; also called "delay-and-sum" processing. In this technique, illustrated in Figure 4.1, a source direction is assumed and time-delays are calculated, relative to a reference sensor in the array, for the assumed signal passing over the array elements. These delays are applied to the time series received at the array and the delayed signals are added together to form a "beam". The idea is that if the delays are correct then the signal will add coherently and noise will add incoherently.

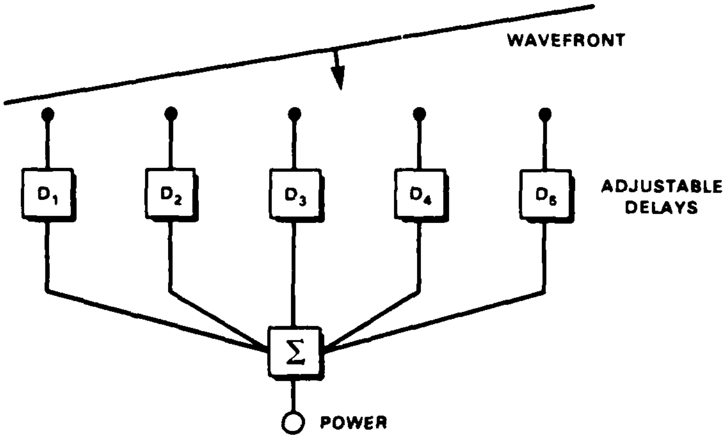


Figure 4.1. Delay and sum beamforming.

Many beams are formed, covering the range of possible directions for signal arrival, and a power detector is applied to each beam. Beams that maximize power are assumed to correspond to the correct signal directions.

A variation of this method is the "frequency domain beamforming method" which can be applied when the signal is narrow band or if one is willing to search over the entire band of frequencies for wideband signals. For each frequency, the frequency domain method reduces to a spectral analysis problem in the spatial wavenumber domain and thus is a "frequency-wavenumber" method. One advantage of the frequency domain method is that it can easily be modified to act as an optimum adaptive direction finding method [6]. The modified algorithm is often referred to as the Maximum Likelihood Method (MLM) of frequency-wavenumber analysis.

The DSN project initially chose to use the MLM version of the frequency-wavenumber method and to apply it at a subset of frequencies within the band where signal-to-noise ratios were expected to be largest. The real-time implementation was limited by computational constraints to a maximum of eight frequencies for each analysis interval. The frequency selection was done by performing a spectral analysis on a single microphone signal and choosing the frequencies corresponding to peaks in the spectrum. This algorithm was implemented and used during early phases of the project.

The frequency selection for the initial algorithms was done without benefit of any signal-to-noise gain from the microphone array. This tended to cause the algorithm to become locked onto loud signals with many harmonics and to not find secondary signals. The alternative of performing the spatial analysis at every frequency would have increased the signal processing load by another order of magnitude and exceed the capacity of the test-bed hardware by an equal amount. We also could have taken the time-domain beamforming approach but that would also have exceeded the real-time capacity of the general purpose digital hardware in the test bed. Small special-purpose digital hardware could have been developed for either alternative but this was beyond the scope of the project.

The need for improved detection and direction finding performance within existing hardware computational constraints motivated research that led to the development of a new kind of array processing algorithm. The result is a computationally efficient method for estimating the bearings of multiple planewave sources with unknown frequency characteristics within a wide band. The method uses the zero-delay covariances of the complex analytic representations of the sensor signals of a planar array to estimate the wavenumber spectrum. This spectrum determines the source bearings uniquely over the entire 360° range. The computational requirements of this method are of the same order as frequency-wavenumber analysis on a planar array at a single frequency.

The following Sections 4.2 through 4.4 provide more information concerning the basic concept, describe the wideband algorithms, and show examples of experimental results obtained using the algorithm. This material is covered in more detail in References [7] and [8].

4.2. Theoretical Basis for Wideband Array Method

The wideband array processing method is based upon the basic properties of covariance functions and frequency-wavenumber representations for zero-mean homogeneous random fields. Let $s(t,x)$ be such a field where t and x represent time and three-dimensional space, respectively. The space-time covariance function of $s(t,x)$ is given by

$$R(\tau,\rho)=E[s(t,x)\cdot s^*(t-\tau,x-\rho)] \tag{4.1}$$

and the frequency-wavenumber spectrum is the multidimensional (time and space) Fourier transform of $R(\tau,\rho)$:

$$P(k,\omega)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} R(\tau,\rho)e^{-j(\omega\tau-k\cdot\rho)}d\tau\, d\rho \tag{4.2}$$

where $k\cdot\rho$ is the scalar product between the vector quantities. Consistent with this we define the zero-delay wavenumber spectrum $P_{ZD}(k)$ as

$$P_{ZD}(k)=\int_{-\infty}^{+\infty} R(0,\rho)e^{jk\cdot\rho}d\rho. \tag{4.3}$$

Note that $P(k,\omega)$ and $P_{ZD}(k)$ are related through the integral

$$P_{ZD}(k)=\int_{-\infty}^{\infty} P(k,\omega)d\omega. \tag{4.4}$$

That is, the zero-delay wavenumber spectrum is obtained by collapsing $P(\omega,k)$ onto the wavenumber space k.

The underlying theoretical basis for the wideband method is the fact that, for a planar array, the zero-delay wavenumber spectrum can be used to determine the bearing of planewave sources.

Let $s(t-\alpha\cdot x)$ be a planewave propagating in three dimensions. The vector α points in the direction of propagation and has magnitude equal to one over the magnitude of the propagation velocity of the planewave. The position vector x is a three-dimensional vector. If the bearing and elevation angles of the vector α are θ and ϕ, respectively, then the projection of the wave onto a two-dimensional plane P is the planewave $s(t-\alpha_P\cdot x)$, where α_P is the projection of α onto P and x is the two-dimensional vector position in P. At any instant of time t_0, the wavefield in the plane P has a constant value along any line orthogonal to θ, the direction of propagation of the planewave. The spatial autocovariance function also has the same property. Therefore, it can be shown that the zero-delay wavenumber spectrum, which is the Fourier transform of this two-dimensional spatial autocovariance function, is a distribution (delta function) in the two-dimensional wavenumber space which is confined to a line passing through the origin with an azimuthal angle θ. Furthermore, the value of the distribution along the line is proportional to the temporal spectrum of the planewave scaled in frequency (see Figure 4.2). If the signal has a temporal spectrum $S(\omega)$, then the radial distribution is proportional to $S(\omega/|\alpha_P|)$. Thus the zero-delay wavenumber spectrum contains temporal spectrum information as well as bearing information. The magnitude of the projected slowness vector α_P is equal to the cosine of the angle of elevation multiplied by the inverse of the wave speed. If the wave speed and the temporal spectrum are known, the elevation angle of the plane wave can be determined by stretching the radial wavenumber spectrum to match the temporal spectrum.

Observe that if the planewave is real, the temporal power spectrum is even. In that case, the distribution along the axis will be the same whether the planewave came from bearing θ or $\theta+180^\circ$. This problem can be overcome by temporally prefiltering the sensor signals to obtain complex analytic representations which :ontain no negative frequencies. This way, if the planewave is coming from bearing θ, there is zero contribution from the direction $\theta+180^\circ$ in the wavenumber space.

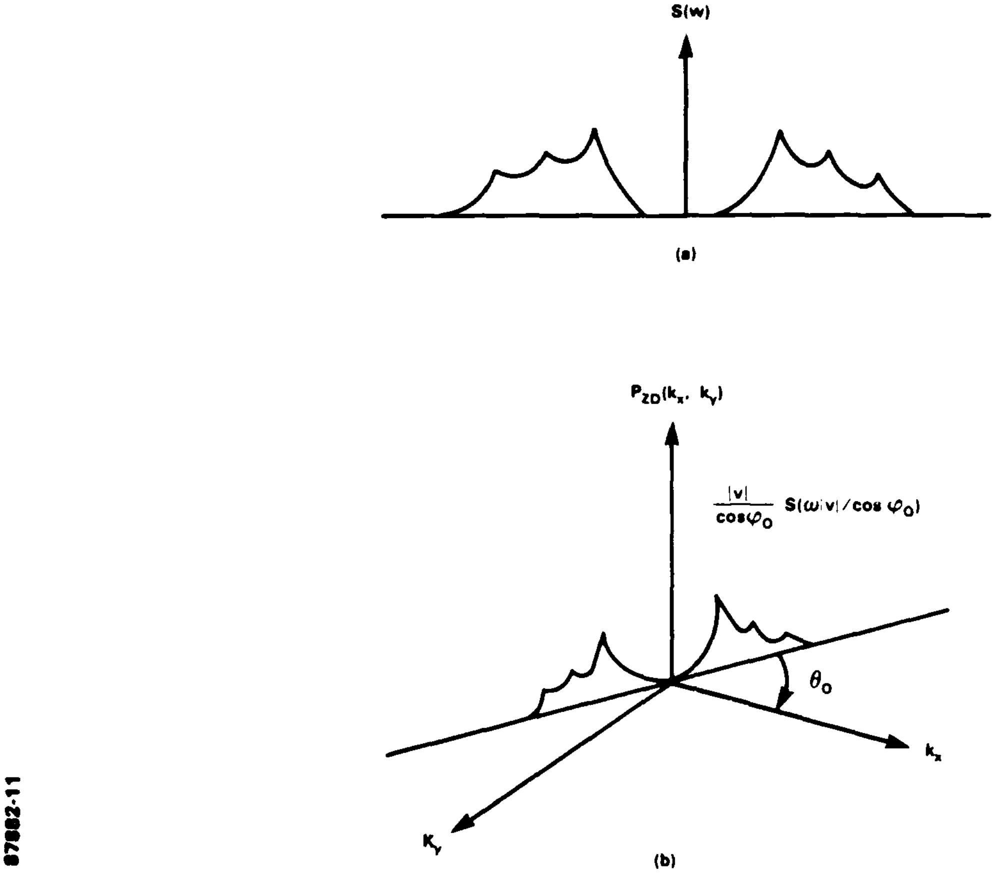


Figure 4.2. Spectral mapping property of the zero-delay wavenumber spectrum. (a) Temporal power spectrum of a planewave coming from the direction with elevation ϕ_0 and bearing θ_0. (b) The zero-delay wavenumber spectrum of the same planewave on the array plane.

The homogeneous field $s(t,x)$ formed by the sum of uncorrelated planewaves has a zero-delay wavenumber spectrum which is the sum of the zero-delay wavenumber spectra of the component planewaves. Thus for multiple planewave sources the zero-delay wavenumber spectrum has nonzero contributions only in those radial directions corresponding to source directions.

4.3. Wideband Algorithm for DSN Planar Arrays

The structure of the wideband algorithm for detection and direction finding is shown in Figure 4.3. The first step of the algorithm is to calculate samples of the complex analytic representations of band pass versions of the sensor signals. These samples are then used to estimate the

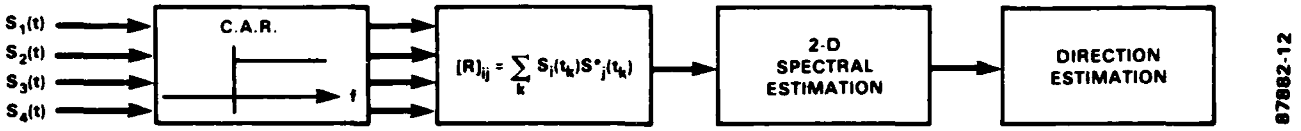


Figure 4.3. Schematic representation of wideband array algorithm. The four inputs $s_i(t)$ correspond to an assumed four-sensor array.

zero-delay covariances of the sensors. Then the zero-delay wavenumber spectrum is estimated from the zero-delay covariance estimates. Any two-dimensional wavenumber estimation algorithm can be used for this purpose. Finally the wavenumber spectrum is processed to detect targets and determine their directions.

The complex analytic representation (CAR) of a real signal is a complex signal whose real part is the original signal and whose imaginary part is the Hilbert transform [9] of the original signal. One method for obtaining the complex analytic representation of a signal is to take its Fourier transform, set the negative frequency sideband to zero, and then to take the inverse Fourier transform. For discrete-time signals, the equivalent of the negative frequency sideband is the interval $(-\pi,0)$. However, if only a small subset of the samples of the CAR is required this is not computationally efficient.

A computationally efficient algorithm for estimating M uniformly distributed samples of the CAR signal from a finite-length record of a sensor signal $s(n)$ was developed. The finite-length record is divided into M (possibly overlapping) N point regions with one of the M CAR estimation locations near the center of each region. Each of the CAR samples is then estimated from the corresponding N point region. This could be done by taking the DFT of the N point region, setting the negative frequency sideband to zero, and taking the inverse DFT. However, since we are interested in the value of just one CAR sample per region, the computation involved can be considerably reduced. In particular, the operation of taking an N point DFT, setting the negative frequency sideband to zero, and then evaluating the N point inverse DFT at just one location can be reduced to a single N point linear combination of the samples of $s(n)$. If the N point region is in $0 \le n < N$, then the CAR sample $c(n_o)$ of $s(n)$ is given by (assuming N is even) :

$$c(n_o)=\sum_{n=0}^{N-1} a_n s(n) \tag{4.5a}$$

where

$$a_n = \begin{cases} N/2 & \text{if } n = n_o \\ e^{-j\pi(n-n_o)(N-2)/2N} \dfrac{\sin[\pi(n-n_o)/2]}{\sin[\pi(n-n_o)/N)]} & \text{if } n \neq n_o \end{cases} \tag{4.5b}$$

The resulting linear combination is the estimate of the CAR sample of $s(n)$ at $n = n_o$.

An additional feature can be added to the above algorithm. In some cases it may be desirable to find the complex analytic representation of $s(n)$ filtered through a pre-specified positive frequency passband. If this passband can be specified as $2\pi K_1/N \leq \omega < 2\pi K_2/N$ where K_1 and K_2 are integers and $0 \leq K_1 < K_2 \leq N/2$, then the only change to the CAR algorithm is that the coefficients a_n are now specified by:

$$a_n = \begin{cases} K_2 - K_1 & \text{if } n = n_o \\ e^{-j\pi(n-n_o)(K_1+K_2-1)/N} \dfrac{\sin[\pi(K_2-K_1)(n-n_o)/N]}{\sin[\pi(n-n_o)/N)]} & \text{if } n \neq n_o \end{cases} \tag{4.6}$$

The next step in the wideband array processing method is to estimate the zero-delay covariance of the complex analytic representations of the sensor signals. Let $c_i(n)$ denote the mth CAR sample of sensor i that is obtained using the above algorithms. An estimate of the zero-delay covariance between sensors i and j can then be obtained by averaging the product $c_i(n)c^*{}_j(n)$ over the M values at which the CAR signals have been estimated. Thus

$$r_{ij} = \frac{1}{M} \sum_{n=1}^{M} c_i(n) c_j(n) \tag{4.7}$$

is an estimate of the i,j element of the zero-delay covariance matrix R.

Several issues are involved in selecting the total number of samples, the best value for M, and the block size. A related question is: Why not use all the values of the complex analytic representation? The principal factors are the bandwidth of the process, the sampling rate, and the time period over which it is stationary.

First consider stationarity. One aspect of stationarity is whether sounds appear to be coming from approximately fixed directions. Is the direction change during the observation interval smaller than the directional determining capabilities of the microphone array? If not, the wavenumber distribution for the source will be smeared out. The signal will then be more difficult to detect and there will be more ambiguity associated with the direction estimate. The stationary times can be several seconds for distant aircraft sources and only a fraction of a second for nearby aircraft. For most experiments we have used 1- to 2-s time intervals. This is a compromise between the long intervals that could be used to detect distant aircraft and the short intervals needed for close high-speed aircraft. But it is good for a wide range of aircraft speeds and distances. An adaptive signal processing system might provide some improvement but at the cost of considerable computational load and complexity. Background noise can also be non-stationary, but moving aircraft source considerations normally dominate the selection of a measurement interval to estimate R.

Given the measurement interval, the number of complex analytic values to calculate R and the best block size to use depend upon the sampling rate and bandwidth of the signals. If the sampling rate is exactly matched to the bandwidth of the signal (the signal fills the entire Nyquist band) then all CAR samples for one time series are independent of each other, including adjacent samples. In this case, assuming that the method described above is used to calculate one CAR sample for each block, the best estimate of R would be obtained by overlapping the blocks almost completely and using very long block sizes. The large block sizes are required to calculate the CAR for broadband signals. The large overlap is required to minimize the statistical fluctuations in the estimate of R. In this case, M becomes the total number of data samples and every CAR sample is used. But data are often oversampled in time so the signal bandwidth fills only a fraction of the Nyquist band. In this case a good approximation to the CAR can be obtained with shorter blocks, and samples close to each other in time are no longer statistically independent. The first of these facts pushes in the direction of using shorter blocks to reduce the computational load with no significant performance changes. The second fact pushes toward not using all the CAR samples to get R. When the samples used to estimate R are highly correlated, the computational cost of reducing statistical fluctuations can be very high. It is often true that an uncorrelated subset of the values can be used with only a slight increase in statistical fluctuations and large computational savings. The real-time test bed has generally used abutting blocks or blocks with 50% overlap.

The next step in the signal detection and direction finding process shown in Figure 4.3 is to estimate the zero-delay wavenumber spectrum $P_{ZD}(k)$ from the zero-delay covariance matrix R. There are many methods in the literature [10] for such multidimensional spectral estimation problems. For example, the conventional Bartlett method [11] gives the estimate

$$P_{ZD}(k) = (1/N^2)\, E^H R E \tag{4.8}$$

where E is the column vector

$$E = col\,[e^{-jk \cdot x_1}, \cdots, e^{-jk \cdot x_N}], \tag{4.9}$$

N is the number of sensors, x_i is the location of sensor i, and E^H is the complex conjugate transpose of E. Another spectral estimation technique, known as the Maximum Likelihood Method (MLM) [11], belongs to the class of so-called high-resolution techniques. The MLM estimate of $P_{ZD}(k)$ is given by

$$P_{ZD}(k) = 1/E^H R^{-1} E \tag{4.10}$$

where E is the same as in (4.5). Both the Bartlett method and MLM method were considered for use in the test bed. The MLM method was selected because it provides more resolution, at least under conditions of high signal-to-noise ratios.

The amount of computation involved in the sensor array processing method outlined above is approximately the same as that required for estimating the frequency-wavenumber spectrum at just one frequency. To see this refer to Figure 4.4 showing the major computational steps for both the wideband algorithm and the frequency wavenumber method. Where the wideband method must calculate a single N-dimensional scalar product to obtain a bandpassed complex analytic signal sample, the frequency-wavenumber method must perform a full N-point Fast Fourier Transform. Where the wideband method calculates a single covariance matrix, the frequency-wavenumber method must calculate a power spectral density matrix for each frequency. The number of calculations per frequency for the power spectral matrix calculation is the same as the total number of calculations for the covariance matrix estimate. Finally, in normal frequency-wavenumber analysis, a wavenumber spectrum is estimated for each frequency while the wideband method requires only one wavenumber analysis. This is the essence of the computational advantage of the wideband method; it detects the energy from all frequencies while requiring

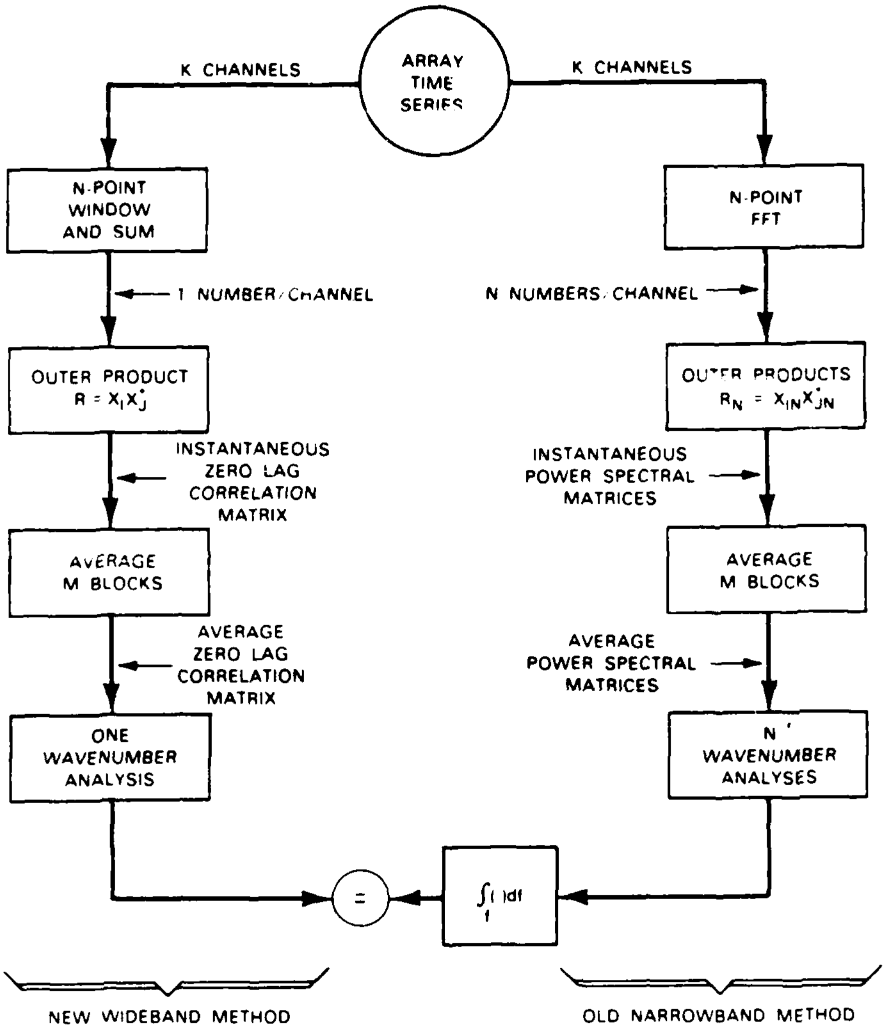


Figure 4.4. Comparison of wideband and frequency-wavenumber methods.

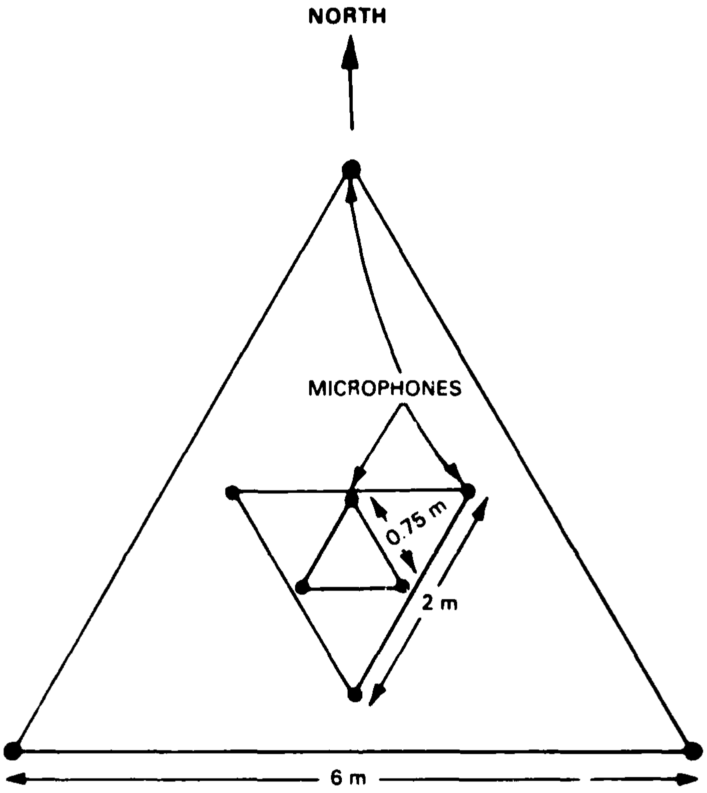
only a single wavenumber analysis. It follows that the computational requirements for estimating the zero-delay wavenumber spectrum is a fraction of the computation required for estimating the frequency-wavenumber spectrum.

The last step in the process shown in Figure 4.3 is to detect the signal and estimate its direction. This is simply done. For the expected range of aircraft elevation angles and frequency bands there is a minimum and maximum value of $|k|$ in wavenumber space. The minimum value is zero, corresponding to aircraft that are overhead or that produce signals with significant power only at very low frequencies. The maximum value is $2\pi F_{\max}/c$, where $F_{\max}$ is the highest frequency in the arriving signal and c is the speed of sound in air. This corresponds to the wavenumber for the highest frequency component of the signal when it is arriving horizontally so that its phase velocity across the array is equal to the speed of sound. The detection algorithm performs a numeric integration along radial arms in wavenumber space as a function of azimuth. The sum approximating the integral goes from zero out to the expected maximum value of $|k|$. This converts the wavenumber spectrum into a detection function that represents power as a function of direction of arrival. The final step is to select the largest peaks in that function and report their azimuths as possible target directions. The size of the peaks is used to assign a strength to the signals. The maximum number of reported peaks is a parameter. The average value of the lowest quartile of the values of the detection function is also reported to provide the tracking algorithms with information that can be used to estimate the accuracy of the direction estimates.

4.4. Experimental Results

The wideband algorithm has become the standard acoustic signal processing algorithm for the test bed. All acoustic tracking results cited in this report, with the exception of results based upon simulated data, were obtained using acoustic measurements produced by the wideband algorithm. We include here two examples that illustrate the signal processor output in more detail than it is illustrated elsewhere in this report.

One example illustrates the output of the signal processor when there are two aircraft present. The other is a comparison of the performance of the wideband algorithm with the MLM narrowband algorithm that was used earlier in the project. The input data for the algorithm was



104059-8

Figure 4.5. Tri-delta microphone array configuration.

acoustic data from a 9-microphone array, sampled at a 2048-Hz rate. The array configuration is illustrated in Figure 4.5 and is the same as that used throughout most of the DSN work. The dimensions are large enough to provide directional accuracy of a few degrees or less.

The details of the processing steps to obtain the experimental results shown in Figures 4.6 and 4.7 are as follows. The first step was to calculate 32 uniformly spaced samples of the complex analytic representations of each of the sensor signals. The observation interval covered by the 32 samples was 1 s for the multiple target experiment and 2 s for the other experiment. To obtain the 32 samples, each signal was divided into 32 nonoverlapping regions of 64 or 128 samples each, depending upon the observation interval. For each block, the middle sample of the complex analytic representation was computed. Next, the zero-delay covariances were computed and averaged over all 32 samples. MLM estimation was then used to obtain the zero-delay wavenumber spectrum from the zero-delay covariance estimates. The wavenumber spectrum was computed at 60 equally spaced points along each of 120 radial wavenumber directions selected uniformly over the entire 360° range of bearings. The maximum wavenumber radius for these computations was selected to correspond to a 128-Hz single-frequency acoustic planewave with a direction vector parallel to the array plane. Source bearings were then determined by summing the integrated power along the corresponding wavenumber radius for each direction and picking peaks in the resulting power versus bearing functions.

Figure 4.6 shows the integrated power as a function of bearing for three different 2 s intervals during a time period when there were two aircraft passing the array. One aircraft maintained a constant bearing to the east of the array, while the second moved from the southeast to the northeast. The figure shows the output 5 s before the aircraft azimuths cross, at the time they are crossing and 5 s after they have crossed. Strictly speaking, this description applies to acoustic azimuths. The acoustic azimuths lag the true aircraft azimuths because of acoustic propagation delays. The distinction is not important here, but it is important for tracking and is fully covered in Section 6.

Figure 4.7 shows the comparison of the wideband algorithm and our previous algorithms during an 80-s period when a helicopter was flying past an acoustic array. Only the azimuth of the largest detection peak is shown as a function of time. The wideband outputs are more consistently near the correct azimuth.

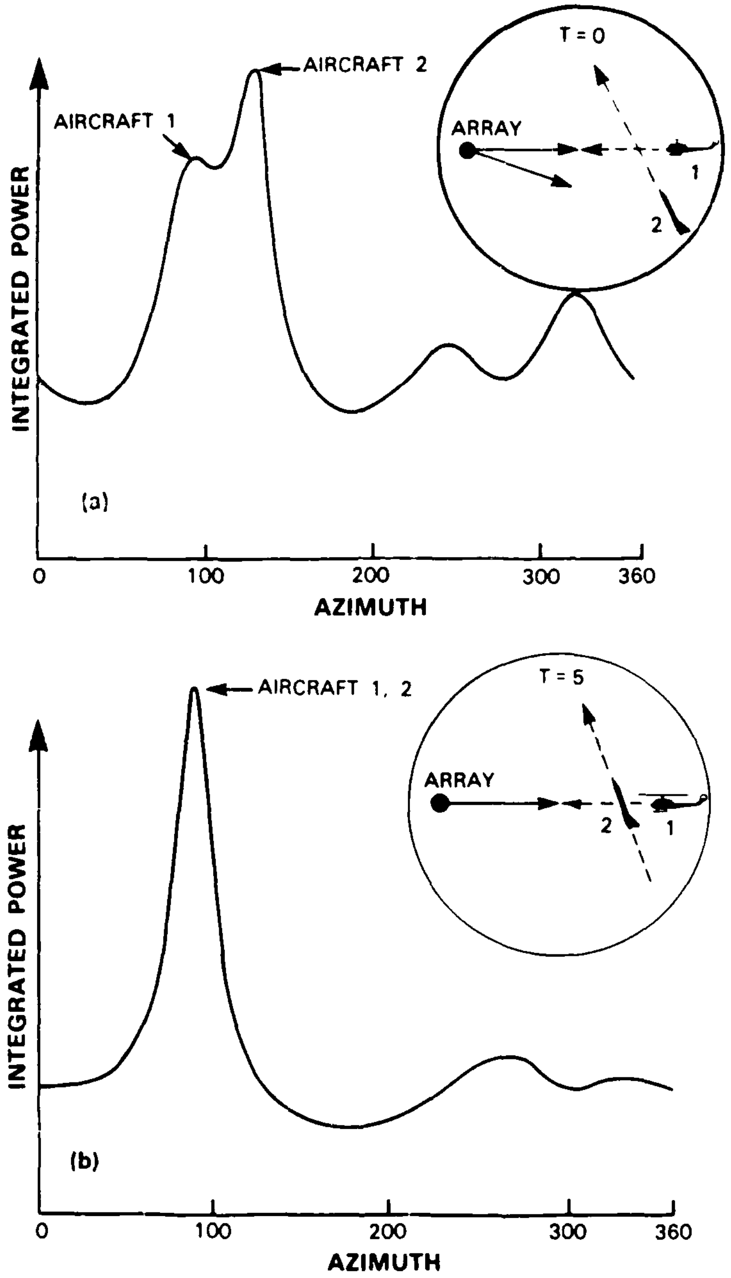


Figure 4.6. Acoustic power as a funciton of bearing for two targets with aircraft located (a) to the east and southeast of the array, (b) east of the array, and (c) to the east and northeast of the array.

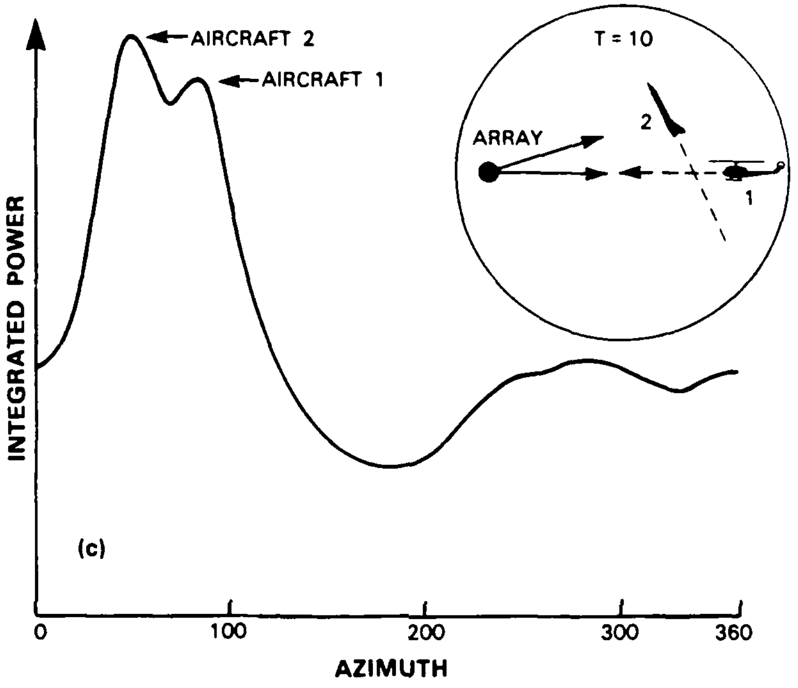
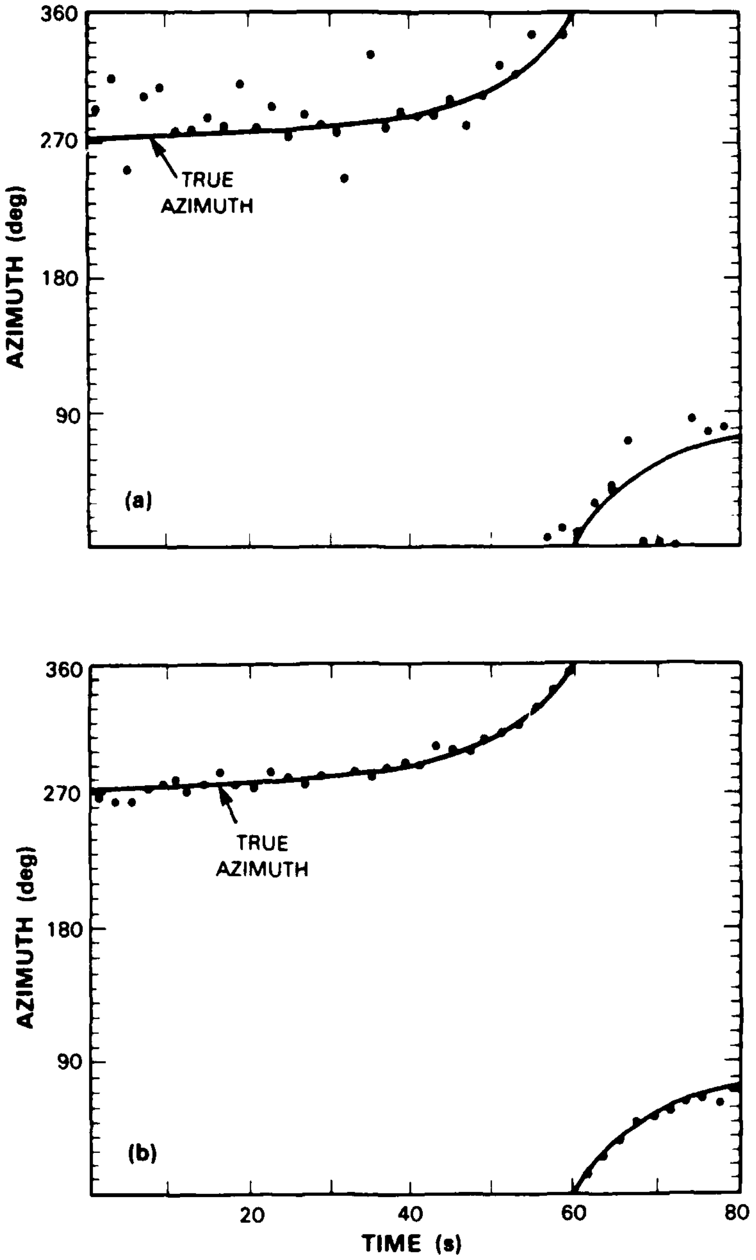


Figure 4.6. Continued.

104069-10



104059-11

Figure 4.7. Azimuth estimates as a function of time. (a) From initial frequency wavenumber algorithm; (b) from wideband algorithm. Solid lines are true azimuth vs time.

5. TV SENSOR ALGORITHMS

5.1. Introduction

Unlike acoustic arrays, TV cameras have a limited Field-Of-View (FOV) and are hence more useful for improving tracks than for finding new targets. The TV subsystems were therefore designed to accept cueing information from the tracker and to provide the tracker with azimuth measurements for selected targets. The approach used to integrate the TVs into the tracking system to complement the acoustics applies equally well to other imaging sensors such as passive IR sensors.

Figure 5.1 shows the relationship between the tracker and the TV subsystem. The TV subsystem obtains position tracks from the tracking system like any other system user. It then selects a specific target, controls the camera to point at the target, collects and processes image data, and provides the resulting azimuth measurements, if any, to the tracker. The TV subsystem has a closed loop relationship with the tracking system. In general the measurements from the TV subsystem can substantially improve target tracks because the TV measurements have no inherent propagation delays and are more accurate than those of the acoustic arrays.

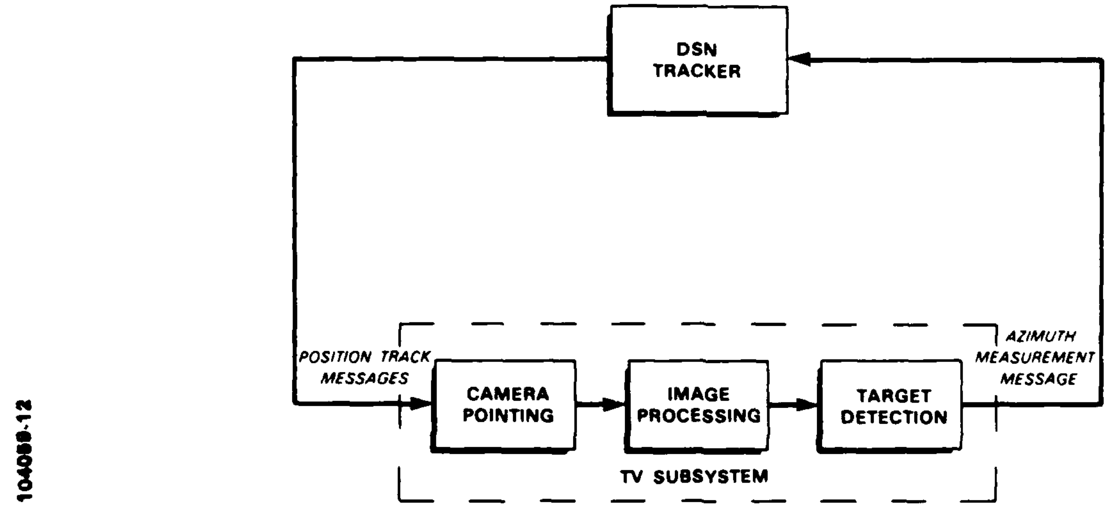


Figure 5.1. TV subsystem elements and tracker interactions.

This Section describes the TV subsystem algorithms developed for use in the test bed. These algorithms provide reliable detections of experimental targets while producing very few false alarms. The hardware is described elsewhere (Section 3).

5.2. Target Selection and Camera Control

The inputs to the camera pointing algorithm are tracking messages from the DSN tracker. Each message contains tracks for up to 5 targets, a limit set by the number of tracks that can be contained within a standard tracking message. The information for each target includes time, position, velocity, and a target identifier (ID). Based on this information, the algorithm performs two functions: it selects a single target and generates camera control commands (azimuth, elevation, and zoom) to bring the target into the FOV of the camera.

The overall target selection objective is to select a target that can be quickly acquired and for which TV azimuth measurements can be made for a specified minimum time duration. The selection process is summarized in Figure 5.2. First, for each target, a list of feasible camera-pointing "policies" (types of camera motions that might bring the target into the field-of-view) is calculated. Next, four features are calculated for each feasible policy and target combination.

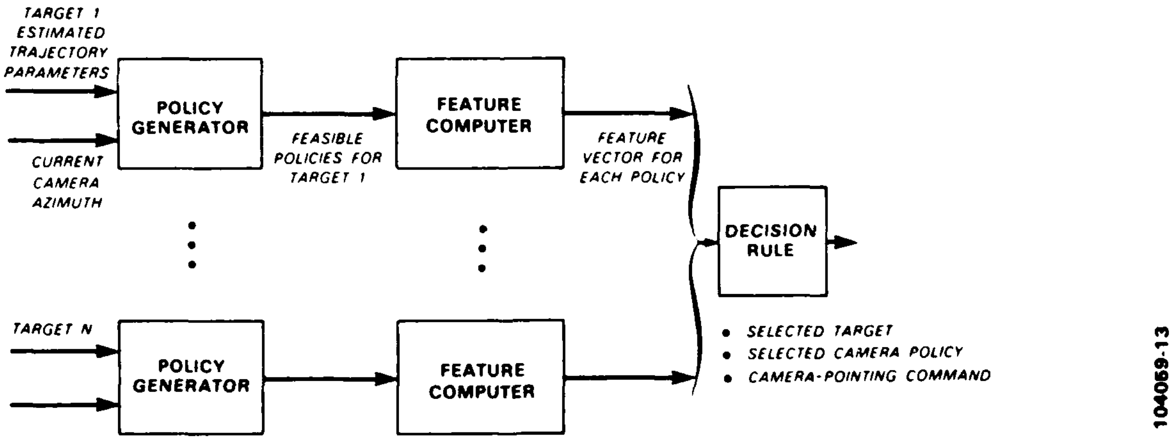


Figure 5.2. Complete camera pointing algorithm.

The features are: (1) Time-to-Catch (how long will it take the camera to slew to the target), (2) In-Track-Time (how long can the camera follow the target), (3) Acquisition-Range (camera-to-target range at acquisition), and (4) Acquisition-Azimuth (target azimuth at acquisition). Finally, a decision rule is applied to the feature vectors to select a target, a camera policy, and an azimuth-slew command that will implement the policy. The rule that has been used thus far selects the policy-target pair that minimizes the Time-to-Catch while achieving an In-Track-Time greater than a specified value (e.g., 10 s).

The list of feasible policies for each target typically includes only 2 to 5 policies. It is obtained by pruning a larger list of 34 possible policies that exhaustively account for three different target/camera situations. These situations are: (1) The target may fly over the camera, thereby moving above the camera field-of-view; (2) The target may fly through a region where the camera cannot slew fast enough to keep up (an "untrackable region"); and (3) a camera dead-zone, resulting from the mechanical limits of the camera mount, may prevent the camera from slewing to a desired azimuth in one direction. Two of these situations are illustrated in Figure 5.3 where the camera is pointing west and the target is traveling south on a track that will bring it to within d = 1 km from the camera. The camera cannot slew through its dead-zone (which begins/ends at $105^o/115^o$ in azimuth). In addition it cannot slew fast enough to follow the target while the target is in the region labeled "untrackable." Clockwise and counterclockwise camera motions are treated separately in the possible policy list. Simple rules, based upon the camera initial position and the aircraft track, are used to prune the list of possible policies down to the list of feasible policies.

Two feasible policies and their associated features are also shown in Figure 5.3. For Policy 1 the camera slews clockwise for 11 s and acquires the target relatively quickly but is able to maintain track for only 6 s, at which time the target enters the untrackable region. For Policy 2 the camera slews counterclockwise (CCW) for 21 s and acquires the target as the target emerges from the camera dead-zone. Thereafter it can follow the target forever. If the required minimum In-Track-Time is less than 6 s, then Policy 1 will be selected because it is the option with the smaller Time-To-Catch. If the required minimum In-Track-Time is greater than 6 s then Policy 2 will be selected because it is the only one meeting the In-Track-Time requirement.

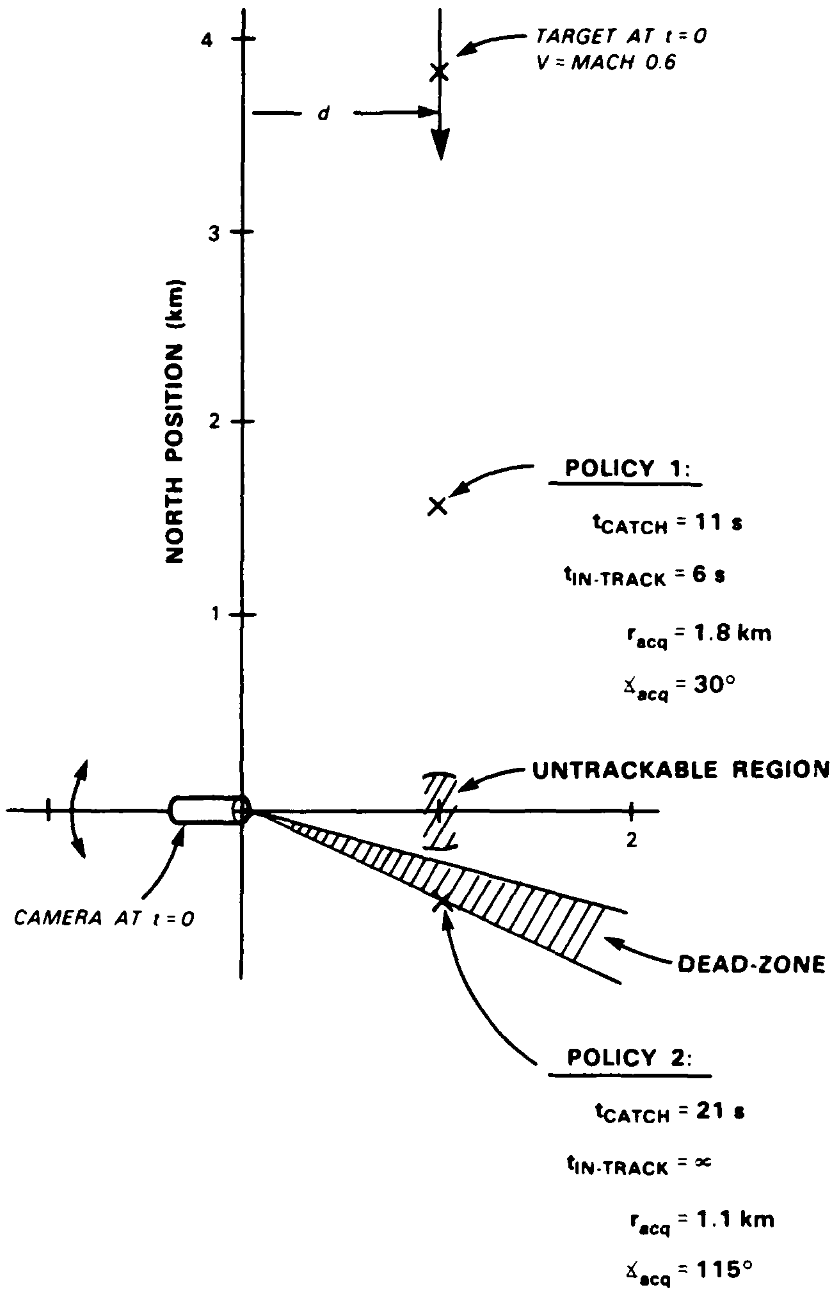


Figure 5.3. Examples of camera pointing policies.

The feasible policies corresponding to Figure 5.3 depend upon the distance d. For small distances the untrackable region becomes larger and overlaps the dead-zone, resulting in different feasible camera policies (e.g., slew CCW and wait at the end of the untrackable region). If the distance d is almost zero, a "target-over-camera" condition would be declared, resulting in still other feasible policies (e.g., acquire inbound or outbound).

The computations required by the target selection procedure described above take approximately 0.8 s per target to execute in the Standard Nodal Computer. An abbreviated version of the algorithm, requiring only 0.2 s to execute, was also developed to reduce the average processing load. The abbreviated version forces the TV subsystem to adhere to a target for several tracker cueing cycles if possible.

The logic for selecting between the complete and abbreviated algorithms is shown in Figure 5.4. The first time a position message arrives (i.e., when the TV node is first enabled), the full target selection algorithm is executed, resulting in the selection of a target and camera-pointing policy, and in the issuing of a camera-pointing command. In addition, the identifier of the selected target and its associated In-Track-Time feature are saved. Thereafter, as new position messages arrive, the full target selection algorithm is executed only if: (a) the track corresponding to the

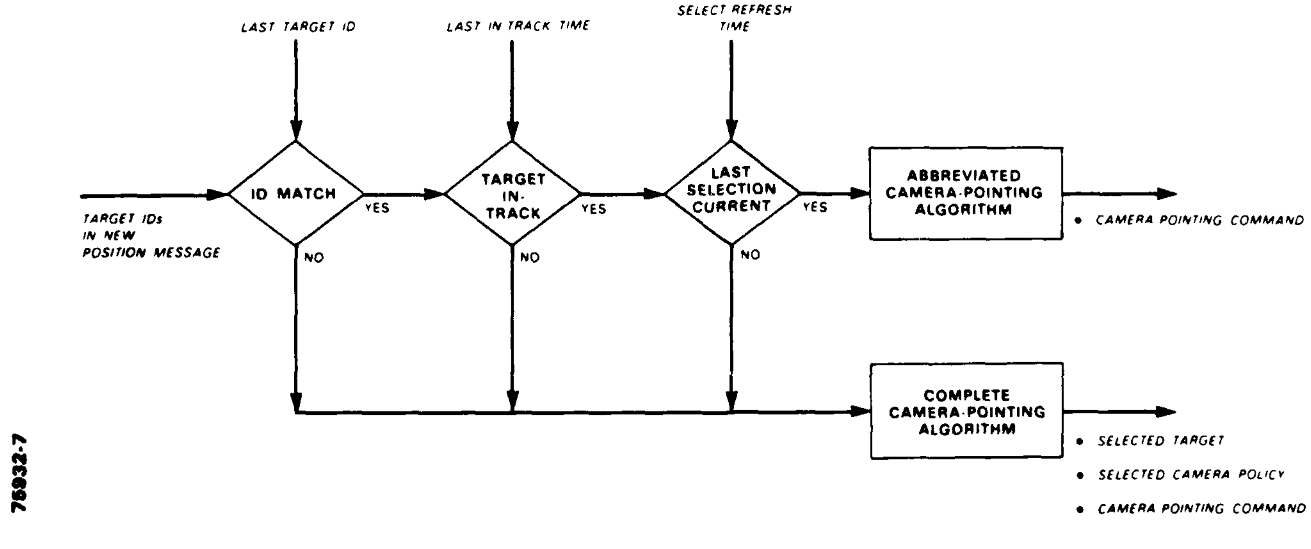


Figure 5.4. Selection of pointing algorithms.

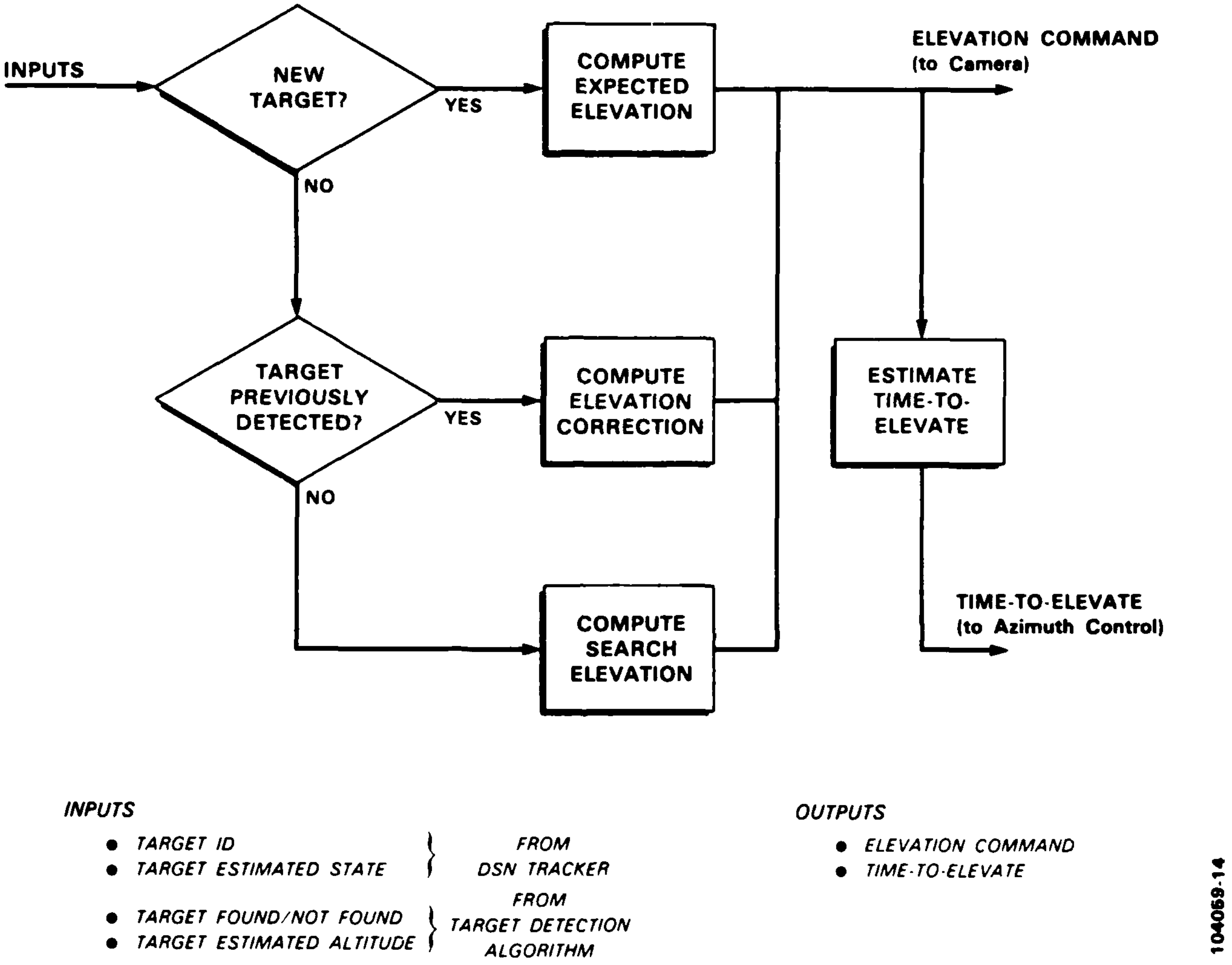


Figure 5.5. Elevation control algorithm.

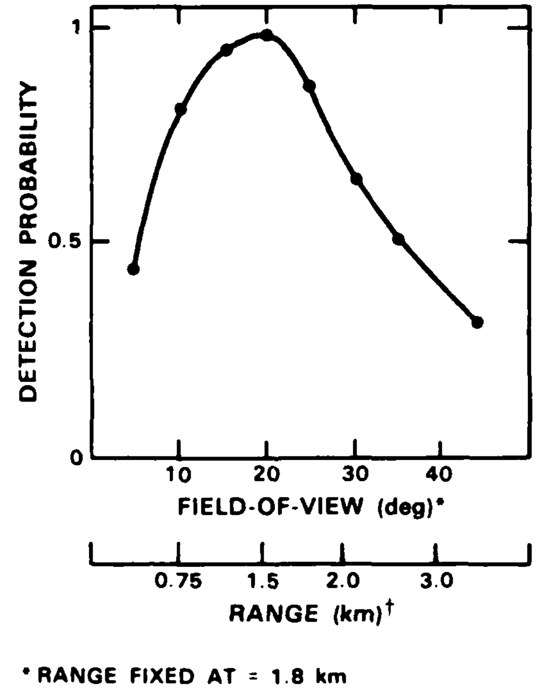
previously selected target has been dropped by the cueing DSN node; (b) the In-Track-Time period of the previously selected target has expired; or (c) the previous target selection is declared to be obsolete on the basis of the "select refresh time" parameter (typically 20 to 60 s). The select refresh time parameter forces the system to occasionally reconsider its selections if other criteria do not require it to do so. In all other situations the abbreviated selection algorithm is executed. The abbreviated algorithm selects the previously selected target, bypassing most of the policy selection functions, and generates camera-pointing commands using the most recent target state estimate provided by the tracker.

Initial live testing of the target selection algorithm showed that it occasionally elected to search for acoustically noisy construction equipment that had been located by the acoustic tracking system but was masked from the camera position. Two modifications were made to avoid searching for such nuisance targets. First, if a target is not detected after a search in elevation (described in Section 5.2.2), the track identifier is entered on a false-track list. All future cues with this track identifier are disregarded. Second, exclusion regions are defined specifying areas known to contain ground construction equipment. Targets with estimated positions within the exclusion regions are bypassed. These modifications are admittedly ad hoc solutions to specific problems, but they have been very effective.

Analysis and initial experiments using the TV subsystem also revealed that targets were often lost or not acquired because the camera zoom and elevation were constant throughout an experiment. Camera elevation and zoom control algorithms were then developed that significantly increased the target acquisition probability and the number of measurements obtained after acquisition.

The functional elements of the elevation control algorithm are illustrated in Figure 5.5. The algorithm first checks to determine if a target is new (i.e., has not been acquired before). If it is new, the camera elevation angle is selected using a default value for target altitude. If the target is not new, two options exist. If it is not new and has been detected in at least one of the last three attempts, a 2° camera elevation change is made, if necessary, based on the latest altitude estimate provided by the target detection algorithm (described below). The objective is to keep the target near the center of the vertical field-of-view. If the target was not detected in any of the last three attempts, an elevation search is initiated. The lower limit on the elevation search is based on an "elevation map"--a table of elevation angles as a function of azimuth specifying the elevation of ground obstacles such as trees and buildings. If no detection is made during the elevation search, the track identifier is entered into the false-track list.

The probability of detecting a target with a TV camera is a function of camera zoom, a parameter that determines the field-of-view (FOV). The detection probability is equal to the probability that the target is in the FOV multiplied by probability of detection given that the target is in the FOV. The overall detection probability is plotted in Figure 5.6 as a function of FOV for a fixed target range and as a function of target range for a fixed FOV. First consider the case of a fixed target range. For a small FOV (large magnification), the probability is small due to random errors in the position cue and in the positioning of the camera. For a large FOV (small magnification), the probability is small because, even if the target is in the FOV, the size of the target image on the screen is too small to be reliably detected. Similar logic explains the shape of the probability curve as a function of range for a fixed FOV as shown in Figure 5.6.



* RANGE FIXED AT = 1.8 km
† FIELD-OF-VIEW FIXED AT = 25°

104069-15

Figure 5.6. Target detection probability of TV subsystem as a function of field-of-view and range for a 10-m target and 200-m RMS target position error.

The zoom control algorithm adjusts the FOV to maintain a high detection probability independent of target range and aspect. To achieve this objective, the FOV is maximized while keeping the size of the target image on the TV monitor between 1 and 1.5 cm. This size provides a near unity detection probability when the target is in the camera FOV. The calculation of image size is based upon the assumed minimum physical dimensions of targets. If the estimated size is in the 1- to 1.5-cm range no action is taken; otherwise a table of options is revised with a target size and time-to-zoom calculated for each option. Selection of an option depends on whether enough time is available to complete the zoom change during the camera azimuth and elevation slew time. If a zoom command is issued the option table is updated to correctly represent the state of the zoom control options after the command is executed. The use of zoom to maintain an approximately constant target image size made it possible to develop simple and effective signal processing algorithms for target detection.

The test-bed target selection and camera control algorithms ignore the uncertainty estimates provided by the tracking system. Use of the uncertainty estimates should be investigated. For example the track uncertainty will affect the probability that the target is within the camera FOV. An estimate of this probability might be an additional feature to consider for target selection. The value of doing this will depend upon the quality of the track uncertainty estimates.

5.3. Image Processing

Once the camera is pointing at a target, two video frames spaced 1/30 s are taken and stored in two frame buffers. Image processing consists primarily of forming the difference image of these two frames to produce a frame that emphasizes the outlines of moving objects within the field-of-view. The difference frame replaces one of the input frames and is kept for processing by the target detection algorithm described in Section 5.4. The positive image in the other frame buffer is also kept and used for two purposes: to aid in the discrimination function of the target detection algorithm; and to preserve the target image for possible transmission to remotely located network users.

5.4. Target Detection

The target-detection algorithm in the TV subsystem determines whether a target is present in the field-of-view and, if present, determines the azimuth and elevation of the target. The

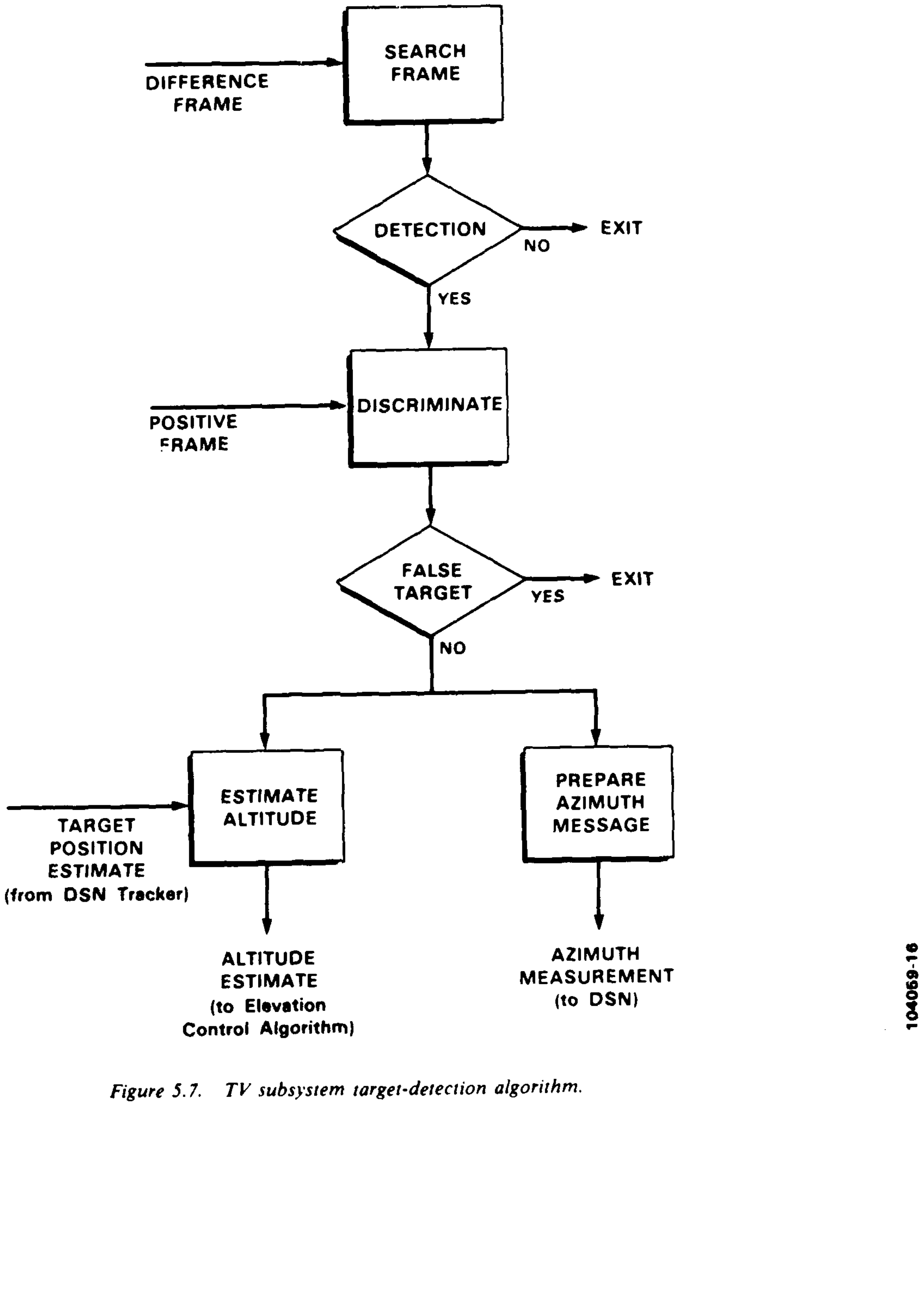


Figure 5.7. TV subsystem target-detection algorithm.

azimuth measurement is returned to the DSN tracker and the elevation measurement is used to control elevation.

The logic followed by the target-detection algorithm is summarized in Figure 5.7. First, the entire difference frame (obtained from the image processing algorithm) is scanned and the image intensity is averaged over blocks of 4 x 6 pixels. The averaging operation is performed by sampling 10 out of the 24 pixels in each block (to reduce processing time) in a pattern of two 5-pixel columns separated by an unsampled column. Block size selection is based on the dimensions of the projection of the target on the frame which is regulated by the zoom control algorithm to be approximately 5 pixels high and 16 pixels long (0.25 cm by 1 cm).

Next, the block average with the largest magnitude is compared with a threshold. If the average exceeds the threshold, indicating the presence of a moving object, the average brightness of the corresponding block in the positive frame is computed. Excessive average brightness indicates that the detection was caused by a moving bright cloud or by a false bright spot such as caused by diffraction within the camera lens system.

Finally, if a detection passes the brightness threshold and if the measured elevation is within the range specified by the elevation map (Section 5.2), the algorithm computes the target azimuth and altitude. The altitude is returned to the elevation control algorithm where it is associated with the target ID provided by the DSN tracker and used to predict target elevation for the next measurement cycle. The azimuth measurement and an estimate of its accuracy are transmitted to the DSN tracker where they are used to update the target track.

The accuracy assigned to a TV azimuth measurement is a function of the camera zoom setting. It is set equal to 1/33 of the angular FOV of the camera. This assumes that the target size is about 1 cm in extent within a 33-cm screen and that the precision with which the target can be located within the image is equal to about 1 cm. These assumptions are consistent with the fact that the camera zoom is manipulated to keep the target size in the range from 1 to 1.5 cm within the same 33-cm screen. The assigned accuracy can be interpreted as a rough estimate for the standard deviation of the measurement error.

The test-bed tracker, which interacts with the TV subsystem, does not provide the TV subsystem with target altitude information and does not accept elevation measurements. Clearly, this would change if a three-dimensional tracker, including target altitude, was developed and used.

5.5. Experimental Performance

The TV algorithms were tested, refined, and demonstrated during a sequence of experiments with live targets conducted during the summer and fall of 1986. Typical is the pattern shown in Figure 5.8 obtained during a real-time experiment on 15 September 1986. The figure shows the estimated trajectory of a UH-1 helicopter with annotations that give the camera elevation and field-of-view during the entire period that cues were provided to the TV subsystem. The track is roughly east-to-west parallel to one of the Hanscom Air Force Base runways.

When the initial cue was received, the camera was pointing east with a 3° elevation angle and a 7° field-of-view. The pointing algorithm then predicted a target/camera intercept and, while the camera slewed in azimuth, elevation and zoom settings were changed. Elevation was changed to 11° reflecting the expected target altitude and position at intercept. Field-of-view was increased to 17° because the target was close to the camera and very little magnification was needed to obtain a 1-cm target image. When camera settings were completed, two frames were stored and processed, a target found, and the azimuth measurement sent to the DSN tracker.

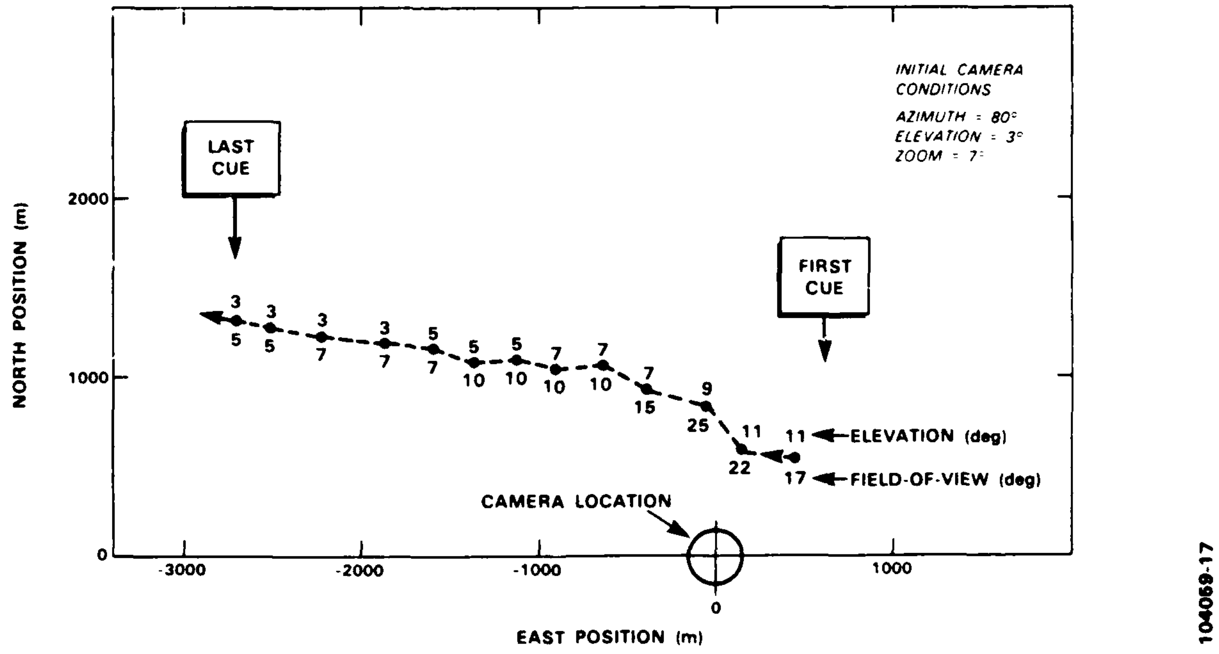


Figure 5.8. Camera control during live experiment.

Elevation and zoom were adjusted for subsequent cues as shown in Figure 5.8. Elevation decreased as the target moved away from the camera because of the effect of geometry and changes in the target altitude. Field-of-view was increased (magnification decreased) while the target was close to the camera and decreased (magnification increased) as the target moved farther from the camera. Thirteen cues were received and twelve azimuth measurements returned to the tracker. One detection was missed because the difference-frame threshold criteria was not satisfied. The last cue was received and the last azimuth measurement returned when the helicopter was at a range of approximately 3 km.

6. DISTRIBUTED TRACKING ALGORITHMS

6.1. Introduction

A primary research goal was to develop distributed tracking algorithms that were effective but simple enough to be demonstrated in the DSN test bed. Two different approaches were investigated. The preferred one is described in detail in Section 6.2 [12-14]. The algorithms described there have a firm theoretical basis, easily accommodate different sensor types as well as distributed sensors, and provide real-time tracking of subsonic targets in spite of acoustic signal delays. They represent a realistic basis for practical distributed tracking systems. The restriction to subsonic targets comes only from using acoustic sensors and is not a fundamental limitation of the approach. For perspective, the other approach is discussed briefly in Section 6.3. It was not developed beyond the level of two-node track initiation when it became clear that it had serious drawbacks as discussed in Section 6.3. While it was capable of distributed tracking, it substantially complicated the tracking process and limited the cooperation between nodes.

6.2. Bayesian Model-Based Distributed Tracking Algorithm

6.2.1. Overview

Figure 6.1 shows the overall structure of the model-based distributed tracking algorithm that has been developed for the DSN test bed. The elements shown in the figure are duplicated throughout the network. Each node operates independently and asynchronously, driven by data from local sensors connected to the tracker and by data received from other sensors via broadcast communications. The tracker may be connected to a local acoustic subsystem and possibly a local TV subsystem. Each tracking node in the test-bed system contains a complete tracking algorithm although only portions of it may be exercised, depending upon the sensors attached to the node.

Each node contains a "position track" file. Estimates of aircraft positions and velocities are stored in this file and the term "position track" is intended to encompass both the position and velocity estimates. Tracks are represented by target specific quantities independent of how the tracks were obtained, with the one exception that track names include the identities of the two nodes that first initiated the track. Altitudes are not estimated; all aircraft are assumed to be flying

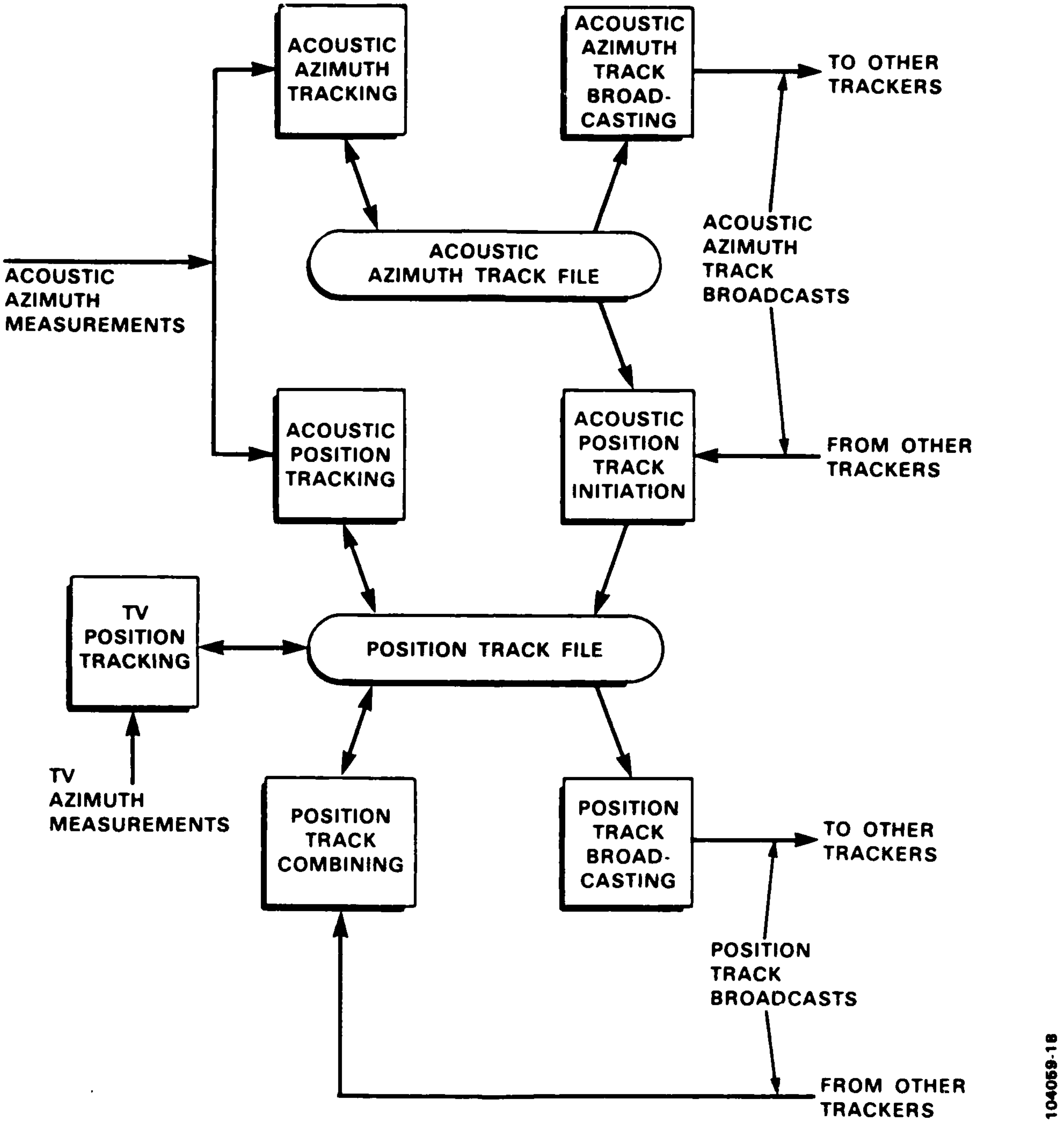

ACOUSTIC AZIMUTH TRACKING
ACOUSTIC AZIMUTH TRACK BROAD-CASTING
TO OTHER TRACKERS
ACOUSTIC AZIMUTH TRACK FILE
ACOUSTIC AZIMUTH MEASUREMENTS
ACOUSTIC AZIMUTH TRACK BROADCASTS
ACOUSTIC POSITION TRACKING
ACOUSTIC POSITION TRACK INITIATION
FROM OTHER TRACKERS
TV POSITION TRACKING
POSITION TRACK FILE
TV AZIMUTH MEASUREMENTS
POSITION TRACK COMBINING
POSITION TRACK BROAD-CASTING
TO OTHER TRACKERS
POSITION TRACK BROADCASTS
FROM OTHER TRACKERS
104059-18

Figure 6.1. Organization of the distributed tracking algorithm.

at low altitudes. Track confidences are maintained in the form of position/velocity error covariances. Similar tracking algorithms could be developed for three spatial dimensions but the tracking algorithms developed for the test bed are for two spatial dimensions.

Position tracks are updated using either local sensor measurements or by integrating tracks received from other nodes. If the update uses local sensor measurements then an extended Kalman filter is employed. Slightly different filters are used for acoustic and for TV azimuth measurements. In both cases, the filters compensate for the passive (i.e., bearing-only) nature of the measurements. In the case of acoustic azimuth measurements, the filter also compensates for acoustic propagation delays. Measurement confidences are used to update track confidences in both cases. It is the use of Kalman filters, and certain assumptions that are needed for acoustic track initiation, that make the system "model-based."

Track data are shared between nodes by broadcasting position tracks. Such a broadcast may occur immediately after a position track is updated using local sensor data or it may be deferred to broadcast only the cumulative effect of many updates. The position track combining algorithm in a node receiving such a broadcast merges the received information with that in its own track file. It is through this mechanism that the local track in a node incorporates sensor and track information from other nodes. Confidences are merged along with estimates. This algorithm is the essential glue that holds the network together and causes it to function as a single distributed tracking system.

The test-bed system is a distributed system for track initiation as well as for track maintenance. It uses acoustic measurements for track initiation, with measurements from two geographically separated sensor arrays needed to start a track. Moreover, because of target motion and acoustic delays, at least two measurements are required from each site. The measurements could be azimuth measurements or could be estimates of azimuth and azimuth rate. The second option was selected for the test-bed system.

The test-bed track initiation algorithm requires azimuth and azimuth rate estimates from two nodes. These are obtained by forming azimuth tracks, each of which is an evolving estimate of the directional time history of a passing sound source. Both the apparent acoustic direction and the rate of change of that direction are estimated as part of the tracking process. Confidence estimates for these quantities are maintained in the form of covariances.

The upper portion of Figure 6.1 shows that position track initiation is performed usin;: only acoustic sensors. TV data could also be used, but would be of limited value. Microphone arrays can scan a full 360° in a short time while TV cameras have narrow instantaneous fields-of-view. A TV camera would require a much longer time to complete a full scan. Although TV could be used for surveillance it is better suited to help track known targets and was restricted to that role.

The details of the tracking algorithm are presented and discussed in the following subsections. The position track representation is described in Section 6.2.2. Section 6.2.3 describes how position tracks are updated using local TV azimuth measurements, while Section 6.2.4 describes how position tracks are updated using local acoustic azimuth measurements, with emphasis upon the effects of propagation delay. Section 6.2.5 describes the combining of local and foreign position tracks with particular attention to the problems of combining position tracks based on shared information. With Section 6.2.5 to provide motivation, Section 6.2.6 describes alternate policies for position track broadcasting and the consequences thereof. Section 6.2.7 describes the creation and maintenance of acoustic azimuth tracks, while Section 6.2.8 describes the combination of local and foreign azimuth tracks to initiate position tracks. With Section 6.2.8 to provide motivation, Section 6.2.9 describes alternate policies for azimuth track broadcasting and the consequences thereof. Section 6.2.10 describes the features of the test-bed tracking algorithm which cope with multiple aircraft. Section 6.2.11 cites some ways in which the tracking algorithm might be improved to better handle multiple targets. The tracking of multiple and maneuvering targets, especially with acoustic sensors, is a topic requiring further research to develop more optimal algorithms and to quantitatively evaluate performance.

6.2.2. Basic Notation ar.: the Target Dynamic Model

Tracks in the position track file specify "current" ground positions and velocities. A "current" estimate is the most up-to-date estimate that is possible in a real-time system. It is the estimate of greatest interest to a system user. For sensors with no significant inherent delays, such as TV sensors, it is obvious that there are no fundamental difficulties involved in providing users with current estimates. But an acoustic azimuth measurement is affected by propagation delay. During the propagation time the aircraft bearing can change significantly. In the worst case, the change approaches 60°. In the early stages of the DSN project it was unclear if it was practical or desirable to perform all tracking in terms of current positions. (Section 6.3 covers the issues in more detail.) However, subsequent research has shown that it is both possible and desirable to do so.

Current position tracking has resulted in an extensible distributed tracking system architecture that easily accommodates different sensor types. It required that statistical dynamic models of aircraft motion and measurements be employed to use the information from acoustic sensors. Statistical models made it possible to exploit well-developed methods such as Bayesian estimation and Kalman filters. The resulting algorithm estimates aircraft positions and velocities using all available information, regardless of whether it is provided directly from a local sensor or indirectly via a position track broadcast by a foreign node.

The simplest possible target motion model has been used. It assumes that the target travels at a constant speed with a constant heading. Using this model a current position and velocity are easily extrapolated to any other time. Let the north and east components of position at time t be denoted by $p_N(t)$ and $p_E(t)$, respectively. Let the corresponding velocities be $v_N(t)$ and $v_E(t)$ and group the positions and velocities into a 4-dimensional state vector $X_P(t)$:

$$X_P(t) = \begin{bmatrix} p_N(t) \\ p_E(t) \\ v_N(t) \\ v_E(t) \end{bmatrix} \tag{6.1}$$

For constant speed and heading, the state at time τ, $X_P(\tau)$, is obtained from the state at time t by:

$$X_P(\tau) = A_P(\tau - t)\, X_P(t) \tag{6.2}$$

where

$$A_P(d) = \begin{bmatrix} 1 & 0 & d & 0 \\ 0 & 1 & 0 & d \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

Multiplication by this matrix represents simple extrapolation.

Let $\hat{X}_P(\tau \mid t)$ be an estimate of the target state at time τ based upon all information that is available up to time t. This estimated state can be extrapolated in the same way as the true state vector can be extrapolated. However, the estimated state vector also has an associated covariance matrix. Let

$$E\left[\, (X_P(\tau) - \hat{X}_P(\tau \mid t)\,)\,(X_P(\tau) - \hat{X}_P(\tau \mid t)\,)^T \, \right] = \Sigma_P(\tau \mid t) \tag{6.4}$$

be that covariance matrix, where $E\left[\, \cdots \right]$ denotes statistical expectation. The equation to extrapolate the covariance matrix is

$$\Sigma_P(\tau \mid t) = A_P(\tau - t)\, \Sigma_P(t \mid t) A_P(\tau - t)^T + \Xi_P(\tau - t) \tag{6.5}$$

where

$$\Xi_P(d) = \alpha_P \; B_P(d)\, B_P(d)^T, \tag{6.6}$$

$$B_P(d) = \begin{bmatrix} \frac{d^2}{2} & 0 \\ 0 & \frac{d^2}{2} \\ d & 0 \\ 0 & d \end{bmatrix} \tag{6.7}$$

and α_P is a constant. The first term is the extrapolation of the target state for the case when the target exactly follows as constant velocity and heading track. The second term accounts for possible random accelerations of the target during the extrapolation time. The nominal acceleration variance α_P is a constant that controls the degradation of confidence over time when there are no additional measurements.

The state estimate and state covariance pair $\hat{X}_P(\tau \mid t)$ and $\Sigma_P(\tau \mid t)$ can also be regarded as defining a Gaussian probability density function for the true state $X_P(\tau)$. With this viewpoint the nominal acceleration variance α_P can be regarded as the variance of a random acceleration applied in a random direction.

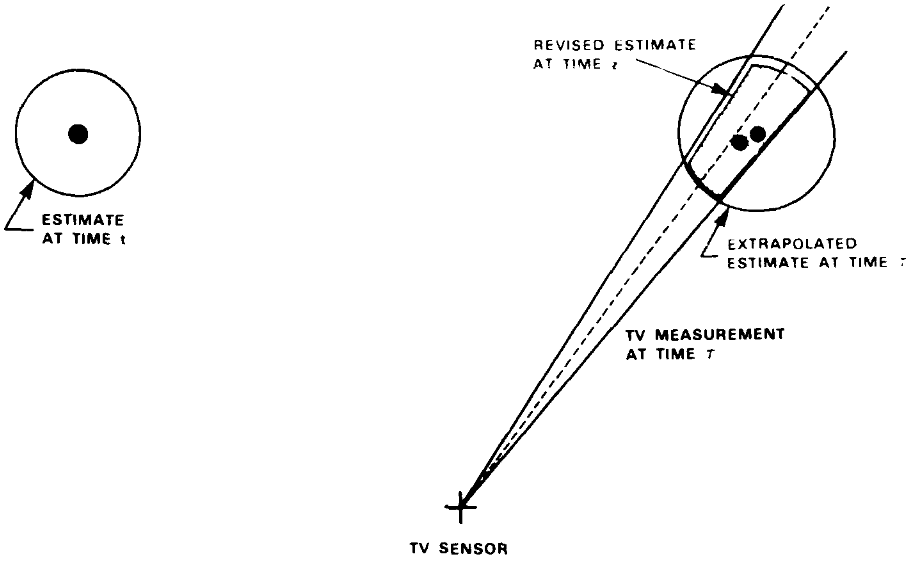
This model and the estimation procedures can be extended to handle three-dimensional motion but this is an area requiring further research. Topics needing attention include an analysis of the sensitivity of the two-dimensional tracker to the aircraft altitude, an analysis of the potential value of acoustic elevation angle measurements, and the development of a three-dimensional tracker.

6.2.3. Position Track Updating with TV Azimuth Measurements

Although a TV azimuth measurement provides no information about the range to an aircraft it is still possible to update a position track with such an angle measurement. A simplified geometric interpretation of the procedure shows how this works. The actual updating algorithm is more subtle than indicated in the following description, but analogous in how it works and makes use of the measurement and measurement confidences.

Figure 6.2 illustrates position updating with TV azimuth measurements. Suppose that an aircraft is flying along a straight path from left to right across the figure. The position estimate for a past time t is shown on the left side of the figure, along with a circle representing the confidence region for the estimate. The location of the TV sensor is indicated by a plus symbol. The measurement for a later time τ is represented by the dashed line radiating from the sensor. The confidence region for the measurement is the pie-shaped region around the measurement.

The first step in updating a position track is to extrapolate the position track to the measurement time. The extrapolated position is the red dot at the right side of the figure. The size of the confidence circle has increased to represent a decrease in certainty about the aircraft's location due to the extrapolation from t to τ. The decrease in certainty is because the velocity at time t, as well as the position, is known only approximately. At time τ the target position becomes constrained by two confidence regions: one derived from the extrapolation from time t and the other from the TV measurement confidence. A new net confidence region, approximately equal to the intersection of the two contributing confidence regions, is indicated by green shading in the figure. The new position estimate (the green dot) is slightly displaced from the previously extrapolated estimate (the red dot) because of the effect of the TV measurement, which is behind the extrapolated position. The confidence area after updating to time τ is less than it was at time t. Updating has improved the overall accuracy of the position track. However, although the area is less, the error along the line-of-sight of the TV sensor at time τ is larger than it was at time t.



87882-16

Figure 6.2. Updating a target track using a TV measurement: conceptual steps. See text for discussion.

The precise mathematical details of the updating algorithm closely parallel the above description. The inputs to the algorithm consist of a position track at time t, represented by a state estimate and state covariance pair $\hat{X}_P(t \mid t)$ and $\Sigma_P(t \mid t)$, a TV sensor location represented by north and east position coordinates s_N and s_E, and a TV azimuth measurement at time τ, represented by a measurement and measurement variance pair $z_{TVP}(\tau)$ and $\Theta_{TVP}(\tau)$. The measurement variance represents confidence in the measurement just as the state covariance does in the state estimate. They can be regarded as defining a Gaussian probability density function for the true aircraft azimuth at time τ, just as the state estimate and covariance can be regarded as similarly defining a Gaussian probability density function for the true aircraft position and velocity at time t.

The first step in updating is to extrapolate the position track to the measurement time τ, yielding $\hat{X}_P(\tau \mid t)$ and $\Sigma_P(\tau \mid t)$. The updated state estimate and covariance are then computed using the extended Kalman filter algorithm [15]:

$$\hat{X}_P(\tau \mid \tau) = \hat{X}_P(\tau \mid t) + K_{TVP}(\tau)\left[z_{TVP}(\tau) - \hat{z}_{TVP}(\tau \mid t) \right] \tag{6.8}$$

and

$$\Sigma_P(\tau \mid \tau) = \left[I - K_{TVP}(\tau)\, C_{TVP}(\tau) \right] \Sigma_P(\tau \mid t) \tag{6.9}$$

where

$$\hat{z}_{TVP}(\tau \mid t) = \arctan\left[\hat{p}_N(\tau \mid t) - s_N \ , \hat{p}_E(\tau \mid t) - s_E \right] \tag{6.10}$$

$$K_{TVP}(\tau) = \Sigma_P(\tau \mid t)\, C_{TVP}(\tau)^T \left[C_{TVP}(\tau)\, \Sigma_P(\tau \mid t)\, C_{TVP}(\tau)^T + \Theta_{TVP}(\tau) \right]^{-1} \tag{6.11}$$

$$C_{TVP}(\tau) = \begin{bmatrix} -\dfrac{\hat{p}_E(\tau \mid t) - s_E}{\hat{r}(\tau)^2} \\ \dfrac{\hat{p}_N(\tau \mid t) - s_N}{\hat{r}(\tau)^2} \\ 0 \\ 0 \end{bmatrix}^T \tag{6.12}$$

and

$$\hat{r}(\tau) = \left[\left[\hat{p}_N(\tau \mid t) - s_N \right]^2 + \left[\hat{p}_E(\tau \mid t) - s_E \right]^2 \right]^{1/2} \qquad (6.13)$$

The state estimate is corrected by appropriately weighting the difference between the TV azimuth measurement and the extrapolated aircraft azimuth relative to the TV sensor. That weight is the Kalman filter gain $K_{TVP}(\tau)$. The state covariance is then reduced appropriately given the correction and the Gaussian probability density function interpretation of the position track and the measurement. The Kalman filter gain is calculated using the partial derivatives of the extrapolated aircraft azimuth with respect to the extrapolated positions and velocities, $C_{TVP}(\tau)$.

There are some minor but important problems that must be addressed when the extrapolated aircraft range from the TV sensor, $\hat{r}(\tau)$, is small. The problems are typical ones for extended Kalman filters and are solved pragmatically. When the range is small some partial derivatives may become excessively large, resulting in large Kalman gains and tracking instabilities. The solution for this problem is to artificially increase the measurement variance when the extrapolated range is small. Another problem is that when aircraft-to-sensor range is small the extrapolated position may be on the opposite side of the sensor from the actual aircraft position, resulting in a large difference between the azimuth measurement and the predicted azimuth. This can result in a large position correction although the error is actually small. The solution is to skip the update when the difference between the predicted and measured azimuths is unreasonably large.

Figure 6.3 shows the results of an early experiment [16] with the test bed that demonstrated tracking with a single TV camera. The experiment involved real-time tracking of a helicopter. The track was manually initiated with a circular error ellipse at the instant the helicopter reached the center of the TV field-of-view. A line connects the points in the resulting position track in the figure and an error ellipse is shown centered at each track point. The ellipses are the two-dimensional equivalent of a one-standard-deviation error bar. As expected, updating with the TV measurements produced eccentric ellipses with their major axes aligned with the TV sensor's line-of-sight. The major axes expand slowly even as subsequent updates are made. When no more measurements are available, the ellipses expand along both axes.

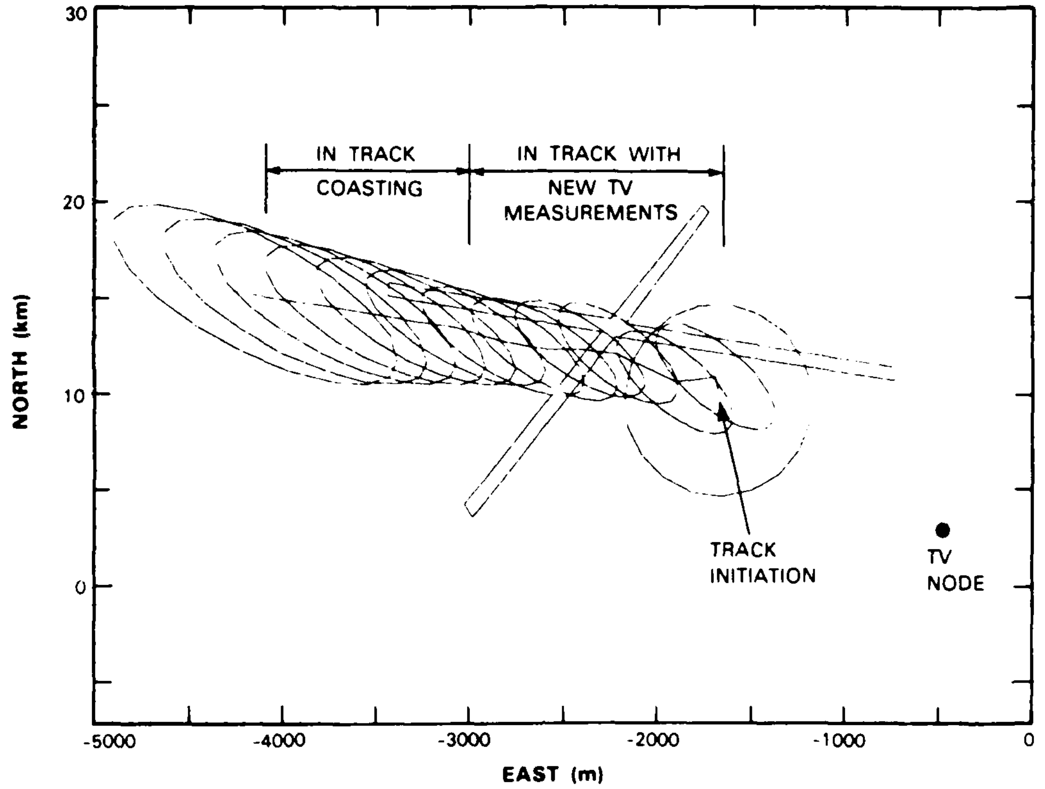


Figure 6.3. Real-time helicopter tracking with a single TV subsystem.

The extended Kalman filter algorithm works well as long as the true state estimation errors are statistically consistent with those expected on the basis of the error covariance matrix. But if there are too many large state estimation errors, the position track can diverge from the correct value. The tracker becomes overconfident in its own estimate. One way to avoid this is to make the acceleration variance α_P large, thereby keeping the covariance matrix large. Although this does stabilize the tracker it has the negative effect that large estimation errors can result. Figures 6.2. and 6.3 illustrate one reason: the updating procedure does not reduce the extrapolated position estimation error along the TV line-of-sight. So the best choice for the acceleration variance parameter is a compromise between avoiding divergence and accurate tracking for acceleration-free aircraft. Section 6.2.11 briefly discusses a more general way to avoid divergence and treat maneuvering targets.

6.2.4. Position Track Updating with Acoustic Azimuth Measurements

Position track updating also can be done using acoustic azimuth measurements. The algorithm is more complicated than for TV azimuth measurements because of acoustic delays. Figure 6.4 illustrates the basic ideas as did Figure 6.2 for the case of TV measurements.

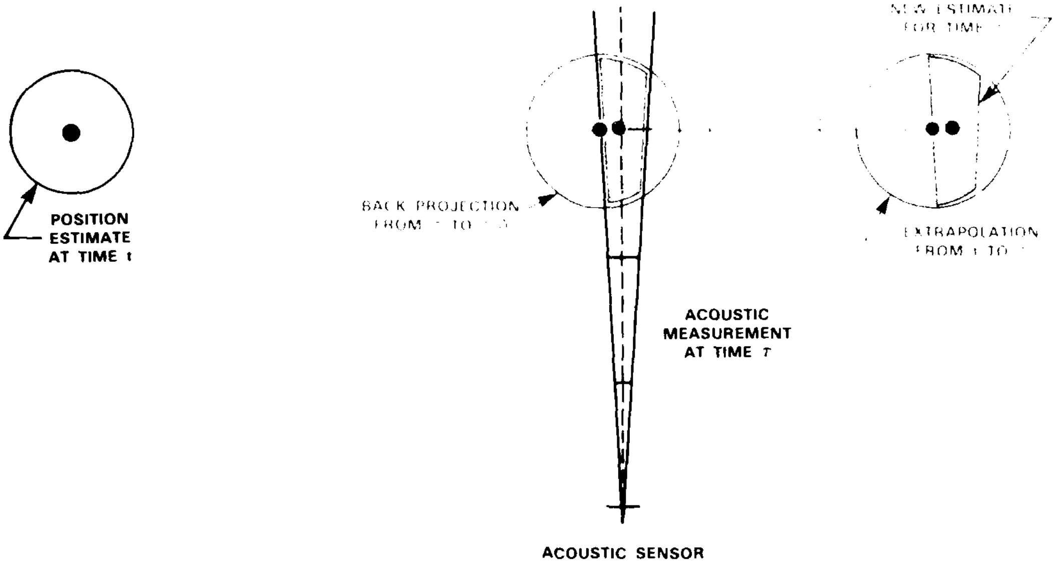
The first step in updating a position track is to extrapolate the position track to the measurement time τ. The rightmost red dot and circle in Figure 6.4 represent this extrapolation. Unlike the previous example with a TV sensor, there is no intersection between the extrapolated confidence region and the measurement confidence region. This is due to acoustic propagation delay.

The second step is to estimate the propagation delay by moving the target backward to the point where sound from the aircraft will arrive at the sensor at time τ. This is illustrated by the dashed line in the figure leading back toward the left red dot. The tick marks represent aircraft positions for equally spaced times in the past. The same time increments are indicated by the small arcs shown on the acoustic measurement. The arcs correspond to increasing propagation delays as well as distances away from the sensor. The correct projected position, represented by the left red dot, is that for which the propagation time from the aircraft to the sensor equals the time the aircraft takes to move to the extrapolated position at time τ.

The third step is to shift the state estimate uncertainty region back in time by the estimated propagation delay $\hat{\delta}$. The result is the left red circle in Figure 6.4. Two constraint regions now restrict the aircraft location at time $\tau - \hat{\delta}$. A revised position estimate and the combined constraint region at time $\tau - \hat{\delta}$ can now be calculated. The final step is to shift that joint constraint and the position estimate forward in time to τ. The estimates and constraint regions at time $\tau - \delta$ and at the τ are shown in green on the figure.

As was the case with the TV sensor, the area in which the aircraft is presumed to lie after updating at time τ is usually less than the area in which it was presumed to lie at time t. That area is elongated with the error in one direction larger than it was before the update. But the confidence region will not be oriented along the line from the target location to the acoustic sensor because of the shift forward in time.

The mathematical details of the updating algorithm are as follows. Inputs consist of a position track at time t, represented by a state estimate and state covariance pair $\hat{X}_P(t \mid t)$ and



87882-18

Figure 6.4. Updating a track with an acoustic measurement: conceptual steps. See text for discussion.

$\Sigma_P(t\,|\,t)$, an acoustic sensor location represented by north and east position coordinates, s_N and s_E, and with an acoustic azimuth measurement at time τ, represented by a measurement and measurement variance pair $z_{AP}(\tau)$ and $\Theta_{AP}(\tau)$. The measurement variance represents confidence in the measurement just as the state covariance does in the state estimate.

The first step is to extrapolate the position track to the measurement time, yielding $\hat{X}_P(\tau\,|\,t)$ and $\Sigma_P(\tau\,|\,t)$. The second is to estimate the propagation delay implicit in the acoustic azimuth measurement at time τ. The model of constant speed, constant heading motion results in the following equation for the propagation delay:

$$c\;\hat{\delta} = \left[\left(\hat{p}_N(\tau\,|\,t) - \hat{v}_N(\tau\,|\,t)\,\hat{\delta} - s_N\right)^2 + \left(\hat{p}_E(\tau\,|\,t) - \hat{v}_E(\tau\,|\,t)\,\hat{\delta} - s_E\right)^2\right]^{1/2} \tag{6.14}$$

where c is the speed of sound. This equation has one positive, real solution for $\hat{\delta}$ provided the estimated velocity is subsonic.

The third and fourth steps are merged in the application of the extended Kalman filter algorithm [15] as follows:

$$\hat{X}_P(\tau\,|\,\tau) = \hat{X}_P(\tau\,|\,t) + K_{AP}(\tau)\left[z_{AP}(\tau) - \hat{z}_{AP}(\tau\,|\,t)\right] \tag{6.15}$$

and

$$\Sigma_P(\tau\,|\,\tau) = \left[I - K_{AP}(\tau)\,C_{AP}(\tau)\right]\;\Sigma_P(\tau\,|\,t) \tag{6.16}$$

where

$$\hat{z}_{AP}(\tau\,|\,t) = \arctan\left[\hat{p}_N(\tau\,|\,t) - \hat{v}_N(\tau\,|\,t)\,\hat{\delta} - s_N\;,\hat{p}_E(\tau\,|\,t) - \hat{v}_E(\tau\,|\,t)\,\hat{\delta} - s_E\right], \tag{6.17}$$

$$K_{AP}(\tau) = \Sigma_P(\tau\,|\,t)\,C_{AP}(\tau)^T\left[C_{AP}(\tau)\,\Sigma_P(\tau\,|\,t)\,C_{AP}(\tau)^T + \Theta_{AP}(\tau)\right]^{-1}, \tag{6.18}$$

$$C_{AP}(\tau) = \begin{bmatrix} -\dfrac{\hat{p}_E(\tau \mid t) - s_E}{\beta\, \hat{r}(\tau) c\, \delta} \\ \dfrac{\hat{p}_N(\tau \mid t) - s_N}{\beta\, \hat{r}(\tau) c\, \delta} \\ \dfrac{\hat{p}_E(\tau \mid t) - s_E}{\beta\, \hat{r}(\tau)\, c} \\ -\dfrac{\hat{p}_N(\tau \mid t) - s_N}{\beta\, \hat{r}(\tau)\, c} \end{bmatrix}^T , \tag{6.19}$$

$$\beta = \left[1 - \frac{\left[\hat{p}_N(\tau \mid t)\, \hat{v}_E(\tau \mid t) - \hat{p}_E(\tau \mid t)\, \hat{v}_N(\tau \mid t) \right]^2}{\hat{r}(\tau)^2\, c^2} \right]^{1/2} , \tag{6.20}$$

and

$$\hat{r}(\tau) = \left[\left[\hat{p}_N(\tau \mid t) - s_N \right]^2 + \left[\hat{p}_E(\tau \mid t) - s_E \right]^2 \right]^{1/2} . \tag{6.21}$$

As one would expect, these formulae reduce to those for TV azimuth measurements in the limit as c goes to infinity. The similarities between the two updating procedures extend to the pragmatic modifications that must be made to avoid unrealistic corrections to the state estimate when the aircraft-to-sensor range is small or when there is a large difference between the TV azimuth measurement and the extrapolated aircraft azimuth relative to the TV sensor. An additional problem can occur if the estimated speed becomes close to the speed of sound. The pragmatic solution has been to clip the speed estimates to keep them less than the speed of sound, typically less than Mach 0.95.

Updating with acoustic measurements makes more extensive use of the aircraft motion model than updating with TV measurements, so it should not be surprising that it can be more sensitive to aircraft maneuvers. As with TV measurements it is beneficial to choose α_P as large as possible to avoid this problem. But the more general solution discussed in Section 6.2.11 may be a better approach, although it has not been tested.

Figure 6.5 show the results from an early real-time test-bed experiment designed to demonstrate tracking with a single acoustic node using the algorithms described above. A line connects all the estimated positions. Around each is an ellipse expressing the two-dimensional equivalent of a one-standard-deviation error bar. As with the confidence region in Figure 6.4, the initial error ellipse is circular. Updating produces eccentric ellipses with their long axes misaligned with the acoustic sensor line-of-sight, in keeping with Figure 6.4. The long axis slowly increases even as updates are made. When no more measurements are available, the ellipse grows along both axes.

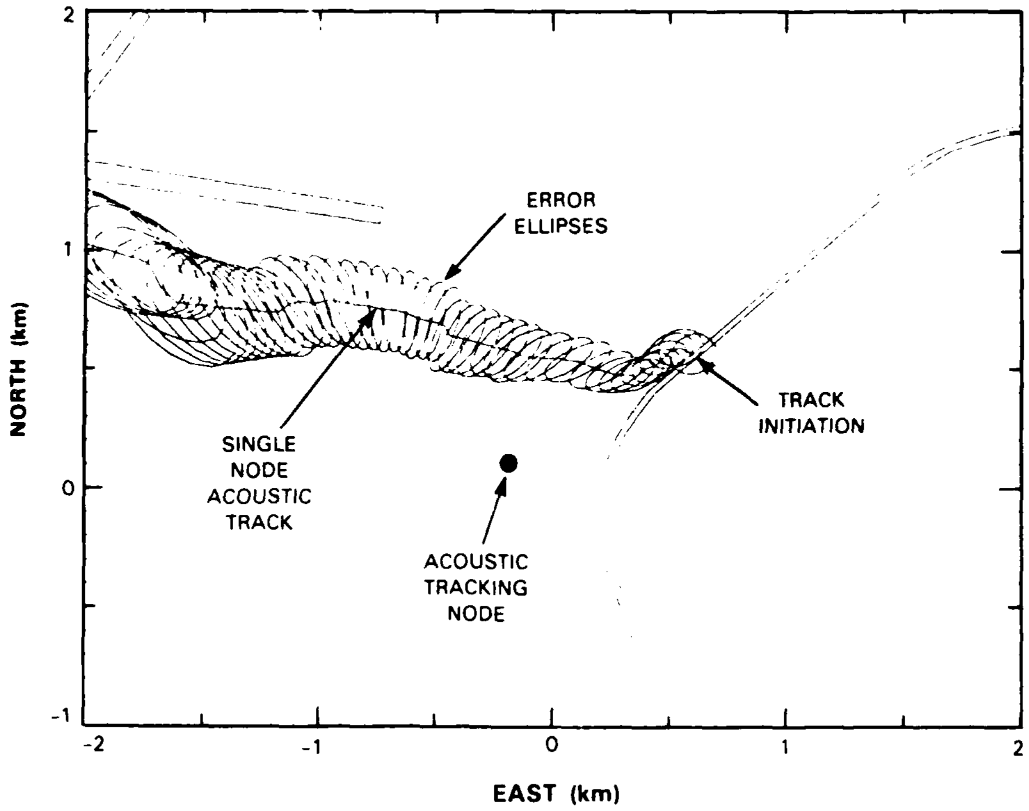


Figure 6.5. Real-time helicopter tracking with a single acoustic array.

6.2.5. Combining Local and Foreign Position Tracks

The preceding two sections describe how sensor measurements made by a node are directly incorporated into a track maintained by the node. The incorporation of information from sensors attached to other ("foreign") nodes is more complex. That information is first incorporated into a position track by the foreign node, then transmitted to the local node as a position track and is finally incorporated into the local position track by merging the local and foreign position tracks. The exchange of position track information between nodes is through broadcasts, with several nodes in general receiving each broadcast. Each node operates independently, considering itself a "local" node and all others as "foreign" nodes.

The merging of local and foreign tracks should optimally combine the information in each of the tracks. If the local and foreign position tracks are based on disjoint sets of sensor data this can be easily done by extrapolating the two position tracks to a common time and forming a weighted average as follows. Let the local position track be denoted by the state estimate and covariance pair $\hat{X}_{LTP}(t_{LTP} \mid t_{LTP})$ and $\Sigma_{LTP}(t_{LTP} \mid t_{LTP})$, the foreign position track by the pair $\hat{X}_{FTP}(t_{FTP} \mid t_{FTP})$ and $\Sigma_{FTP}(t_{FTP} \mid t_{FTP})$, and the common time by τ. Typically, t_{LTP} is the last time the local position track was updated using a TV or acoustic azimuth measurement, t_{FTP} is the last time the foreign position track was updated, and τ is the next local measurement time. The time τ is used since the times of local measurements are used in each node to trigger all track update and combining activity. After the extrapolations have been done, the weighted averaging takes the form:

$$\hat{X}_{LTP}(\tau \mid \tau) = \Sigma_{LTP}(\tau \mid \tau) \times$$

$$\left[\Sigma_{LTP}(\tau \mid t_{LTP})^{-1} \hat{X}_{LTP}(\tau \mid t_{LTP}) + \Sigma_{FTP}(\tau \mid t_{FTP})^{-1} \hat{X}_{FTP}(\tau \mid t_{FTP}) \right] \quad (6.22)$$

and

$$\Sigma_{LTP}(\tau \mid \tau) = \left[\Sigma_{LTP}(\tau \mid t_{LTP})^{-1} + \Sigma_{FTP}(\tau \mid t_{FTP})^{-1} \right]^{-1} \quad (6.23)$$

This is an optimum Bayesian combination of the probability density functions of the track estimates under the assumption that the extrapolated state estimates and covariances define independent Gaussian probability density functions for the true state of the aircraft.

But because of information exchange between nodes, the local and foreign tracks are not independent. Each is based upon its own local sensor data plus foreign data collected up to the time of the previous foreign node broadcast. Each local track contains some unique information and some that is common with other tracks. The unique information is provided by the locally attached sensors since the time of the last broadcast. The common information is any information acquired before that broadcast. If Equations (6.22) and (6.23) were used to merge local and foreign tracks, the common information would be counted twice.

One way to handle the common information problem is to keep a "common" track as well as a local track. The local track is the best track estimate available at each node, The "common" track can be used to keep a record of the information that is common between nodes. Let $\hat{X}_{CP}(t_{CP} \mid t_{CP})$ and $\Sigma_{CP}(t_{CP} \mid t_{CP})$ denote the state and covariance for the common position track at the time of the last internodal broadcast. The correct position track combination procedure becomes:

$$\hat{X}_{LTP}(\tau \mid \tau) = \Sigma_{LTP}(\tau \mid \tau) \times \tag{6.24}$$

$$\left[\Sigma_{LTP}(\tau \mid t_{LTP})^{-1} \hat{X}_{LTP}(\tau \mid t_{LTP}) + \Sigma_{FTP}(\tau \mid t_{FTP})^{-1} \hat{X}_{FTP}(\tau \mid t_{FTP}) - \Sigma_{CP}(\tau \mid t_{CP})^{-1} \hat{X}_{CP}(\tau \mid t_{CP}) \right]$$

and

$$\Sigma_{LTP}(\tau \mid \tau) = \left[\Sigma_{LTP}(\tau \mid t_{LTP})^{-1} + \Sigma_{FTP}(\tau \mid t_{FTP})^{-1} - \Sigma_{CP}(\tau \mid t_{CP})^{-1} \right]^{-1} \tag{6.25}$$

The state and covariance of the common position track are updated to time τ from time t_{cp} using the usual model equations given in Section 6.2.2.

Equations (6.24) and (6.25) implement an optimum Bayesian combination of Gaussian probability density functions if the information in common tracks and the added information in local tracks are independent. For some technical reasons related to target maneuvers [17], the independence assumption is not quite correct; but, fortunately, this does not result in any practical problems if the true state estimation errors are consistent with the error covariances, which seems to have been the case in most DSN test-bed experiments.

As described above, the time t_{CP} is the time of the last position track broadcast by either the local or foreign node, and $\hat{X}_{CP}(t_{CP} \mid t_{CP})$ and $\Sigma_{CP}(t_{CP} \mid t_{CP})$ are the estimate and covariance pair that were broadcast. In practice, each node may have different common position tracks. For example, a node may broadcast a position track and assume that the information in that position track is now common. But a foreign node may fail to receive the broadcast, and therefore cannot update its common track. Because of this each node must maintain its own version of common track. The common track for a node is set equal to the local track each time it broadcasts the corresponding local track.

The multiplicity of local and foreign tracks requires some additional nomenclature to avoid confusion. Hereafter what were simply called local and foreign position tracks will be called local and foreign *total* tracks to distinguish them from local and foreign *common* tracks. The notation of Equations (6.24) and (6.25) presaged this nomenclature. Similar notation will also be used to distinguish between local and foreign common position tracks.

With distinct local and foreign common tracks it is not obvious which one to use for track merging. Both options have been investigated, with the conclusion that the foreign common track is the correct choice. However, during early stages of algorithm development the local common track was used as the common track in Equations (6.24) and (6.25) since it is always locally available. This option was initially thought to be most desirable because it would require less internodal communication than the other option.

Unfortunately, the use of the local common track for track merging can lead to problems. A simple example of such problems is illustrated symbolically in Figure 6.6. Suppose each node starts with identical information I_0 in its total and common tracks. Both make measurements, gaining additional information M_L and M_F. The local node then broadcasts and changes its common position track. But the foreign node does not receive the broadcast and therefore makes no changes in either the total or common track. Measurements are again made, but for simplicity and illustrative purposes it is assumed that they add no information and serve only to trigger a second broadcast, this time by the foreign node. The local node changes both total and common tracks upon reception of the broadcast from the foreign node. At this time both nodes again have identical information in their total and common tracks. But the information from the local measurement M_L has been lost from the system permanently.

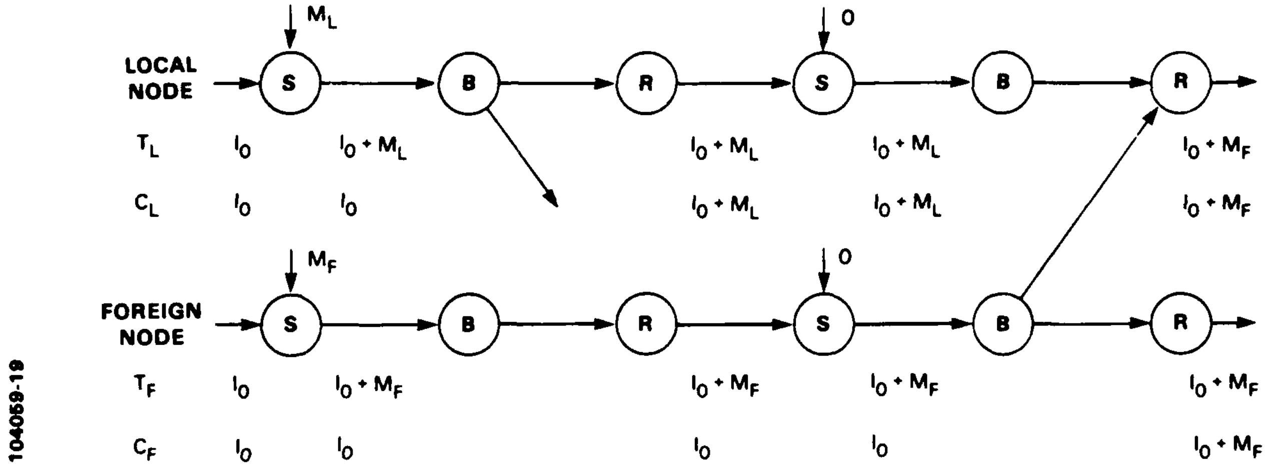


Figure 6.6. Information diagram illustrating serious information loss when only total tracks are broadcast. See text for discussion.

Use of the foreign common position track avoids the problem illustrated in Figure 6.6 as well as other system stability and track quality problems. It does require that each node broadcast a common track along with every total track. For convenience, we will continue to refer to position track broadcasts with the understanding that such broadcasts include total and common position track pairs. In addition, all unqualified references to position tracks outside this section will refer to "total" position tracks, not "common" position tracks.

The resulting track updating formulas for the total and common tracks are given by Equations (6.26) through (6.29). The formulas to update the local total position track become :

$$\hat{X}_{LTP}(\tau \mid \tau) = \Sigma_{LTP}(\tau \mid \tau) \times \tag{6.26}$$

$$\left[\Sigma_{LTP}(\tau \mid t_{LTP})^{-1} \hat{X}_{LTP}(\tau \mid t_{LTP}) + \Sigma_{FTP}(\tau \mid t_{FTP})^{-1} \hat{X}_{FTP}(\tau \mid t_{FTP}) - \Sigma_{FCP}(\tau \mid t_{FCP})^{-1} \hat{X}_{FCP}(\tau \mid t_{FCP}) \right]$$

and

$$\Sigma_{LTP}(\tau \mid \tau) = \left[\Sigma_{LTP}(\tau \mid t_{LTP})^{-1} + \Sigma_{FTP}(\tau \mid t_{FTP})^{-1} - \Sigma_{FCP}(\tau \mid t_{FCP})^{-1} \right]^{-1} \tag{6.27}$$

These formulas add the information in the local total track to the information which is in the foreign total track but not in the foreign common track, i.e., the information acquired by the foreign sensor or sensors since the previously received broadcast. The local common track must similarly incorporate that new information as follows:

$$\hat{X}_{LCP}(\tau\,|\,\tau) = \Sigma_{LCP}(\tau\,|\,\tau)\times \tag{6.28}$$

$$\left[\Sigma_{LCP}(\tau\,|\,t_{LCP})^{-1}\,\hat{X}_{LCP}(\tau\,|\,t_{LCP}) + \Sigma_{FTP}(\tau\,|\,t_{FTP})^{-1}\,\hat{X}_{FTP}(\tau\,|\,t_{FTP}) - \Sigma_{FCP}(\tau\,|\,t_{FCP})^{-1}\,\hat{X}_{FCP}(\tau\,|\,t_{FCP})\right]$$

and

$$\Sigma_{LCP}(\tau\,|\,\tau) = \left[\Sigma_{LCP}(\tau\,|\,t_{LCP})^{-1} + \Sigma_{FTP}(\tau\,|\,t_{FTP})^{-1} - \Sigma_{FCP}(\tau\,|\,t_{FCP})^{-1}\right]^{-1} \tag{6.29}$$

Only if the local and foreign common tracks are equal does this reduce to replacing the local common track with the foreign total track.

Figure 6.7 uses the same format as Figure 6.6 to illustrate one advantage of broadcasting both total and common tracks. As before, the foreign node does not receive the information M_L. But the system as a whole does not lose the information because the local node still retains it wnen the interactions and processing are completed.

Thus far we have assumed that both nodes begin with position tracks and that they are identical. But it is possible, as discussed more fully in the following section, for the local node to have no initial position track. In that case, the only track information available to the local node is a position track received from the foreign node. Local total and common position tracks are

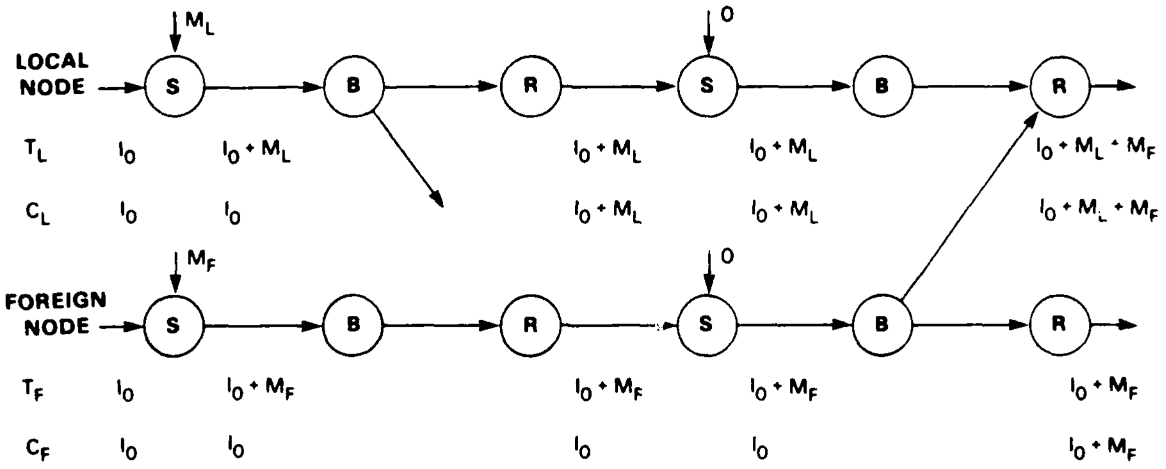


Figure 6.7. Information diagram illustrating minor information loss situation. Situation is the same as for Figure 6.6 except that total and common tracks are broadcast. See text for discussion.

created upon receipt of the foreign position track. The tracks are initialized with the state and covariance from the received foreign track. Figure 6.8 illustrates this aspect of position track combination.

Figures 6.6 through 6.8 are symbolic information flow graphs. They were generated using a symbolic information flow simulator that was developed to investigate what information should be communicated, when it should be communicated, and how it should be processed to achieve a robust position combining algorithm. Section 10.6 describes the simulator in somewhat more detail.

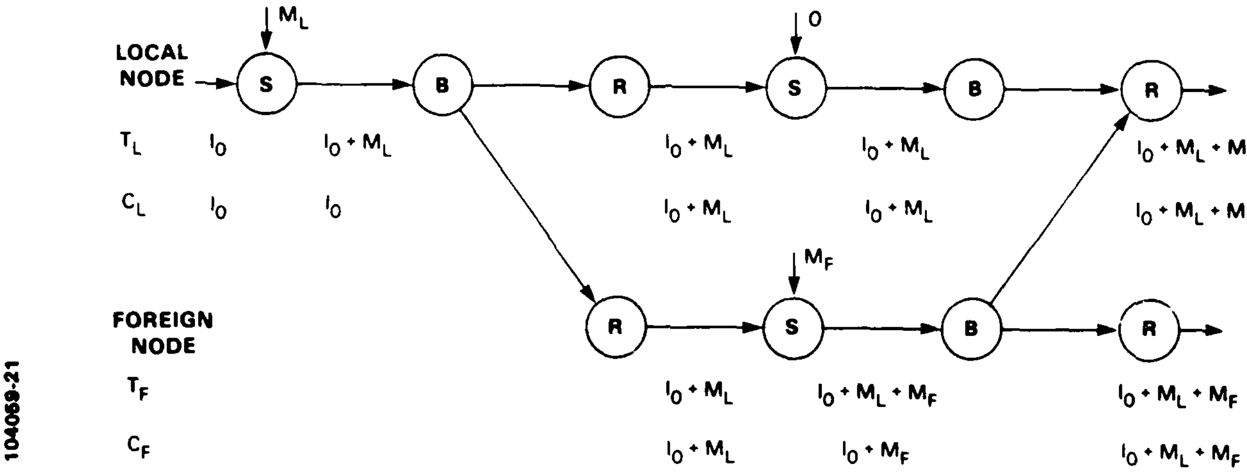


Figure 6.8. Symbolic information flow diagram illustrating alerting.

The above discussion is for two nodes. In practice more nodes will be involved, but that does not change the position track merging algorithm. It can be applied to each foreign position track individually in order of reception. The processing order for nearly simultaneous receptions does not matter; the merging operations are commutative.

6.2.6. Position Track Broadcast Policies

The preceding section described what to broadcast and how to use the broadcasts, but did not specify when to broadcast. The rules that determine when to broadcast are broadcast policies. There are many policy options. The test-bed tracking algorithm places only two weak constraints. First, a node should broadcast an aircraft track when it predicts that, for the first time, the aircraft is entering the detection range of a sensor attached to another node within broadcast range. Such a broadcast constitutes an "alerting" message to the receiving node. Second, a node should broadcast whenever a target is leaving the area of coverage of its own sensors. This broadcast is a "handover" message. Neither is essential but they are both clearly reasonable and very useful as discussed below.

Aircraft tracks are a mechanism for storing and sharing information in a DSN. A node very distant from an aircraft cannot detect it and therefore has no information to contribute to the tracking process. Although it might do so, it need not maintain a position track for such a target. In general, the question of which nodes maintain tracks for which targets is an important system design issue. The choice we have made is to match the tracking coverage area of a node to the coverage area of its attached sensors. This minimizes internodal communication requirements while fully utilizing all sensor data in the network. The two broadcast policy constraints mentioned above are reasonable consequences of this decision regarding track coverage area.

Implicit in the above discussion is the assumption that two nodes that simultaneously detect an aircraft should be within broadcast range of each other. If this is not the case, it may be necessary to forward track messages over multiple communication hops to reach a node that should obtain a particular message. This could substantially complicate the communication system as well as increase communication capacity requirements due to the message repetition.

Also implicit is the assumption that each node must model the sensor coverages of other nodes. To do so in detail is difficult; but fortunately, accurate modeling is not required, especially if the modeling is conservative. The modeling is conservative if the model coverage area is greater than the true sensor coverage. At worst this will result in a premature alerting message that lacks some information that could have been added later. However, a greatly delayed alerting message, which can result if the assumed sensor coverage is too small, can cause confusion. The confusion will result if the receiving node initiates a track for the target before the alerting broadcast is received. In this case there will be redundant tracks with different track identifiers. (Assignment of unique position track identifiers is described in Section 6.2.8.) Smaller alerting delays will result in some information loss but will not cause serious problems such as the creation of redundant tracks.

Each node must also model its own sensor coverage to decide when a received track broadcast should be ignored and when to delete tracks from its track file. When a track is deleted the node must also broadcast the track if the track has been recently updated with local sensor data. Otherwise, the most recent sensor information from the node will be lost forever to the DSN. This last broadcast from the node is a handover message.

Although coverage models need not be accurate, provided they err on the conservative side, they must be consistent from node to node. Otherwise, a node could send out an alerting message only to have the neighboring node ignore the message.

The sensor coverage models used in the DSN test bed are very simple. Acoustic sensors are assumed to have a circular coverage area, with perfect detection within the circle and no detections outside of the circle. The model also takes into account the acoustic propagation delay and predicts the time at which sound from a target entering or leaving a coverage area will reach the sensor. The coverage of TV sensors is assumed to be very large and was effectively ignored for the purposes of position track broadcast policy. The DSN test bed performs reasonably well, even though these models are clearly inaccurate, demonstrating that rough models are sufficient.

A broadcast policy resulting only in alerting and handover messages is a minimal broadcast policy. Figure 6.9 illustrates the sequence of broadcasts that would occur under such a policy as an aircraft flies through the acoustic coverage of four nodes. For simplicity, acoustic propagation effects have been ignored in the figure. A step-by-step discussion of the broadcasts and tracking functions for this simple case should serve to clarify how the overall system operates. For this purpose we have assumed that the broadcast range is large enough so that every broadcast can reach every node.

Assume that the target is initially within the coverage of Node 1, and that Node 1 is tracking it. Node 1 broadcasts an alerting message as the aircraft enters the coverage region of Node 2. Both nodes subsequently and independently maintain tracks, starting with the same information but adding different information from their local sensors. A while later the aircraft passes beyond coverage of Node 1. When that occurs, Node 1 broadcasts a handover message and drops the position track. Upon receipt of the handover message, Node 2 incorporates it into its position track, thereby preserving the information acquired by Node 1 between the alerting and handover message broadcasts. Nodes 3 and 4 will ignore the alerting message from Node 1.

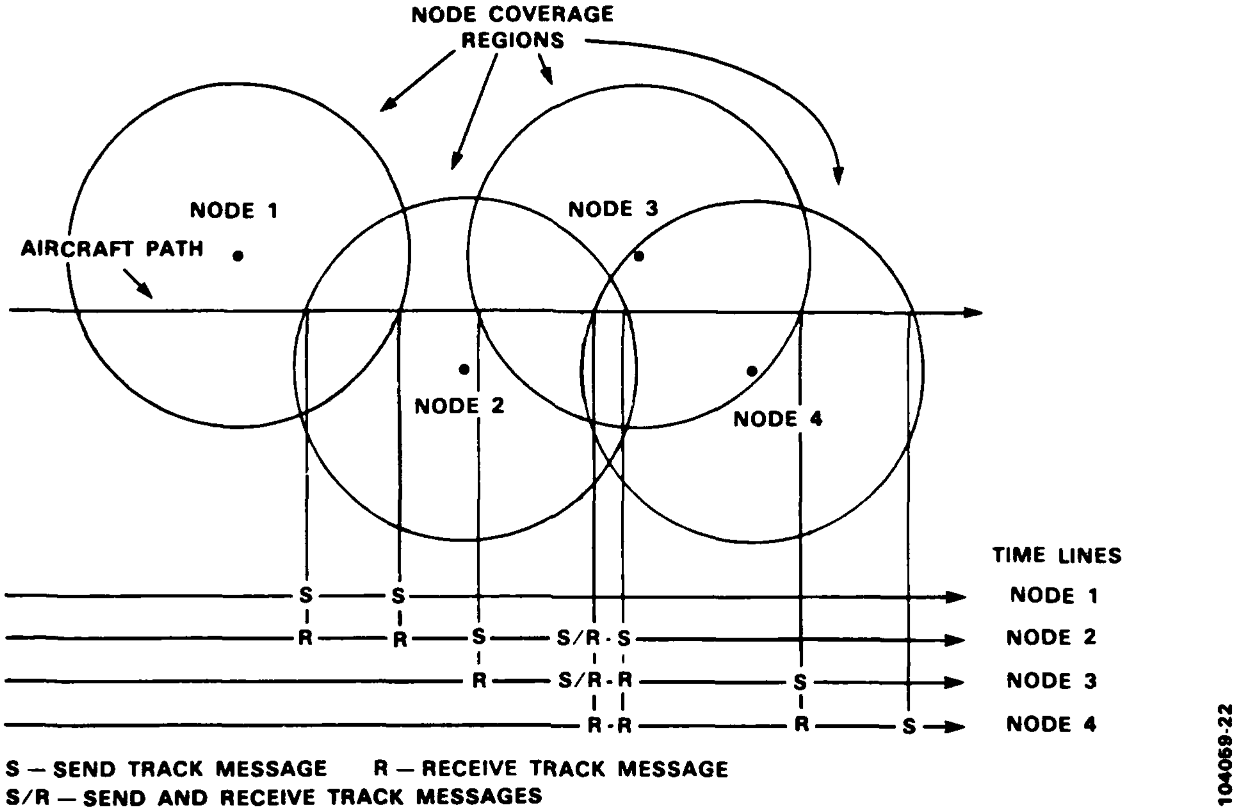


Figure 6.9. Illustration of track broadcast sequence for a single aircraft.

Node 2 similarly alerts Node 3 at the appropriate time. When Node 1 receives this broadcast, it ignores it since the aircraft is beyond the region of interest to Node 1. Nodes 2 and 3 both broadcast alerting messages to Node 4. One of these will serve to initiate a track at Node 4 and the other will simply be merged to improve the track. Nodes 2 and 3 receive each other's alerting message and opportunistically incorporate new information into their tracks. At this point, Nodes 2, 3, and 4 all have identical position tracks.

The ultimate handover message from Node 2 is received by Nodes 3 and 4, and Node 2 drops the track after the handover broadcast. Nodes 3 and 4 both incorporate the new information that has accumulated since the previous alerting message from Node 2. At this point the Node 3 and 4 position tracks are not identical, since each includes different information obtained from their own acoustic sensors.

Note how information about the aircraft propagates across the DSN along with the aircraft. Only those nodes which can detect an aircraft are ''aware'' of it.

The minimal broadcast policy contributes to system covertness and reduces communication requirements, but there is a price to pay for the lack of redundancy, especially since broadcast communications do not guarantee that all messages will be received. The most obvious problems occur when alerting or handover messages are not received. More subtle problems result from excessive reliance on local sensor data for lengthy time periods.

The more subtle problems were investigated in a series of experiments carried out with real acoustic measurements. The test bed was used to process prerecorded acoustic data in real time to experiment with different broadcast policies. The acoustic data were collected using a UH-1 helicopter and four acoustic arrays aligned roughly parallel to the flight path. The flight path was west to east, from left to right across Figure 6.10. The figure shows the tracks produced in real time by two of the nodes. Error ellipses provided by the tracker are also shown. The abrupt changes in the tracks and error ellipses correspond to the broadcasts of alerting or handover messages.

The track generated by the leftmost node was initially poor, but by the time the helicopter left the coverage of that node, the tracking performance was much improved. In comparison the track from the last node along the track is initially very good because of alerting messages received from other nodes. It is important to emphasize that these tracks depend in a complicated way upon the information from all nodes in this small network. No node performs tracking in isolation from the others.

The maximal broadcast policy is to broadcast every time a track is locally updated with sensor measurements. This prompt distribution of information will result in more accurate tracks as illustrated in Figure 6.11. Everything is the same as in Figure 6.10 except for the broadcast policy which was made maximal. The performance improvement is most pronounced for the early part of the track than for the later part. This behavior is representative of a more general observation: Prompt distribution of information is most important when track accuracy is poor and there is substantial room for improvement. This is usually true near track initiation time. It is also true, although not illustrated by the example, that prompt distribution is much more important for

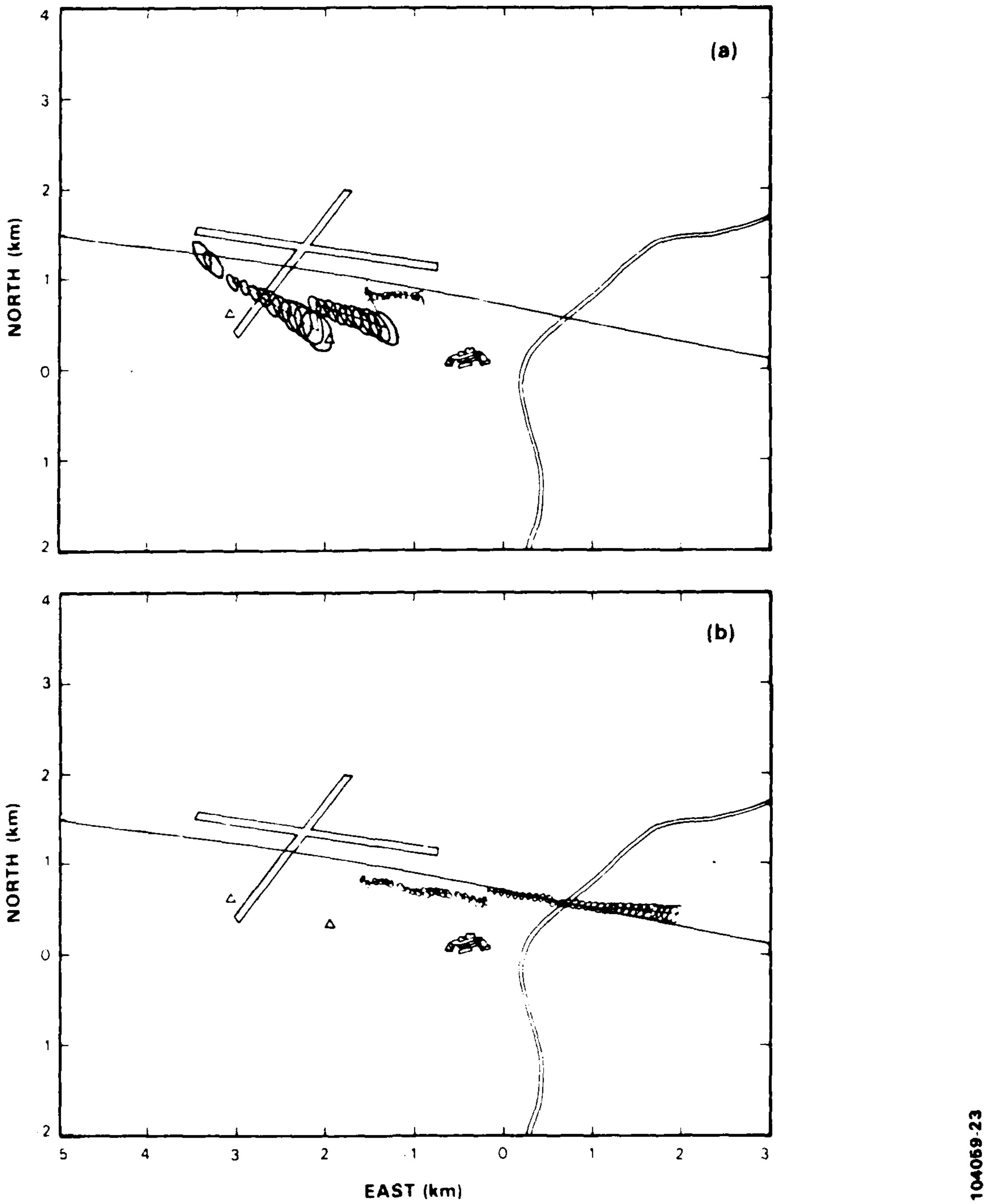


Figure 6.10. Helicopter tracking example for minimal communication policy. (a) Track from leftmost node; (b) track from rightmost node.

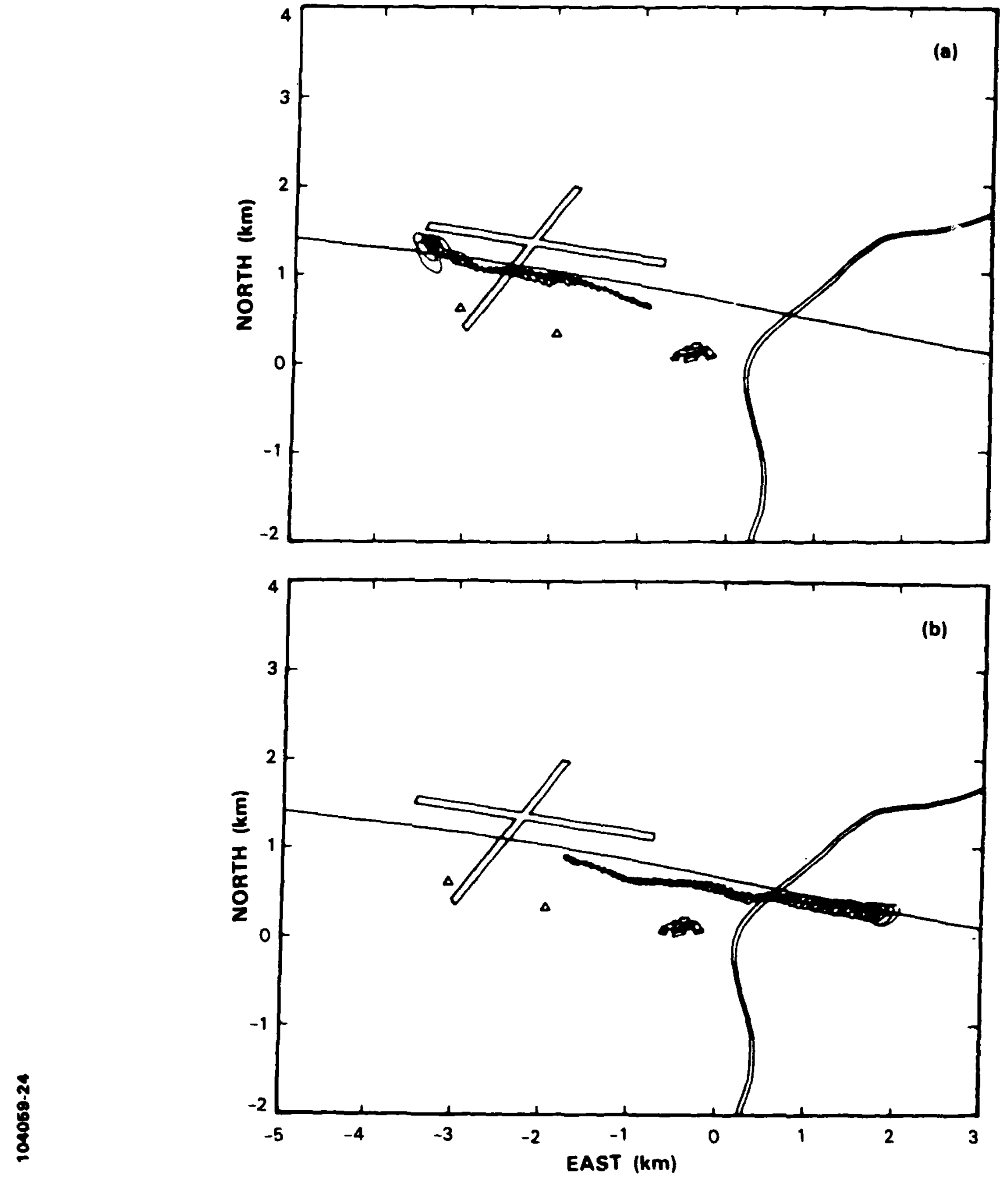


Figure 6.11. Helicopter tracking example for maximum communication policy. (a) Track from leftmost node; (b) track from rightmost node.

maneuvering targets than for nonmaneuvering targets. It is also obvious that a maximal broadcast policy has the additional advantage of minimizing the effect of ''lost'' broadcast messages, i.e., those position track broadcasts that are not received by every node within range for some reason.

The test-bed system provides for a parametrized family of options covering the spectrum from minimal to maximal broadcast policies. First, provision is made to broadacst every N-th time a position track is updated. All nodes operate with the same value of N but each node maintains a separate counter for each target. Second, broadcasts of position tracks with large error covariances can be suppressed on the assumption that they will not provide much significant information. When no broadcasts are suppressed and N=1, the result is the maximal policy. If no broadcasts are suppressed on the basis of error covariances but N is very large, the policy is minimal. If broadcasts are suppressed on the basis of error covariance and N is very large, the policy is still essentially minimal but the criteria for initiating tracks becomes more conservative. Thus an alternative to viewing error-covariance-based broadcast suppression as a broadcast policy is to view it as a modification of the definition of what constitutes a track.

It is easy to conceive of more complex policies that call for frequent broadcasts immediately after track initiation and during maneuvers, and for minimal broadcasts otherwise. But the simple options described above were found adequate for our investigation of communication issues.

It is not clear if the communication saving from other than optimal policies is worth the additional system complexity or reduced tracking performance that it might entail. The answer will be application dependent. However, it should be noted that there is an obvious alternative to our approach to distributed tracking if a maximal broadcast policy is deemed necessary to maximize tracking performance. The broadcast of position tracks is the primary mechanism that we selected for sharing information between nodes. This is the best approach if broadcasts are not frequent. But we could have elected to broadcast sensor measurements as soon as they were available and have each node directly integrate them into tracks. The tracker performance would be the same as with our approach and a maximal broadcast policy. The number of bits required to communicate a total and common position track pair is generally greater than the number required to communicate a sensor measurement. Thus, the alternative approach might achieve the performance of our system with a maximal broadcast policy with less internodal communication. It should also be noted that this alternative could be implemented using the same tracking algorithms that we have developed.

6.2.7. Acoustic Azimuth Tracking

Nodes will detect targets for which they do not yet have position tracks. For example, a node at the edge of a DSN cannot form a position track using only its own acoustic azimuth measurements. (It is conceptually possible to imagine a track with an exceedingly large confidence region, but we have elected to not take that viewpoint.) Acoustic azimuth information from two or more sites is required to begin tracking the position of an aircraft. One option for organizing and exchanging this information, the one selected for the test bed, is to form azimuth tracks at each node and to broadcast the tracks between nodes. The azimuth tracks are obtained by filtering acoustic azimuth measurements to obtain estimates of azimuth time rates of change as well as azimuths. These two quantities are used by the track initiation algorithms described in Section 6.2.8. Of course, acoustic azimuth tracks must be obtained without having estimated aircraft positions.

An azimuth track is maintained by a node until used to initiate a position track or until a foreign position track is received with which the azimuth track can be associated. The azimuth track is then discarded. In addition, an azimuth track is discarded if no new measurements have been added to it for a long time period.

The acoustic azimuth tracking method is the same as that for updating position tracks with sensor measurements, namely Kalman filtering. This requires a dynamic model for the acoustic azimuth measurements, but the model cannot depend upon the aircraft spatial position. We have elected to model the azimuth measurements by assuming a a constant azimuth rate. This is at best a rough approximation since the azimuth rate during a straight constant velocity aircraft flyby starts at zero, becomes large, and dies away to zero. Among others, the azimuth model exactly matches an aircraft flying in a circle around the sensor at constant speed or traveling in a spiral with continuously changing speed. But the azimuth tracker is important only during early detection, before a position track has been established. Fortunately, the model usually serves quite well up to and through the time of position track initiation.

Let $p_\theta(t)$ and $v_\theta(t)$ be the acoustic azimuth and azimuth rate of an aircraft relative to a node at time t. These two quantities can be grouped into a 2-dimensional state vector $X_A(t)$:

$$X_A(t) = \begin{bmatrix} p_\theta(t) \\ v_\theta(t) \end{bmatrix} \tag{6.30}$$

Assuming a constant azimuth rate, the state $X_A(\tau)$ at time τ can be calculated from the state at time t by:

$$X_A(\tau) = A_A(\tau - t)\, X_A(t) \tag{6.31}$$

where

$$A_A(d) = \begin{bmatrix} 1 & d \\ 0 & 1 \end{bmatrix} \tag{6.32}$$

Estimated state vectors $\hat{X}_A$, denoted by a $\hat{}$ over the state symbol, can be extrapolated using the same equations.

State estimate confidences are stored as a covariance matrix Σ_A. The formula for extrapolating the covariance matrix is:

$$\Sigma_A(\tau | t) = A_A(\tau - t)\, \Sigma_A(t | t)\, A_A(\tau - t)^T + \Xi_A(\tau - t) \tag{6.33}$$

where

$$\Xi_A(d) = \alpha_A \; B_A(d)\, B_A(d)^T, \tag{6.34}$$

$$B_A(d) = \begin{bmatrix} \frac{d^2}{2} \\ d \end{bmatrix}, \tag{6.35}$$

and α_A is a constant. The constant α_A can be considered simply as a parameter used to control the loss of confidence in the extrapolated state estimate or it can be viewed as the variance of a random azimuth acceleration.

Acoustic azimuth tracks are updated using acoustic azimuth measurements. The first step is to extrapolate the acoustic azimuth track to the measurement time, yielding $\hat{X}_A(\tau | t)$ and $\Sigma_A(\tau | t)$.

The updated state estimate and covariance are then computed using the conventional Kalman filter algorithm [18] according to:

$$\hat{X}_A(\tau \mid \tau) = \hat{X}_A(\tau \mid t) + K_{AA}(\tau)\left[z_{AA}(\tau) - \hat{z}_{AA}(\tau \mid t)\right] \tag{6.36}$$

and

$$\Sigma_A(\tau \mid \tau) = \left[I - K_{AA}(\tau)\, C_{AA}(\tau)\right] \Sigma_A(\tau \mid t) \tag{6.37}$$

where

$$\hat{z}_{AA}(\tau \mid t) = \hat{p}_\theta(\tau \mid t) = C_{AA}(\tau)\, \hat{X}_A(\tau \mid t), \tag{6.38}$$

$$K_{AA}(\tau) = \Sigma_A(\tau \mid t)\, C_{AA}(\tau)^T \left[C_{AA}(\tau)\, \Sigma_A(\tau \mid t)\, C_{AA}(\tau)^T + \Theta_{AA}(\tau)\right]^{-1}, \tag{6.39}$$

and

$$C_{AA}(\tau) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T . \tag{6.40}$$

For the Gaussian probability density function interpretation of the acoustic azimuth track and the measurement, the Kalman filter implements a Bayesian combination of the information inherent in the acoustic azimuth track and in the measurement.

As noted previously, the azimuth rate during a straight constant speed flyby is not constant. It can become large when the point of closest approach to a node is small. When that occurs the acoustic azimuth track can diverge. The random acceleration constant used in extrapolation of the state covariance can be adjusted to provide some immunity to this at the cost of less accurate azimuth tracks. In practice we have not found divergence to be a serious problem, primarily because position tracks tend to be initiated before targets approach their points of closest approach. Even when that has not been the case, the system quickly recovered by forming a new acoustic azimuth track of the aircraft. The usual effect of these rare divergences was a delay in acoustic position track initiation. These conclusions were obtained primarily from experiments with low-speed helicopter targets and might require some revision for higher speed targets.

Acoustic tracks must be initiated before Kalman filtering algorithms can update them. The technique used in the test bed is simple but effective. Two sequential acoustic azimuth measurements are required. The second measurement is taken as the azimuth estimate and the difference between the two azimuths, normalized by the time difference between the two measurements, is used as the azimuth rate estimate. The state covariance is computed from measurement covariances assuming that the two measurements are independent Gaussian random variables. In this process the confidence assigned to the first measurement is artificially reduced by increasing its measurement variance, just as state covariances are increased in extrapolation. This avoids assigning too great a confidence to the azimuth rate component of the estimated state vector.

6.2.8. Acoustic Position Track Initiation

Acoustic position track initiation processing is performed whenever a node with a local acoustic azimuth track receives an acoustic azimuth track from another node. If possible the two acoustic azimuth tracks are used to form a position track. If a position track is initiated it is assigned a track identifier that uniquely identifies it. The position identifier consists of a concatenation of the acoustic azimuth track identifiers used to form the position track. Each azimuth track identifier consists of an identifier for the node that produced the azimuth track followed by a number assigned by that node. The concatenation of azimuth track identifiers to form position track identifiers is performed in lexical order to avoid ambiguity.

Track naming conventions, in conjunction with other system design features, are designed so that all the tracks for a particular aircraft will be assigned the same identifier by every node in the network. Clearly, any nodes forming a track with the same two azimuth tracks will assign the same identifier. Thereafter, if alerting messages arrive early enough to be effective, other nodes will assign and use the same identifier. But problems can occur. If more than two nodes almost simultaneously initiate a track but do not use the same azimuth measurement pair, then multiple identifiers will exist. This is an unlikely event, but it has occurred in one case during test-bed experimentation. The result was two poor tracks when there should have been one good track. This demonstrated the need for algorithms to detect when this happens and to take corrective action. Another situation when multiple identifiers may be assigned to the same target is when the system loses a target for a short period of time and later initiates a new track. This is not as serious because it does not affect the quality of the tracks, although one might ultimately wish to associate such track segments with each other. The detection and combining of tracks with different identifiers might be a higher level task included as part of the multisite data integration function. (See Section 13 for more discussion. Although that discussion addresses multisite integration, it is clear that techniques for the removal of redundant identifiers should be made part of the distributed tracking algorithms whenever practical.)

The first step in combining two azimuth tracks to form a position track is to extrapolate them to a common time. Let $\hat{X}_{LA}(t_{LA} \mid t_{LA})$, $\Sigma_{LA}(t_{LA} \mid t_{LA})$, and $\hat{X}_{FA}(t_{FA} \mid t_{FA})$, $\Sigma_{FA}(t_{FA} \mid t_{FA})$ be the state estimate and covariance pair provided by the local and foreign node, respectively, and let τ be the common time. (As implemented in the test bed, t_{LA} is usually the last time the local acoustic azimuth track was updated, t_{FA} is the last time the foreign acoustic azimuth track was updated, and τ is the next local measurement time.) Extrapolation is based upon the same measurement model as azimuth tracking.

The remainder of the procedure is analogous to triangulation but more complex. The added complexity has three sources: the need for velocity as well as position estimates; the need to compensate for acoustic propagation delays; and the need to calculate covariances.

Neglecting acoustic propagation delay, the problem of aircraft location using two azimuth estimates reduces to the geometrical problem of finding the intersection of two lines radiating from two sensor locations. It can also be viewed as solving two equations in two unknowns. Each equation expresses the azimuth at one node as a function of the aircraft position (in two dimensions). There is one equation for each sensor site. The "knowns" are the estimated azimuths and sensor locations, and the "unknowns" are the estimated position coordinates. A solution exists under most circumstances although it does not exist when the estimated local and foreign azimuths are identical or reciprocal.

Differentiating those two equations with respect to time gives two more equations. These equations introduce azimuth rates as two new knowns and aircraft velocities as two new unknowns. Under the same circumstances as above, the resulting four equations can be solved for the four "unknowns," the elements of the aircraft's position track state estimate.

The introduction of acoustic propagation delay requires that the model for aircraft motion be incorporated into the equations. The model is used to move the aircraft backward in time as in Section 6.2.4. The resulting equations are nonlinear but fortunately can be rewritten to yield the following nearly linear and easily solvable matrix equation for the position track state estimate:

$$\begin{bmatrix} -\sin p_{L\theta}(\tau | t_{LA}) & \cos p_{L\theta}(\tau | t_{LA}) & -\frac{s_{LE}}{c} & \frac{s_{LN}}{c} \\ -\sin p_{F\theta}(\tau | t_{FA}) & \cos p_{F\theta}(\tau | t_{FA}) & -\frac{s_{FE}}{c} & \frac{s_{FN}}{c} \\ v_{L\theta}(\tau | t_{LA}) \cos p_{L\theta}(\tau | t_{LA}) & v_{L\theta}(\tau | t_{LA}) \sin p_{L\theta}(\tau | t_{LA}) & \sin p_{L\vartheta}(\tau | t_{LA}) & -\cos p_{L\theta}(\tau | t_{LA}) \\ v_{F\theta}(\tau | t_{FA}) \cos p_{F\theta}(\tau | t_{FA}) & v_{F\theta}(\tau | t_{FA}) \sin p_{F3}(\tau | t_{FA}) & \sin p_{F\theta}(\tau | t_{FA}) & -\cos p_{F\theta}(\tau | t_{FA}) \end{bmatrix} \hat{X}_P(\tau | \tau) =$$

$$\begin{bmatrix} s_{LE} \cos p_{L\theta}(\tau | t_{LA}) - s_{LN} \sin p_{L\theta}(\tau | t_{LA}) \\ s_{FE} \cos p_{F\theta}(\tau | t_{FA}) - s_{FN} \sin p_{F\theta}(\tau | t_{FA}) \\ v_{L\theta}(\tau | t_{LA}) \left\{ s_{LN} \cos p_{L\theta}(\tau | t_{LA}) + s_{LE} \sin p_{L\theta}(\tau | t_{LA}) \right\} \\ v_{F\theta}(\tau | t_{FA}) \left\{ s_{FN} \cos p_{F\theta}(\tau | t_{FA}) + s_{FE} \sin p_{F\theta}(\tau | t_{FA}) \right\} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \left\{ \hat{p}_N(\tau | \tau)\, \hat{v}_E(\tau | \tau) - \hat{p}_E(\tau | \tau)\, \hat{v}_N(\tau | \tau) \right\} /c \tag{6.41}$$

where $p_{L\theta}(\tau | t_{LA})$ and $v_{L\theta}(\tau | t_{LA})$ are the extrapolated local azimuth track azimuth and azimuth rate estimate, $p_{F\theta}(\tau | t_{FA})$ and $v_{F\theta}(\tau | t_{FA})$ are the extrapolated local azimuth track azimuth and azimuth rate estimate, s_{LN} and s_{LE} are the north and east position coordinates of the local acoustic sensor, and s_{FN} and s_{FE} are the north and east position coordinates of the foreign acoustic sensor. By introducing a new variable

$$\zeta = \hat{p}_N(\tau | \tau)\, \hat{v}_E(\tau | \tau) - \hat{p}_E(\tau | \tau)\, \hat{v}_N(\tau | \tau) \tag{6.42}$$

we can reduce the problem of solving for $\hat{X}_P(\tau | \tau)$ to the problem of solving a quadratic equation for ζ, substituting the solution back in above, and then solving the resultant linear matrix equation.

An apparent complication is that there can be two solutions for ζ. Fortunately it can be shown that only real solutions are possible, and one always turns out to be extraneous. The extraneous one is revealed by substituting the solution back into the equations for the azimuths as a function of the target state. At least one of the these will be the reciprocal of the measured values.

There are also cases when there are no solutions for $\hat{X}_P$, just as there are no solutions when triangulating with identical or reciprocal bearings. The no-solution regions differ from those for simple triangulation. However, the no-solution regions are still confined to one-dimensional subspaces withing the two-dimensional surface in which the aircraft is located. As a result, position tracks can be initiated almost everywhere using acoustic measurements.

The error covariance matrix for a new position track is estimated using the covariance matrices of the two contributing azimuth tracks. The four equations that relate the acoustic azimuth track state estimate to the position track state estimate are again relevant. From them, the partial derivatives of the acoustic azimuth track state estimates with respect to the position track state estimate can be calculated and arranged into matrices. Let Δ_{LAP} and Δ_{FAP} denote the partial derivative matrices: one for the local azimuth track and the other for the foreign azimuth track.

The formula for Δ_{LAP} is as follows:

$$\Delta_{LAP} = \begin{bmatrix} C_{LAP} \\ C'_{LAP} \end{bmatrix} \tag{6.43}$$

where C_{LAP} is the same as C_{AP} in Section 6.2.4,

$$C'_{LAP} = -\gamma_L \; C_{LAP} + \begin{bmatrix} -\dfrac{\hat{v}_{L\theta}(\tau \mid t_{LA}) \left[\hat{p}_N(\tau \mid \tau) - \hat{v}_N(\tau \mid \tau)\, \hat{\delta} - s_{LN} \right]}{\beta_L \; \hat{r}_L(\tau)\, \delta\, c} \\ -\dfrac{\hat{v}_{L\theta}(\tau \mid t_{LA}) \left[\hat{p}_E(\tau \mid \tau) - \hat{v}_E(\tau \mid \tau)\, \hat{\delta} - s_{LE} \right]}{\beta_L \; \hat{r}_L(\tau)\, \delta\, c} \\ -\dfrac{\left[\hat{p}_E(\tau \mid \tau) - \hat{v}_E(\tau \mid \tau) \hat{\delta} - s_{LE} \right]}{\beta_L \; \hat{r}_L(\tau)\, \delta\, c} \\ \dfrac{\left[\hat{p}_N(\tau \mid \tau) - \hat{v}_N(\tau \mid \tau) \hat{\delta} - s_{LN} \right]}{\beta_L \; \hat{r}_L(\tau)\, \delta\, c} \end{bmatrix}^T , \tag{6.44}$$

$$\gamma_L = \frac{\hat{p}_N(\tau \mid \tau)\, \hat{v}_N(\tau \mid \tau) + \hat{p}_E(\tau \mid \tau)\, \hat{v}_E(\tau \mid \tau)}{\beta_L{}^2 \, \hat{r}_L(\tau)^2} , \tag{6.45}$$

$\hat{r}_L(\tau)$ is the same as $\hat{r}(\tau)$ in Section 6.2.4, and β_L is the same as β. The formula for Δ_{FAP} is identical except that it uses the foreign acoustic azimuth track state estimate and sensor location.

The partial derivative matrices are used to calculate the position state covariance as follows:

$$\Sigma_P(\tau\,|\,\tau) = \left[\Delta_{LAP}{}^T\ \Sigma_{LA}(\tau\,|\,\tau)^{-1}\ \Delta_{LAP} + \Delta_{FAP}{}^T\ \Sigma_{FA}(\tau\,|\,\tau)^{-1}\ \Delta_{FAP}\right]^{-1}. \tag{6.46}$$

As the equation makes clear, the position track state covariance is large (and so confidence in the estimate is small) when either the local or foreign acoustic azimuth track state covariances are large. The position track state covariance is also large when both Δ_{LAP} and Δ_{FAP} are small. The partial derivatives are dependent on the geometry of aircraft and sensors. In some geometries the partial derivatives are small, resulting in large confidence regions for newly initiated tracks. These geometries are "close" to where no solution exists for the position track state estimate. The calculated state covariance is used to detect this situation and suppress track initiation . When a track is initiated, both total and common tracks are initiated.

Note that when a new position track is created, the aircraft is in the coverage of at least the local and foreign nodes that produced the azimuth tracks used to initiate the position track. Whenever a node initiates a track it must broadcast it to alert the foreign node which, for whatever reason, may not have independently initiated a track. If the foreign node has independently initiated track, there still is no problem because it will have assigned the same track identifier as the local node. The aircraft may also be within the coverage of other nodes and the alerting broadcast will serve to alert them as well.

As with other parts of the tracking algorithm, the acoustic position track initiation algorithm relies on an acceleration-free model of target motion. Acoustic delays can cause large track initiation errors when there is a significant change in heading or speed during the time between when the sound used for track initiation was emitted and the track initiation time τ. Even earlier maneuvers, resulting in inaccuracies in the acoustic azimuth tracks, can increase position track inaccuracies. But even without significant aircraft maneuvers, we have observed initial position errors larger than those expected on the basis of the calculated covariance matrix. The reason is not fully understood. A possible explanation is that the covariance matrix assigned to a newly initiated track is smaller than it should be due to the linearization involved in the calculation of the covariance matrix. Fortunately, in most cases, the initial errors have been promptly damped out by subsequent track updates.

6.2.9. Azimuth Track Broadcast Policies

Position track initiation using a particular acoustic azimuth track could be attempted as soon as the acoustic azimuth track is created from a sequential pair of acoustic azimuth measurements. In practice, both the azimuth track state covariance and the resulting new position track state covariance would often be unacceptably large. This is why position track initiation is suppressed until the contributing acoustic azimuth track state covariances are sufficiently small.

Since an acoustic azimuth track will not be used for position track initiation unless its state covariance is sufficiently small, there is no point to broadcasting it unless its state covariance is sufficiently small. Thus, the algorithms incorporate an acoustic azimuth track broadcast policy matched to the criteria that determine if an azimuth track is good enough to participate in track initiation. All acoustic azimuth tracks with sufficiently small covariances are broadcast. Tracks below the covariance threshold are not broadcast. (Actually, the test bed provides for broadcasting every Nth track value that is below the covariance threshold, but essentially all experimentation has been performed with N=1.)

A more restrictive broadcast policy, intended to further reduce network communication requirements, would be to broadcast the first time that the covariance falls below an acceptance threshold, and never again. However, this policy can create problems. Suppose that the local acoustic azimuth track is the first to have a sufficiently small state covariance. It would then be broadcast, but the foreign node would not use it, at least not until later. Further assume that the covariance of the azimuth track at the foreign node eventually drops below the threshold and it then broadcasts. Both the local and the foreign nodes can now attempt position initiation. But the geometry of the aircraft and sensors may be such that a position track will not be initiated because the position covariance is too large. If neither node broadcasts subsequent versions of their acoustic azimuth tracks, then neither will ever initiate track, even after the geometry changes to a more favorable one. Broadcast communication failures have a similar effect for the minimal broadcast policy.

6.2.10. Tracking Multiple Aircraft

The above sections emphasize single aircraft tracking without false detections. In practice, several aircraft may be within the coverage of a single sensor, resulting in several simultaneous azimuth measurements. In addition, the sensors may produce false detections that result from general background noise and other phenomena. (See Figure 4.6 as an example of a two target situation where lower peaks could also be taken as detections.) The distributed tracking algorithm has several features for coping with multiple aircraft and false detections.

At any given time there will be several position and acoustic azimuth tracks active in any given node. If the node has a TV sensor, it must decide which TV azimuth measurement, if any, to use in updating each position track. If it has an acoustic sensor, it must decide which acoustic measurement, if any, to use in updating each position and/or acoustic azimuth track, which to use to initiate new acoustic azimuth tracks in combination with old acoustic azimuth measurements, and which to save for future use. This problem is known as the data association problem and has been much studied [19]. Our approach has been to employ simple ad hoc data association methods while another DSN contractor has been investigating more sophisticated methods [20].

The test-bed approach to data association can be simply illustrated by considering the association of TV azimuth measurements and position tracks. Corresponding to each position track is an extrapolated TV azimuth and an associated variance. A TV azimuth measurement is associated with a position track if the difference between the measured and extrapolated azimuths, normalized by the sum of their variances, is sufficiently small and if it is smaller than that for any other TV measurements.

These association criteria can be stated with more mathematical precision. Assume that there are N TV azimuth measurements at time τ, represented by measurement and measurement variance pairs $z_{TVPi}(\tau)$ and $\Theta_{TVPi}(\tau)$ for i from 1 through N. Further assume that there are M (total) position tracks all extrapolated to time τ, represented by state estimate and state covariance pairs $\hat{X}_{Pj}(\tau \mid t)$ and $\hat{\Sigma}_{Pj}(\tau \mid t)$ for j from 1 through M. The ith TV azimuth measurement is associated with the jth position track if and only if

$$\frac{\left[z_{TVPi}(\tau) - \hat{z}_{TVPj}(\tau \mid t) \right]^2}{C_{TVPj}(\tau)\, \Sigma_{Pj}(\tau \mid t)\, C_{TVPj}(\tau)^T + \Theta_{TVPi}(\tau)} = \min_k \left\{ \frac{\left[z_{TVPk}(\tau) - \hat{z}_{TVPj}(\tau \mid t) \right]^2}{C_{TVPk}(\tau)\, \Sigma_{Pk}(\tau \mid t)\, C_{TVPk}(\tau)^T + \Theta_{TVPi}(\tau)} \right\} \tag{6.47}$$

and

$$\frac{\left[z_{TVPi}(\tau) - \hat{z}_{TVPj}(\tau \mid t)\right]^2}{C_{TVPj}(\tau)\, \Sigma_{Pj}(\tau \mid t)\, C_{TVPj}(\tau)^T + \Theta_{TVPi}(\tau)} \leq T_{TVP} \qquad (6.48)$$

where T_{TVP} is some threshold and where, for the jth position track, $\hat{z}_{TVPj}(\tau \mid t)$, $\hat{\Theta}_{TVPj}(\tau \mid t)$, and $C_{TVPj}(\tau)$ are the same as $\hat{z}_{TVP}(\tau \mid t)$, $\hat{\Theta}_{TVP}(\tau \mid t)$, and $C_{TVP}(\tau)$ in Section 6.2.3. If a TV azimuth measurement is associated with a position track, then the position track is updated as in Section 6.2.3 using the associated measurement. Unassociated TV measurements are discarded.

Acoustic azimuth measurements are handled similarly, subject to certain priorities for using the measurements. Acoustic measurements are first associated with position tracks if possible and the position tracks updated as in Section 6.2.3. Next, acoustic measurements that are not associated with position tracks are associated with acoustic azimuth tracks if possible. The acoustic azimuth tracks are then updated as in Section 6.2.7 using the associated acoustic azimuth measurements. (At most, one acoustic azimuth measurement will be associated with a single azimuth track.) The remaining unassociated acoustic azimuth measurements are compared with old unassociated acoustic azimuth measurements to determine if a new acoustic azimuth track should be initiated as described in Section 6.2.7. Any of the new acoustic azimuth measurements that are left unassociated at this stage become old unassociated acoustic azimuth measurements the next time the data association procedure is applied. All of the association criteria used in this process are variations on Equations (6.47) and (6.48).

The position track files typically contain several tracks. Position track broadcasts also contain several tracks. Part of the data association problem is to decide which pairs of local and foreign position tracks should be combined. An approach similar to that for sensor data association could be used but there is an additional option available. As described in Section 6.2.8, the system incorporates track naming conventions designed so that different nodes usually have the same name for tracks associated with the same target. Since the assignment of unique track identifiers is a specifically distributed aspect of the test bed, it was decided to use these identifiers as the association mechanism for position tracks. Thus, position track association is done using the track identifiers. Two tracks are associated if and only if their identifiers match. Position track association based on "closeness" in position and velocity could easily be added, and should be to catch redundant tracks, but is not now part of the system.

At any given time, a node may have several local acoustic azimuth tracks and receive additional ones from a foreign node. The foreign and local azimuth tracks must be paired for presentation to the track initiation algorithm. The solution used in the test bed is to form all possible pairs of local and foreign acoustic azimuth tracks into a list and run through the list, trying all of them. When position track initiation succeeds for a pair, all other pairs involving the same local or foreign acoustic azimuth track are deleted before continuing down the list. This procedure can obviously result in false pairings that nevertheless produce a track initiation. Such a mistake leads to the creation of a position track for which no aircraft exists, a "ghost" track. Because no aircraft exists for a ghost track, it is rare that later measurements continue to reinforce the track. The confidence of ghost tracks tends to decline steadily with time. The system deletes position tracks with low confidence on the assumption that they are ghosts. Because of the random effects and the asynchronous nature of much of the system operation, it is rare for an incorrect association to occur twice.

It is also possible to experience acoustic azimuth tracks even when no aircraft or other machinery is nearby. Such false tracks result from random sensor noise and are uncommon. These tracks are rarely updated and tend to persist for only a few time samples. They are quickly deleted because they are not updated. Nevertheless, false acoustic azimuth tracks do occasionally contribute to the creation of ghost position tracks.

6.2.11. Possible Improvements

Baseline distributed tracking algorithms have been developed but they require further evaluation and refinement before they will be ready for use in an operational system. Areas requiring more work include the handling of multiple and maneuvering targets and aircraft altitude effects. Following is a brief discussion of these and other areas for possible improvement.

The simple test-bed data association procedures work reasonably well for simple situations, but there is room for improvement. Following are two simple and obvious examples. When two or more measurements are close together, good measurements may be discarded or the wrong measurement may be associated with a track. Second, the position track association procedure does not recognize the rare situation when two position tracks for the same aircraft have different names.

Research performed by another DSN contractor, ADS, suggests how improvements might be made [20]. Multiple possible associations should be allowed and the consequences of each pursued for several time steps before deleting the less likely in favor of the more likely associations. For example, if two azimuth measurements could reasonably be associated with a position track, that position track should be duplicated and one copy updated using each of the azimuth measurements. Another example: If the position track initiation procedure might succeed for two foreign azimuth tracks paired with the same local acoustic azimuth track, apply it to both pairs and create two new position tracks. Note that this will require that a record be kept of which position tracks (and acoustic azimuth tracks) reflect mutually exclusive associations. This is to assure that fundamentally conflicting explanations of the data are not eventually accepted. Alternative track hypotheses can be kept active until more information is acquired. Incorrect associations will eventually be revealed if tracks are rarely updated or by other "aberrant" behavior.

Keeping a record of which tracks are mutually exclusive reduces to keeping a record of what measurements went into the tracks. Such a record can grow without bound as new measurements are acquired. This must be controlled and limited; a difficult task in a DSN where the record must be distributed across nodes. How best to maintain these records while avoiding excessive internodal communication is unclear at this time. Additional research and experimentation are needed on this problem.

Problems with maneuvering targets have been mentioned several times. There are techniques for handling maneuvers[21], developed primarily in the context of centralized tracking systems. The extension of these techniques to distributed tracking systems is a subject of ongoing research. One approach to maneuver detection is to make multiple, mutually exclusive hypotheses about the occurrence and magnitude of maneuvers. The effectiveness of this approach is not yet clear.

Aircraft and sensor geometries that lead to poor track initiation performance have not yet been fully analyzed. More analysis and experimentation is needed to better understand failure modes and exploit this understanding to improve system performance by improving the track initiation process. This could also result in reduced computation and communication loads by suppression of the broadcast or processing of azimuth tracks that should not be used for track initiation. As part of this process, the use of pairs of acoustic azimuth tracks for track initiation should be evaluated in detail and compared with other alternatives. One of these is to directly use N-tuples of acoustic azimuth measurements from pairs of nodes.

As one would expect, and experience with the DSN test bed has shown, the distributed tracking algorithm tracks ground vehicles and machinery as well as aircraft. The time behavior of tracks that correspond to ground activity is markedly different from those that correspond to aircraft. It should be possible to recognize ground activity and adapt the processing to improve performance and conserve processing resources. Expert system techniques might be used to recognize unusual tracks, including ghost tracks, that should be eliminated. Experimentation is required to evaluate the feasibility of this idea.

The acoustic direction finding algorithms provide azimuth information and the tracking algorithms assume that all targets are at very low (zero) altitude. However, aircraft will normally be detected at altitudes up to at least a few kilometers. More analysis and experimentation are needed to better understand the system sensitivity to aircraft altitude. Research should also be directed toward developing algorithms that specifically consider the three-dimensional problem. One option is to develop acoustic signal processing algorithms that provide elevation as well as azimuth information. This could be used to identify higher altitude targets to exclude from the tracking process. Alternatively, the tracking algorithms could be changed to track in three dimensions. Compared with excluding high-altitude targets, the success of this approach would be more dependent upon the accuracy of the sensor elevation measurements.

6.3. Delayed Acoustic Tracking

In the initial phases of the DSN project there was emphasis upon acoustic position track initiation and tracking with minimal assumptions concerning aircraft motion [22-24]. The thought was to defer the introduction of target motion models and thereby avoid problems that might result when the model was incorrect. Delayed acoustic tracking is a consequence of this approach. That is, because of acoustic propagation delays, it is possible only to know where aircraft have been but not where they are. It was recognized that dynamic models would eventually be needed to extrapolate tracks to current time and to integrate TV sensors. The plan was to do this in a second stage of tracking where aircraft locations derived from pairs of sensors would be combined into overall tracks.

Two different target location algorithms were developed consistent with this philosophy. Each one estimates aircraft locations using acoustic azimuth measurements from two nodes. One, the "possible position method," takes two acoustic azimuth tracks as input and produces an estimated aircraft position for a time T which is τ seconds in the past. The delay τ is an algorithm parameter. If τ is less than the propagation time from the aircraft to the most distant of the two sensors, no solution will be found. The other, the "reflection method," takes as inputs an acoustic azimuth track from one node and a single acoustic azimuth measurement from the other and produces an estimated aircraft position as well as an estimate of when the aircraft was at that position. Assuming that the node contributing the single azimuth measurement is furthest from the sound source, this algorithm estimates the minimum value of τ for which a solution can be obtained. Both methods assume that the speed of sound is known. Each makes use of acoustic azimuth tracks. Dynamic models are utilized only to generate azimuth tracks (see Section 6.2.7.). Neither requires models for aircraft motion in space.

The possible position method was so named because it uses a locus of possible aircraft positions for each acoustic azimuth track. A position estimate is obtained where two of these possible position curves intersect. The method is illustrated in Figure 6.12 which shows two hypothetical acoustic azimuth tracks, the corresponding possible position curves, and their intersection. The possible position curve for Node A is constructed as follows. Given the time T at which the aircraft is to be located, the sound from the aircraft must arrive at Node A some time between T and the current time. If sound was radiated at time T and received at time t, then it must have propagated over a distance of $\frac{t-T}{c}$, where c is the speed of sound. Thus, at time T an aircraft might have been located at this distance from Node A in the direction measured acoustically at time t. This is a possible position at time T corresponding to the measurement at time t. The entire possible position curve is produced as t sweeps from T to the current time. The possible position curve for Node B is obtained in the same way.

There are at least two alternatives for selecting T in a real-time system. One is to select T to be a fixed time in the past. Long distance acoustic detections will never be utilized if T is too recent. If T is in the distant past, then all estimated positions will be for the remote past and may be of little current interest. One solution to this problem is to duplicate the entire possible position target location process for several different values of delay. This will significantly increase

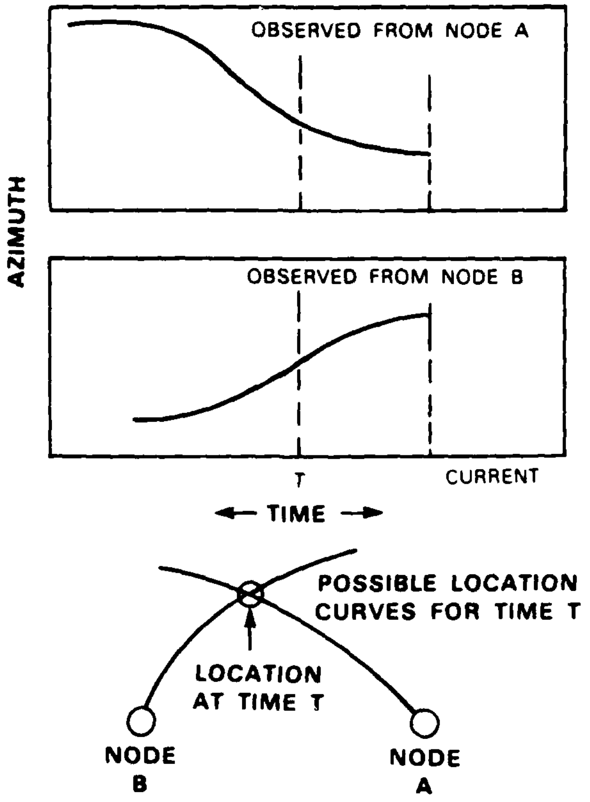


Figure 6.12. Illustration of location using possible position curves. Hypothetical azimuth-vs-time histories are illustrated for two nodes along with the target position at the intersection of possible position curves derived from the azimuth time histories.

the computational load as well as the complexity of the software. The added complexity is because a single value of T can appear many times, corresponding to different values of delay as real time progresses.

These difficulties with the possible position method motivated the development of the reflection method to estimate the aircraft location at the most recent time possible without modeling aircraft motion. Figure 6.13 illustrates how this method works. Suppose that Node A has been receiving signals from the aircraft for some time when Node B first detects the target at azimuth ϕ_B. The target could have been at any point along the straight line extending out from Node B when it radiated the sound resulting in the measurement ϕ_B. Each position along that line corresponds to an angle ϕ_A that would be measured but at a different time at Node A. If the travel times of sound from the target to A and B, respectively, are t_A and t_B then A would have

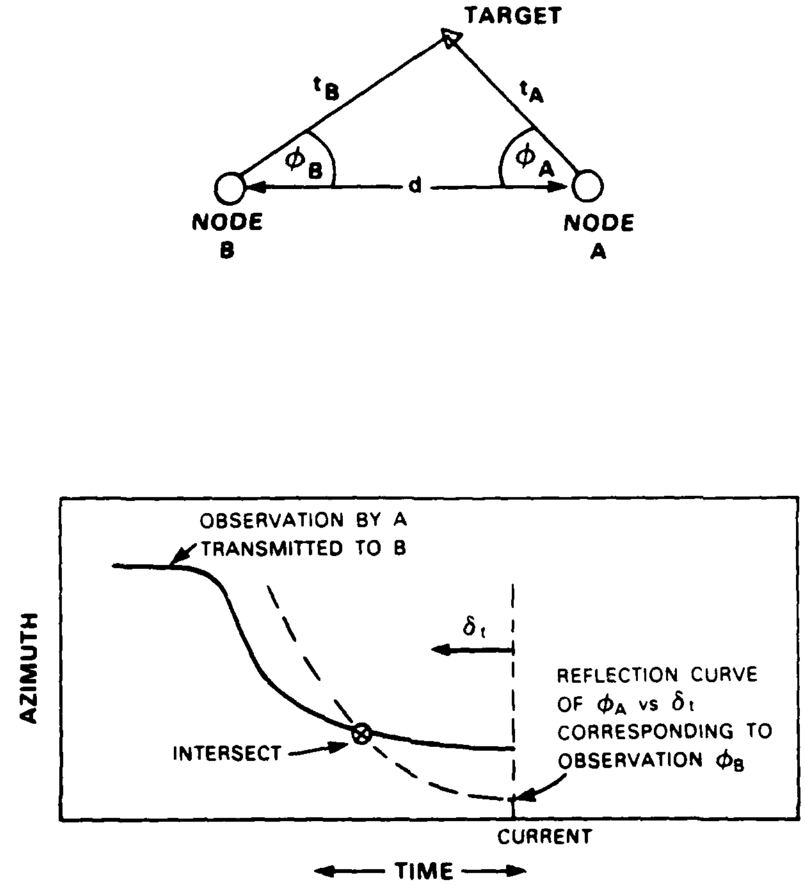

TARGET
tB
tA
φB
φA
d
NODE B
NODE A
OBSERVATION BY A TRANSMITTED TO B
AZIMUTH
δt
REFLECTION CURVE OF φA vs δt CORRESPONDING TO OBSERVATION φB
INTERSECT
CURRENT
TIME

Figure 6.13. Illustration of location using the reflection method. (a) Target and node geometry; (b) illustration of reflection method. See text for discussion.

observed the target at a time $\delta_t = t_B - t_A$ in the past. For all angles for which the lines from A and B intersect it can be shown that

$$\delta_t = \frac{d}{c}\,\frac{\sin(\phi_a) - \sin(\phi_B)}{\sin(\phi_a + \phi_B)}. \tag{6.49}$$

This equation can be used to translate the single observation ϕ_B into a curve of possible observations of ϕ_A versus δ_t as shown in the figure. Each point on this curve is a "reflection" of the measurement at B into a possible measurement at A; hence, the "reflection method." The intersection of the reflection curve with acoustic azimuth track from A gives the angle at A corresponding to the measurement at B. The target can now be located by triangulation. The time at which the target was at this position can be determined by subtracting the travel time to Node B from the observation time at Node B.

The reflection method can result in zero, one, or more intersections. Zero intersections simply means that the measurement at B cannot correspond to a previous measurement at A. Single intersections correspond to aircraft positions. When there are multiple intersections, one will correspond to the true position and the others to ghost targets that must be identified, perhaps by tracking them for a short time. In the case of multiple azimuth tracks and one new measurement there may be solutions for more than one of the azimuth tracks, a classic ghosting problem. All of these situations can be handled and do not constitute fatal difficulties.

But there were more serious problems that were not specific to the reflection method. They were fundamental to the deferred modeling approach and tracking by combining position estimates from pairs of nodes. These result in ineffective use of measurement data and substantial complication of the overall tracking process.

If there is a flight segment where only one node can detect an aircraft the measurements for that segment will be used ineffectively. Two nodes are required to locate the aircraft, so the measurements from the one node must be discarded without ever being used.

The problems, particularly with regard to complexity, are compounded when the step must be made from delayed two-node position tracks to current time tracking, which requires the use of dynamic models. Assume that Kalman filtering algorithms can be developed to process pairwise target position estimates and extrapolate them to the current time. For simplicity also assume that there is only a single pair of nodes. The propagation time to the further of the two nodes determines the time delay associated with the most recently available two-node position. Note that there is no mechanism for using the most recent and probably best measurements from the closer of the two nodes. There is a way out. It is to develop Kalman filters that directly process the more recent acoustic data from the closer node. That is, the system must be augmented by the addition of the track updating algorithms described in Section 6.2.4.

Now consider the same situation, but at the next measurement time. A more recent two-node position estimate can now be calculated using the model-free two-node location algorithm. This time the algorithm will make use of a more recent measurement from the closer of the two nodes. But the acoustic measurement from the closer node has already been used to update the current time track. Again, in theory, there is a solution. The current time track could be backed up, the information from the problematic azimuth measurement removed, the track updated with the new two-node location and, finally, made current by reintroducing the most recent azimuth measurements from the closer of the two nodes. It might even be possible to remove the information from the offending measurement without the need to remove and reintroduce the information from the more recent of the measurements. Alternatively note that the new two-node location is obtained only because there is a new azimuth measurement available from the more distant of the two nodes. Thus, the update with the new two-node position could be skipped altogether and the track could be updated directly using the new azimuth measurement. This is possible, but complicated when compared with using target motion models from the start and performing all location and tracking in terms of current time.

The complications are multiplied when there are more than two nodes participating in a target track. Since target dynamics are not modeled to calculate two-node locations, each pair of nodes will produce target locations for different past times. Moreover, locations calculated using different pairs of sensors are not statistically independent and their functional relationships are complicated. If they are combined without recognizing this functional dependence the resulting tracks will be suboptimal. The problem is similar to the local-foreign track combining problem treated in Section 6.2.5, but probably more difficult to solve. The added difficulty is because the basic data, two-node locations, are not independent for two different node pairs with a common node. Although it might be possible to develop algorithms to handle this situation, the solution would be at least as complex as the one of Section 6.2.5, and probably more so.

The growing complexity and suboptimality of the system quickly became clear as the focus of the research moved from two-node locations to current-time tracking with any number of nodes. This triggered a reexamination of the problem, resulting in the approach and algorithms presented in detail in Section 6.2. Our conclusion is that the value of not making assumptions about target dynamics does not compensate for the difficulties that result, especially since system users need current time information, which ultimately requires dynamic models.

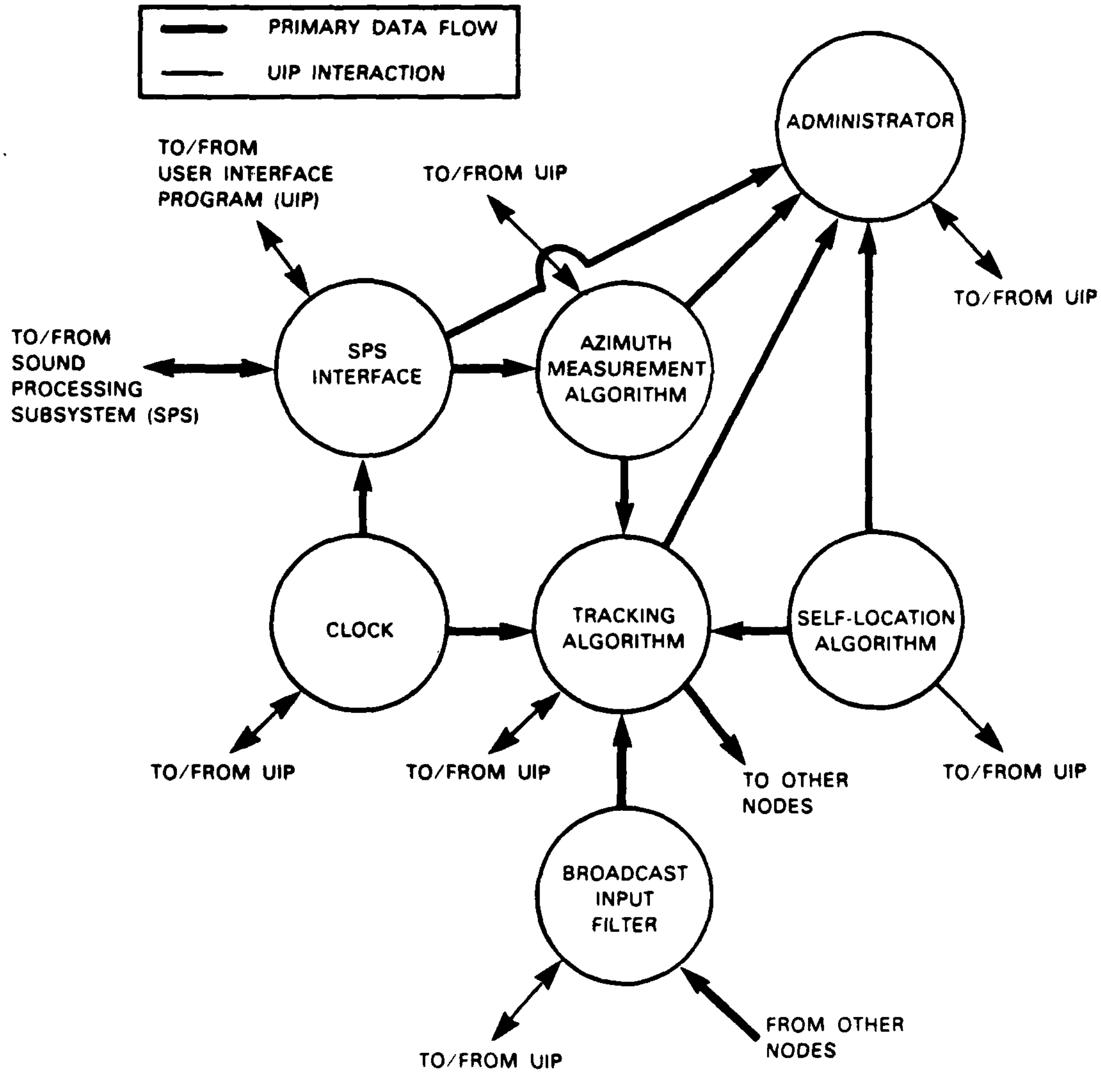
7. DISTRIBUTED SURVEILLANCE SOFTWARE

7.1. Introduction

The distributed surveillance software for the DSN test bed consists of three elements: (1) Tracking Software, (2) Network Control Software, and (3) Sensor Subsystem Software. The tracking software implements the algorithms described in Section 6.2. It is organized as many asynchronous modules interacting through message passing. This organization helped to control complexity and is well suited to the DSN SNC multiple processor system. Modules can be written with no concern for where they are executed. Interprocess communication is provided at execution time by the underlying NRTS message passing system, regardless of which processes are loaded onto which processor.

The control software provides users with the ability to start and control the tracking software, to collect data from nodes, and to display the collected data. This software consists primarily of a User Interface Program (UIP) and display software that run on a user workstation. The UIP interacts with nodes through special interface modules that are part of the tracking software. Many user workstations can be used simultaneously. Experiments have been conducted with up to three UIP systems in operation.

The following sections provide additional details about the design and implementation of the tracking software and the procedures for running an experiment using the UIP. Sections 7.2 through 7.4 concentrate upon the nodal software that implements tracking functions and interacts with the UIP. Section 7.2 describes the major functional elements and information flows. Section 7.3 explains the implementation of individual functions as a few cooperating processes. Section 7.4 covers the message structures used for interprocess communication. Section 7.5 describes how the UIP program is used to operate the test bed. It gives some feel for the extensive interactions required to operate the system and is intended to emphasize the need to automate the system to avoid operator errors and expedite the control and operation of experimental distributed networks. It should be noted that, although this point is valid for any distributed network, the needs are most stressing for an experimental system. An operational system would have more predetermined system parameters and provide less operator flexibility than an experimental system.



87882-26

Figure 7.1. Functional organization of tracking software.

There are some software peculiarities that reflect the fact the nodal tracking software was designed when several changes were planned for the UIP and SPS systems. It has not been possible to implement the planned changes and so compensating features have been added to the tracking software. System implementation as cooperating message-passing processes made the addition of these features a simple task requiring less effort than the implementation of the UIP and SPS changes. The planned changes and resulting peculiarities are reviewed here to eliminate subsequent confusion.

An initial version of the UIP was designed to provide nodal interaction on an ASCII, line-at-a-time basis. This was because the initial experimental hardware provided only RS-232 serial lines for communication between nodes and between the nodes and the UIP. With the introduction of Ethernets and radio communication, the plan was to make major changes in the UIP including changes to support binary messages. The tracking software was designed and implemented using binary messages for interprocess interactions. Unfortunately, time did not allow the reimplementation of the UIP, and processes were introduced into nodes to translate ASCII lines from the UIP into binary messages and vice versa.

The SPS interface, which uses a serial low-speed ASCII port for control and a parallel high-speed port for data, also introduces unnecessary complexity into the tracking software. Except for some discussion related to this interface, no details are provided about the software in the SPS or the TV subsystems. Both of these interact with the UIP and tracking processes but their implementation was driven by real-time processing and algorithmic considerations, not by their role as part of a distributed surveillance network.

Other software supporting the tracking software is described elsewhere; specifically, the nodal operating system is described in Section 3 and the communication software in Section 8.

7.2. Functional Elements

The main functional element of the surveillance software is the tracking algorithm, but there are several supporting players. Figure 7.1 illustrates the seven functional elements, the primary data flows linking them. and the data flows between the functional elements and the UIP program. The self-location algorithm supplies node locations needed by the tracker. Since the test bed does not include a self-location system, the self-location algorithm is a shell, but it provides the correct interface. The azimuth measurement algorithm translates SPS output information into the form required by the tracking algorithm, discarding less-credible detections in the process. The SPS interface regulates the rate at which measurements are provided to the rest of the system. It also mediates the setting of SPS algorithm parameters and the control of SPS actions. The broadcast input filter simulates network connectivity. The administrator logs performance data sent to it and handles other bookkeeping chores. The clock provides a time reference for the other elements.

The primary data flows between functional elements in the SNC and between SNCs in different nodes are shown as heavy lines in the figure. Interactions with the UIP are shown as light lines. The flows from the UIP fall into two categories: parameter values and commands. Parameter values are usually set at the beginning of an experiment. Commands may occur at any time. They return parameter values and other status information to the control program. They also change operating modes of the functional elements. Commands can be used to start and stop processing and control information flows. The outward flows to the UIP are performance data: typically acoustic azimuth measurements, acoustic azimuth tracks, or position tracks.

The following subsections supply more details about the functional elements and their interactions. Since there are many similarities for the different functional elements, only the tracking algorithm is covered in detail. Other elements are treated only to the extent that there are important differences or special features.

7.2.1. Tracking Algorithm

As Figure 7.1 shows, the tracking algorithm interacts with five other functional elements within the node (including receiving data from other nodes through the broadcast input filter). It transmits to other nodes via broadcast communication links and with the UIP program via point-to-point communication links. If the node in which the algorithm is running has an attached TV subsystem then the interaction with that TV subsystem (not shown in the figure) is, like the interactions with the UIP, by means of point-to-point communication links.

The tracking process is triggered by the arrival of new acoustic azimuth measurements from the acoustic measurement algorithm and also by the arrival of new TV azimuth measurements.

Messages from other nodes, passed along by the broadcast input filter, contain foreign position and azimuth tracks. These foreign tracks are queued up for processing during the acoustic action cycle triggered by the arrival of a message from the acoustic measurement algorithm. Broadcast messages for other nodes, point-to-point messages to the UIP or video sensor, and messages to the administrator also are generated during the cycle.

There are six phases in the acoustic action cycle:

(1) Extrapolate all tracks, including those received from other nodes, to the new measurement time.

(2) Combine local and foreign position tracks and initiate new tracks from azimuth track pairs.

(3) Associate acoustic azimuth measurements with position tracks, azimuth tracks, and previously unassociated acoustic azimuth measurements.

(4) Update position and azimuth tracks using associated measurements. (New azimuth tracks are created at this stage from pairs of new and previously unassociated acoustic azimuth measurements.)

(5) Broadcast azimuth track and position information to other nodcs if they satisfy the broadcast criteria.

(6) Send azimuth and position tracks to the UIP and/or to the administrator if this is enabled, and send position tracks to the video sensor if there is one.

All six phases are skipped if the time tag on the acoustic azimuth measurement indicates that it is too old. This is determined by comparing the time stamp on the measurement with the current time obtained from the clock process. This simple expedient is used to allow the processing to catch up to real time when a temporary overload occurs.

The video action cycle, triggered by the arrival of a set of video azimuth measurements, is similar but simpler. There are three phases:

(1) Extrapolate local position tracks to the new measurement time.

(2) Associate video azimuth measurements with local position tracks.

(3) Update local position tracks with associated measurements.

All three phases are skipped if the video azimuth measurement data are too old.

Interactions of the tracking algorithm with the UIP programs take the form of messages sent by the UIP to the algorithm or vice versa. These messages are processed as soon as practical but outside of the acoustic and TV action cycles. They fall into five categories: control, display, logging, cueing, and parameter commands.

The control commands are "enable," "disable," "reset," and "report control status." The first three control a software switch. When the switch is in the disabled or reset position, all messages other than those from the UIP are ignored. When it enters the reset state from any other state, all stored tracks (azimuth or position, local or foreign) and all previously unassociated azimuth measurements are discarded and all algorithm parameters are set to default values. The "report control status" command causes the control switch position to be reported to the UIP.

The display commands are "forward azimuth tracks," "hold azimuth tracks," "forward position tracks," "hold position tracks," and "report display status." The "forward" commands cause the local azimuth or position tracks to be sent to the UIP for display. The tracks are sent during the fifth stage of the acoustic azimuth measurement action cycle. The "hold" commands suppress the transmission of the tracks to the UIP. The "report" command informs the UIP if azimuth tracks, position tracks, or both are being forwarded.

The logging commands are "log azimuth tracks," "nolog azimuth tracks," "log position tracks," "nolog position tracks," and "report logging status." The first two commands set the position of an azimuth track logging switch, while the second two commands control a position track logging switch. A switch in the log position causes tracks to be sent to the administrator for logging. They are sent during the sixth stage of the acoustic azimuth measurement action cycle. The last command causes the switch positions to be reported to the UIP.

The cueing commands are "cue," "nocue," and "report cueing status." These have meaning only if a TV subsystem is logically interfaced to the tracker. The first two control the position of a switch. When it is in the cue state, local position tracks are sent to the video sensor during the sixth stage of the acoustic azimuth measurement action cycle. The last command causes the position cueing switch to be reported to the UIP.

Parameter commands take the form of a parameter name and a numeric value or the string "report parameters." Most commands cause the named parameter to be assigned the numeric value. One command is used for each parameter. The "report parameters" command causes all parameter values to be reported to the UIP.

As noted previously, the node locations that are required by the tracking algorithm are obtained from the self-location algorithm. One message from the self-location algorithm contains all node locations available to this node. As with the messages from the UIP, the node location messages are obtained and processed outside of the measurement-triggered action cycles.

7.2.2. Other Functional Elements

The other functional elements in Figure 7.1 interact with the UIP in a manner similar to that described in Section 7.2.1. The differences are primarily that the detailed responses to commands may be function specific, different functions recognize different parameters and not all functions recognize every command. For example "log" or "display" commands will cause different quantities to be logged or sent back to the UIP for display, depending upon the function involved. Another example is that all functions respond to the "report parameters" command, but the parameters that are reported are different for different functions. The "enable," "disable," "reset," and "report control status" control commands have similar effects for all functions, with the differences having to do with what data structures the functions must discard and reinitialize.

Following are brief descriptions of each of the functional elements in the system. Detailed descriptions of command responses are omitted because of their great similarity with the tracking function responses that have been described in Section 7.2.1.

The self-location algorithm provides the tracking algorithm with estimates of node locations. These locations include error covariances as well as node positions. The tracking algorithm must obtain a location for every node from which it receives sensor data that it will use for tracking purposes. Since the test bed does not include a self-location subsystem, the self-location function simply provides internally stored default locations or values supplied to it through the UIP. In the latter case, the self-location function forwards the locations to the tracking algorithm when it is first enabled and when locations are changed by the arrival of messages from the UIP. The node locations may also be sent to the UIP for display, to the administrator for logging, and/or to the video sensor for use in cueing.

The azimuth measurement algorithm receives data from the SPS interface and provides lists of azimuth measurements, including an estimated variance for each measurement, to the tracking algorithm. The estimated variance is calculated from signal power and noise power estimates provided by the SPS along with the azimuth measurement. The model for assigning variance assumes that the variance is inversely proportional to signal-to-noise ratio and array size, down to some minimum value.

The azimuth measurement algorithm also discards measurements to eliminate false detections. Detections are discarded for three reasons: if the sound pressure is too small, if the signal-to-noise ratio is too small, and if there are too many detections. In this last case, the detections passing the first two tests are ordered by signal-to-noise ratio and the list is truncated.

The SPS interface mediates all data flows between the SPS and the other functional elements of the SNC applications software and the UIP. It compensates for the limited flexibility of the SPS/SNC interface and the SPS software. It performs format conversions, helps to synchronize the SNC clock with the SPS, loads SPS parameters into the SPS, and starts and stops the SPS.

The synchronization of the SNC clock with the satellite clock in the SPS subsystem for live real-time experiments is an important operation in which the SPS interface plays an important role. When the SPS interface receives a "report time" command it sends a "show time" command to the SPS. This causes the SPS to report the satellite clock time back to the SPS interface. The SPS interface sends both the SPS time and the SNC time back to the UIP where the offset between them can be examined and commands sent to correct the SNC clock. This is explained further in Section 7.5.

The broadcast input filter provides the tracking algorithm with messages from other nodes, deleting messages which it should not receive. Its primary function is test-bed specific, not generic. The NRTS, Ethernet, and radio communication systems that support broadcast communications deliver every broadcast message to every node in the test bed. This includes messages transmitted by the receiving node. The filter discards the messages that are self-generated by the node and filters other messages to simulate partial communication connectivity between nodes. The connectivity can be changed during experiments by sending a command from the UIP to the filter. This mechanism has been used to experiment with communication and nodal failures in the network.

The administrator provides services to other functional elements of each node and to the user. These services include the logging of performance data on a floppy disk and nodal identification. In general, the administrator is the "home" for services that do not warrant the creation of a new functional element and do not conveniently fit anywhere else.

The clock controls the execution of tracking functions. It encapsulates and augments the services provided by the NRTS clock and provides an interface for the UIP. The clock time can be set, stopped, started, read, and waited upon. For live data experiments it is set at the start of the experiment to coincide with satellite clock time obtained from the SPS. Otherwise, experiment start time and the rate of the clock are arbitrary. The clocks are synchronized by interactions with the UIP as discussed in Section 7.5. The overall procedure is to set the times and start the clocks running, check the times, and apply offsets to individual nodes until network synchronization is judged to be satisfactory.

The synchronization of nodes in the test bed is a tedious and somewhat manual process, adequate for small networks with only rough synchronization requirements. The best way to synchronize nodes in a DSN remains an open question. In some situations it may be a by-product of the self-location process. This might be the case when self-location is based upon internodal distances obtained by measuring round-trip radio propagation times between nodes. Section 12 discusses self-location using internodal distances.

7.3. Process Groups

Each major functional element of the SNC applications software is implemented as a group of two or three cooperating processes. Each process has a main input message port and operates on a message-by-message basis. Each message is read and acted upon before the next one is read. The action is determined first by the message type contained in the message header and second by the message contents. A process may also have auxiliary input ports. These are read as part of activity triggered by the messages received on the main input port. Processes can have any number of output ports.

The process ports discussed here are distinct from NRTS ports, the interprocess communication mechanism provided by the NRTS run time system (Section 3.3). NRTS ports are the mechanism used to link process ports. When two process ports are linked, one NRTS port is the

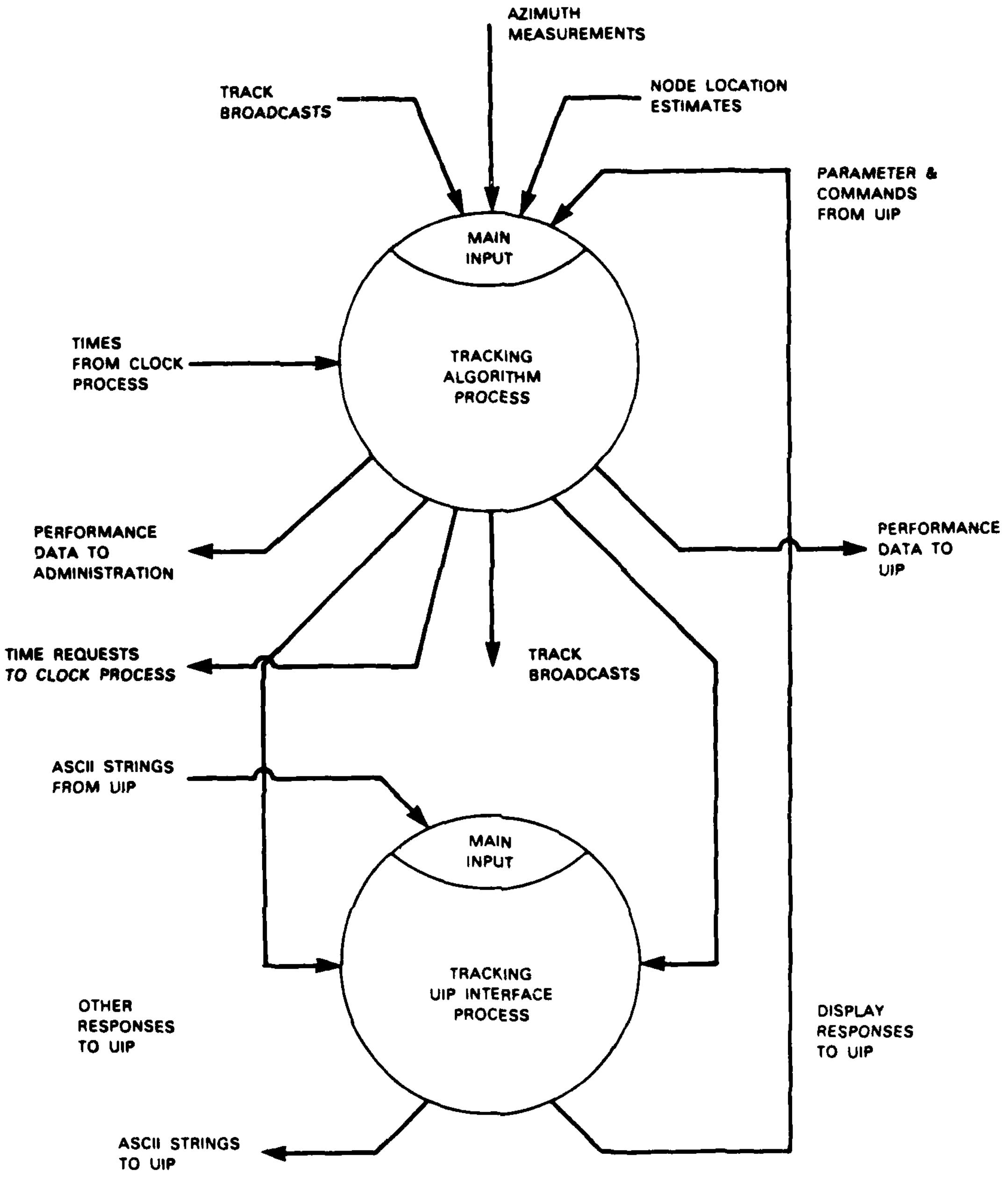


Figure 7.2. Tracking process group.

output of the "upstream" process and the other is the input of the "downstream" process. Many process output ports may be linked to a single input port, but a single output port may be linked to only one process input port. That is, there may be many writers, but only a single reader. Devices treat ports in the same way that processes treat ports. I/O devices can be conceptualized as system-supplied processes. Broadcast communication links between nodes, and point-to-point communication links between nodes and the UIP, are treated like device ports in the nodes where they originate and terminate.

Each functional element is implemented by one or more processes that perform the functional task and one additional process that serves as a UIP "impedance matcher." The former processes communicate among each other and with display programs using standard messages, which are binary data structures. UIP messages are ASCII strings and the UIP typically sends one parameter value or command at a time, while the functional elements expect all parameters to be communicated in one message. The UIP impedance-matching process performs the necessary format conversions, groups the parameter messages, and generates the acknowledgements required by the UIP.

The functional processes and the impedance matchers distinguish between display commands and all other commands. This reflects an unimplemented plan to split the UIP into two programs, one for each of these two different kinds of commands. It has no other significance.

In addition to the processes making up the process groups, an initialization process is included in each SNC processor (Section 3.3). The initialization process in processor P1 of each SNC creates all the NRTS ports used to link the process ports.

More detailed descriptions are given below for the tracking and the broadcast process groups. The former includes two, and the latter three processes. They are representative examples of process groups in the system. The tracking process group consists of one tracking algorithm process and one tracking UIP interface process as illustrated in Figure 7.2. The broadcast input filter process group consists of two major functional processes and one UIP interface process as illustrated in Figure 7.3.

The main input port of the tracking algorithm process receives messages from processes implementing the self-location algorithm, from the azimuth measurement algorithm, from the

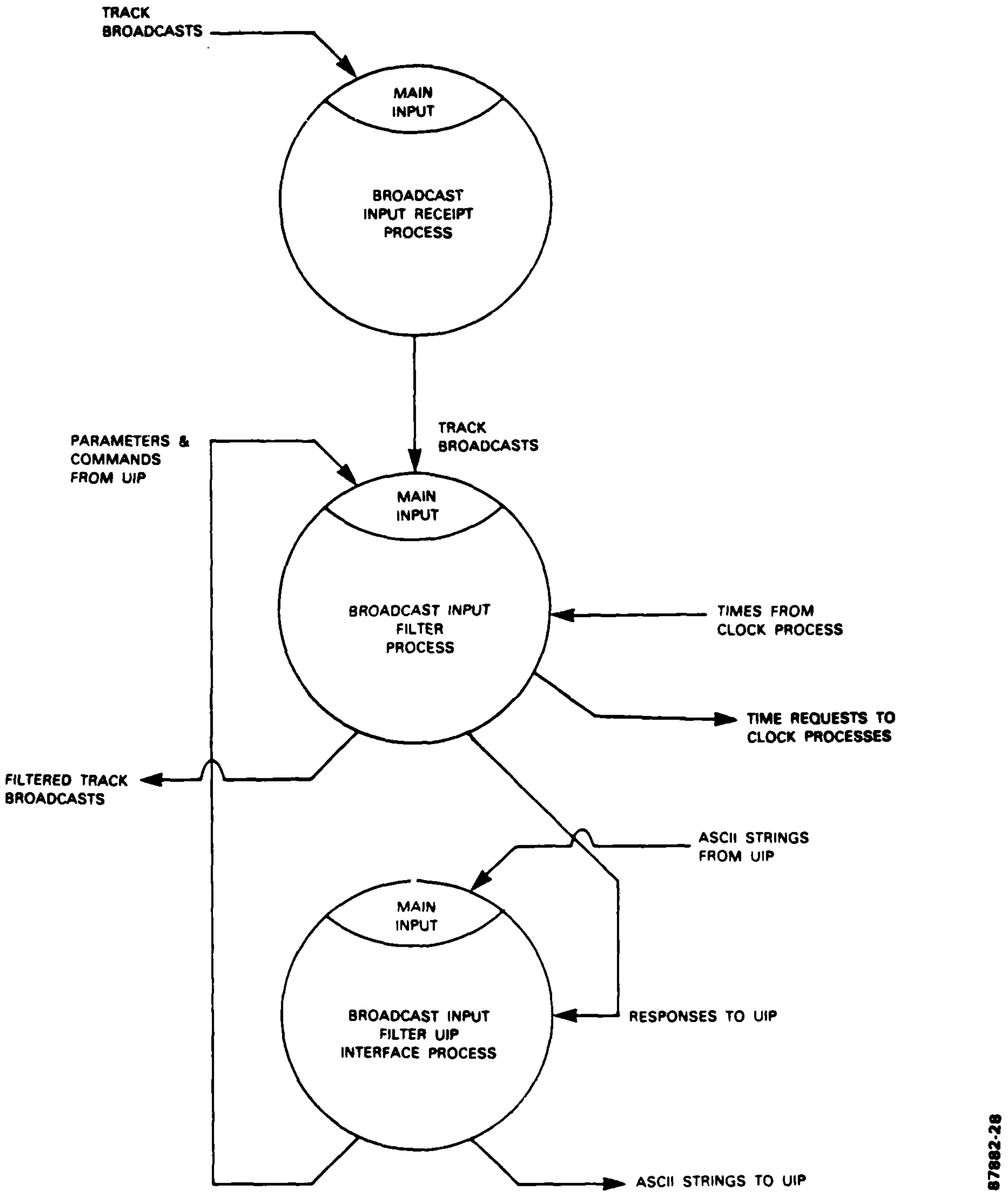

TRACK BROADCASTS
MAIN INPUT
BROADCAST INPUT RECEIPT PROCESS
TRACK BROADCASTS
PARAMETERS & COMMANDS FROM UIP
MAIN INPUT
BROADCAST INPUT FILTER PROCESS
TIMES FROM CLOCK PROCESS
TIME REQUESTS TO CLOCK PROCESSES
FILTERED TRACK BROADCASTS
ASCII STRINGS FROM UIP
MAIN INPUT
BROADCAST INPUT FILTER UIP INTERFACE PROCESS
RESPONSES TO UIP
ASCII STRINGS TO UIP
87882-28

Figure 7.3. Broadcast input filter process group.

broadcast input filter, and from the tracking UIP interface process. It has one auxiliary input port that receives time messages from the clock process. It has seven output ports. One handles broadcast messages, connecting the tracking algorithm process with the broadcast input filter in other nodes via broadcast communications links. A second sends time and waiting commands to the clock process. A third handles display command responses to the tracking UIP interface, and a fourth handles all other command responses. A fifth handles performance data being sent to the administrator, while a sixth handles sending the same data to the UIP for spooling and display. The seventh port sends cueing data to the vi^deo sensor if there is one.

The tracking UIP interface process has a main input port and two auxiliary input ports. The main input port is connected to the UIP via a point-to-point communications link and receives the UIP-originated ASCII string messages. The auxiliary input ports are linked to the command response output ports of the tracking algorithm process. Messages are read from these ports when responses are expected from the tracking process as a result of commands forwarded to it by the UIP interface process. The UIP interface process has two output ports. One handles commands sent to the main input port of the tracking algorithm process. The second output port is connected to the UIP, via a point-to-point communications link, sending translated responses to the UIP as ASCII strings.

The broadcast filter process group illustrated in Figure 7.3 consists of the broadcast input reception process, the broadcast input filter process, and the UIP interface process.

The broadcast input reception process is a repeater; the need for it is explained below. The main input port of the broadcast input filter process in the center of the figure receives broadcast messages from the broadcast input reception process as well as commands from the UIP interface process. It has one auxiliary input port that receives time messages from the clock. It has three output ports. One handles broadcast messages, connecting the broadcast input filter process with the tracking algorithm process. A second sends time requests to the clock process. The third handles command responses to the broadcast input filter UIP interface process.

The main input port of the broadcast input reception process receives broadcast messages from the tracking algorithm processes in other nodes via a broadcast communications link. It has no auxiliary input ports and only one output port. The incoming messages are addressed to a special pseudo device NRTS port which has the same name in each node. But the NRTS port that feeds the main input port of the broadcast input filter process has a unique name in each node. The purpose of the broadcast input reception process is simply to compensate for the name change and forward all messages.

The broadcast input filter UIP interface process has a main input port and one auxiliary input port. The main input port is connected to the UIP via a point-to-point communications link and receives the UIP-originated ASCII string messages. The auxiliary input port is linked to the command response output port of the broadcast input filter process. Messages are read from this port only when responses are expected to commands translated and forwarded by the broadcast input filter UIP interface process.

7.4. Messages

Most messages used in the SNC applications software are binary messages and are termed "standard messages." Other message communication includes nonstandard binary communication with the SPS, and ASCII communication with the UIP.

The number of different message types is large. There are over 50 standard binary message types and many other message formats are used to interact with the UIP. It is not feasible or necessary to describe all of these in detail. Instead, we describe the standard message format and provide a few examples of how specific binary message types are specified using standard C data structures. The use of standard C mechanisms for these complex messages very much simplified their manipulation. The standard message structure is a minimal one developed for the test bed. It is not proposed as a general solution for future DSN systems. Two specific message structures are described as representative examples: acoustic azimuth measurement messages, and position track messages. No further details are provided concerning the nonstandard SPS interface communication or the ASCII communication with the UIP. However, several examples of the UIP interaction messages will be found in Section 7.5.

Each standard binary message begins with a header. The header describes the message type, its length, its time of origin, and its node of origin. The header information allows standard format messages to be read, written, copied, and sorted by type, time of origin, and/or node of origin without reference to the contents of the message. The C definition of the standard header is

```
struct msg_hdr
        {
        unsigned short  msg_typ;        /* message type */
        unsigned short  msg_len;        /* message length */
        float           msg_tim;        /* message time */
        unsigned short  msg_nid;        /* message origin */
        };
```

The message type is a constant, unique for each distinct type of message. Macro definitions are used to make them symbolic in the application code. The message length is given in bytes and includes the message header. The message time is given in seconds of experiment time. It specifies the time at which the contents of the message are valid. A default value is used when time is not meaningful. The message origin is the integer part of a standard node identifier string, e.g., 1 for "n1." (Nodes are assigned names n1, n2, n3, etc.) It specifies the node that originated the message. Messages originating in a UIP program have an origin of 0.

Acoustic azimuth measurement messages are an example of a specific message type. Each message contains a list of acoustic azimuth measurements. Each measurement consists of an azimuth and an associated variance. A single message contains all the azimuth measurements for one measurement time from a single SPS system.

The C definition of the acoustic azimuth measurement message, including the header, is

```
struct dtae_msg
        {
        struct msg_hdr  dtae_hdr;               /* standard message header */
        unsigned short  dtae_cnt;               /* number of measurements */
        struct dtae_des dtae_maz[ MAX_MES ];        /* measurement descriptor */
        };

struct dtae_des {
        float           maz_mean;               /* measured azimuth */
        float           maz_covr;               /* measurement variance */
        };
```

The dtae_cnt variable specifies the number of measurement descriptors included in the message. It is bounded by an arbitrary value here symbolized by MAX_MES. The azimuth is in radians relative to north and is restricted to the range of $\pm\pi$. The variance is given in square radians.

The second example is the structure for the position information messages, which are sent by the tracking process to the broadcast input filter processes and to the administrator process. A single message contains a list of total and common position track pairs for a single time. Each pair is described by an identifier, a flag to indicate if the total and common tracks are equal, the estimated positions and velocities of the total track, the uncertainties of those estimates, the estimated positions and velocities based on the common track, and the uncertainties of those estimates. The message time, assigned by the originating node, is the time for which all estimates are valid.

The definition in C of the entire position information message data structure, including the header, is

```
struct tgpi_msg {
        struct msg_hdr  tgpi_hdr;                /* message header */
        unsigned short  tgpi_cnt;                         /* number of targets */
        struct tgpi_des  tgpi_inf[ MAX_INF ];  /* information descriptor */
        };
```

where the primary information containing structure is defined by

```
struct tgpi_des  {
        struct net_id     inf_id;          /* track identifier */
        unsigned short  inf_equ;                 /* equality indicator */
        struct dyn_stat  tot_meas;      /* total track mean dynamic state */
        struct dyn_covr tot_covr;       /* total dynamic state covariance */
        struct dyn_stat  com_mean;   /* common track mean dynamic state */
        struct dyn_covr com_covr;     /* common dynamic state covariance */
        };
```

The tgpi_cnt variable specifies the number of information descriptors included in the message, which is bounded by an arbitrary value here symbolized by MAX_INF.

The track identifier, inf_id, is the track identifier described in Section 6.2.8 dealing with track initiation and discussed again in Section 6.2.10 in the context of position track association. The equality indicator, inf_equ, is used to indicate when the total and common tracks are equal. Its primary use is to avoid duplicating some calculation when possible. Position and velocity coordinates are state variables for the dynamic models used by the tracking algorithm. The "dyn_stat" and "dyn-covr" terminology reflects that viewpoint. Variable names and data type names begin with a leading short string which indicates a logical grouping. Such naming conventions were used through the software to provide additional modularity and intelligibility to the system.

The following definitions of the position track identifier, dynamic state, and dynamic covariance data structures are included here for completeness:

```
struct net_id
        {
        unsigned short  nd1_nid;                /* TI node 1 */
        unsigned short  nd2_nid;                /* TI node 2 */
        unsigned short  nd1_id;         /* TI azimuth track 1 */
        unsigned short  nd2_id;         /* TI azimuth track 2 */
        };


struct dyn_stat
        {
        float    dyn_pn;                /* north position */
        float    dyn_pe;        /* east position */
        float    dyn_vn;                /* north velocity */
        float    dyn_ve;        /* east velocity */
        };


struct dyn_covr
        {
        float    dyn_pnpn;      /* north position variance */
        float    dyn_pnpe;      /* north/east position covariance */
        float    dyn_pepe;      /* east position variance */
        float    dyn_pnvn;      /* north position/velocity covariance */
        float    dyn_pevn;      /* east position/north velocity covariance */
        float    dyn_vnvn;      /* north velocity variance */
        float    dyn_pnve;      /* north position/east velocity covariance */
        float    dyn_peve;      /* east position/velocity covariance */
        float    dyn_vnve;      /* north/east velocity covariance */
        float    dyn_veve;      /* east velocity variance */
        };
```

The node identifiers are the integer parts of the relevant node names. Positions are in meters relative to an arbitrary reference point, and velocities are in meters per second. The position coordinate (co) variances are given in square meters, and the velocity coordinate covariances are given in squared meters per second.

7.5. Using the UIP to Run an Experiment

The UIP is the User Interface Program employed to operate the test bed and control experiments. Several UIP programs, operating on different user workstations, can be operated simultaneously. The onus for avoiding destructive conflicts is on the users.

The use of a UIP to operate the network is described here from the viewpoint of an experimenter using the test bed. This is intended to provide a feel for the kinds of tasks that are involved, to show examples of the ASCII messages used for communication between the UIP and the nodes, and to emphasize the need to automate tasks. All commands except those beginning with a ":" are messages to a process in a node.

Several commands are needed to initialize the UIP itself. The following is a typical list of such commands for a single node experiment using node n5:

```
    :netin ethernet
    :open n5 ethernet 5
    :spool spool
    :log log
```

The first of these connects the UIP to the Ethernet to receive messages. The second associates node n5 with an Ethernet address (5). The third establishes a spool file for holding display data sent by the nodes to the UIP, and the fourth establishes a log file holding a copy of all ASCII responses received from the nodes. Repeating the second line for different node numbers adds additional connections. These commands are usually placed in a file called "opens" and the entire file of commands is issued by typing

```
    :file opens
```

The ability to issue a sequence of commands from a file is a general feature of the system. As will become clear below, without this feature it would be virtually impossible to conduct an experiment. The number of commands and the propensity of people to make errors conspire to make this true. However, in what follows this encapsulation within command files is stripped off to expose procedures in more functional detail than would be possible otherwise.

If the nodes are already turned on and operational, the user can proceed with an experiment using only the UIP to control and operate the system. If not, some additional steps are required to power up the nodes, load software, and start the system. If the nodes are not powered up, then someone must visit each node and switch on the power. Software loading and system startup is also usually done by an operator at the node. The software is loaded from floppy disks. If there are serial lines connecting the nodes and the UIP processor they can also be used for software loading and startup but this is usually slower and less convenient than using the floppy disks.

Using the Ethernet for downloading could improve the situation. But sequentially loading software for 10 or more nodes might still be time consuming unless the data rate is very high. The problem is compounded because reloading a processor is an all or nothing process. That is, if any part of the code is changed it must all be reloaded. Facilities for remote loading, starting, and modifying of nodal software should be improved for any future experimental system. Future operational systems would also require different remote methods to load, start, and restart nodes. This would extend to include remote testing of hardware and software.

If the nodal software was previously loaded, the nodes can be reinitialized using the reset command. Each process must be individually reset. Thus, the command sequence

```
/n5/aduip reset
/n5/truip reset
/n5/amuip reset
/n5/bifuip reset
/n5/sluip reset
/n5/spsuip reset
/n5/clkuip reset
```

is required to reset the tracking software in a single node, node n5 in this case. The processes "aduip",...,"ckluip" are the UIP interface processes for the seven process groups that implement administration, tracking, azimuth measurement, broadcast, self-location, SPS interface, and clock command functions. The user and the UIP deal only with the UIP interface processes. Those interface processes deal with other elements of the process group as required to accomplish a task.

The form of the reset command is the same as that of all other node commands handled by the UIP. An initial character string is followed by one or more additional strings terminated by a final end-of-line character. The first string identifies a network process and the remaining strings are a message directed to that process. The command destination string is in the form:

```
/<node_name>/<process_name>
```

where the <process_name> is the UIP interface process name in a process group.

A single experiment may require that 50 or more commands be transmitted to a single node to set up the experiment. These included commands to establish: what data will be logged on the nodal disk systems; what data will be communicated back to the UIP for display; if the node will cue a TV sensor; what parameter values should be used for tracking; what is the network communication connectivity; what nodal locations are known to which nodes; what are the parameters for the azimuth measurement process; and what parameters are to be used by the acoustic signal processing subsystem. Some examples of commands would be: "/n5/truip forward position tracks" to forward position tracks from the tracker to the UIP for display; "/n5/truip log position tracks" to cause the tracker to send position tracks to the administration process for logging on the node disk; and "/n5/truip az_diff 5.0" to set one of the more than 25 parameters that are recognized by the tracker. The number of commands multiplies with the number of nodes in the experiment.

The test bed can be operated in different modes. The mode is determined by commands sent to the "spsuip" processes in the nodes. The options are to employ the SPS in real time with time stamps provided from the satellite clock in the SPS, to use the SPS in real time but process previously recorded acoustic data read from the SPS digital tape system, or to not use the SPS at all and to conduct a real-time experiment using azimuth measurement data stored on the nodal floppy disk system. Except during real-time operation with the satellite clock it is possible to operate the system at a fixed fraction of real time.

The most common usage mode has been to operate without the physical SPS systems and to employ measurement data prestored on floppy disks. This is convenient for controlled experimentation with algorithms and for system tuning experiments that require repeatability of data inputs. A typical command sequence required to operate in this mode is:

```
/n5/spsuip sps_over 1.5
/n5/spsuip config floppy
/n5/spsuip ex-start 0.0
/n5/spsuip ex_stop 300.0
/n5/spsuip part_disk 0
/n5/spsuip part_inx 1
```

The first parameter, "sps_over", is a processing slack time. If data read from the SPS, in this case the floppy disk, have a time stamp that is more than the specified number of seconds late relative to the SNC clock, the data will be discarded. The second parameter, "config", specifies that the SPS is to be simulated, i.e., data are to read from the floppy. The third parameter, "ex_start", is the experiment start time in seconds after midnight. Any data on the floppy disk before the start time are skipped. The fourth parameter, "ex_stop", is the experiment stop time in seconds after midnight. Data after the stop time are also ignored. The fifth parameter, "part_dsk", is the disk drive to be used (0 or 1). The sixth parameter, "part_inx", is the disk partition index to be read. From the user's viewpoint, a floppy disk partition is equivalent to a file.

All experiments with the test bed are driven by the real-time clocks in the SNCs. These are set at the start of each experiment and run free for the duration. Fortunately, the required degree of synchrony between nodes is only on the order of a few tenths of a second and the duration of typical experiments ranges from only a few hundred seconds to a few hours. Loss of synchrony has not been a serious problem and more sophisticated network synchronization mechanisms have not been required. An operational system or experimental system designed for long duration experiments would require additional capabilities to maintain synchrony.

Once all other system parameters are set, it remains to set the clocks and start the experiment. The clocks are set by commands of the form

```
/n5/clkuip set time 0.0 1.0
```

where the first number is the time and the second number specifies the rate at which it should tick after it is started. The details after this depend upon whether the nodes will be synchronized to satellite clocks in the SPS subsystems or not.

The startup procedure is simplest when the experiment does not need to be synchronized to the SPS satellite clocks. The procedure in this case is to first start all the other processes and then, after a pause, start the clocks. Thus, the command sequence for a two-node experiment would be:

```
/n5/aduip enable
/n5/truip enable
/n5/bifuip enable
/n5/sluip enable
/n5/amuip enable
/n5/spsuip enable
/n6/aduip enable
/n6/truip enable
/n6/bifuip enable
/n6/sluip enable
/n6/amuip enable
/n6/spsuip enable
```

followed by a pause and

```
/n5/clkuip enable
/n6/clkuip enable
```

The pause is required to clear all message traffic before the clock enable messages are sent. Clock enable messages, which set the clocks ticking, are sent as nearly simultaneously as possible. The degree to which the network is synchronized is determined by the simultaneity of the clock enable messages. The user can verify synchronization by sending "report params" messages in quick succession to each of the clkuip processes.

The synchronization process is more tedious when the network must be synchronized with the satellite clocks. Also, due to some internal peculiarities of the SPS subsystems, synchronization is done before other processes are enabled. The satellite clock in one node is usually selected as the network master. The procedure is to enable all of the SNC clocks, synchronize the master clock with the SNC clock in the same node, and then synchronize all the other SNC clocks with the first one, one at a time. Since all satellite clocks are synchronized with each other it would also be possible, for those nodes containing satellite clocks, to synchronize the SNC clocks with the satellite clocks and then synchronize other nodes.

Suppose node n5 contains the master clock and "clkuip enable" commands have been sent to start the SNC clocks. The synchronization of the master clock with the first SNC clock is done as follows. The command "/n5/spsuip report time" will provide the user with satellite and the SNC clock times for node 5. Based upon the difference between those times the user will issue an "offset time" command to adjust the SNC clock. This is done repeatedly to check and adjust synchrony until it is deemed satisfactory.

Synchronization of other nodes then proceeds one at a time. For example, suppose node 6 is to be synchronized with node 5. The command sequence

```
/n5/clkuip report params
/n6/clkuip report params
/n5/clkuip report params
```

causes the SNC time to be reported twice from node 5 and once from node 6. Based upon the differences in reported times, the user will issue an appropriate "offset time" command to node 6 to bring it into synchrony. This sequence is repeated until the user is satisfied and is then repeated for all other nodes. The peculiar form of this interaction, requiring two reports from one node, is the result of a limitation in the communication system, which should (but does not) attach node identifiers to messages sent to the UIP. (Note that, as described in Section 7.2.4, standard binary messages do contain such information but are not used by the UIP. The lack of a message source identifier at the UIP level is a limitation resulting directly from as yet unexecuted plans to replace the UIP and adopt standard binary messages for all interprocess communication.) Once the network is synchronized, commands are sent to enable all the other processes and start the actual experiment.

The above procedures for network synchronization are at best tolerable and usable for limited experimentation with a small network. The details have been given to make this point perfectly clear, to emphasize that distributed network synchronization is important and that better methods are needed.

The normal way to complete an experiment is to reset all the processes in preparation for the next experiment. These reset commands also assure the validity of data logged on nodal disks.

8. COMMUNICATION SUBSYSTEM

8.1. Introduction

Figure 8.1 shows the primary communication elements of the test bed. These are: (1) Ethernets, (2) microwave radios to link the Ethernets, and (3) low speed serial lines linking software development computers to test-bed nodes. The Ethernets and Microwave radios provide the primary support for distributed tracking experiments. The serial links, augmented by human operators at individual nodes, are used for convenience during system startup. Serial links are available only to nodes located within the main DSN laboratory area.

The Ethernets and Microwave radios are used to completely link all nodes into a single virtual Ethernet. This logical Ethernet supports point-to-point messages between any pair of nodes and broadcast messages from any node to all others. These communication services, point-to-point and broadcast, are the essential communication services that are used for tracking and for controlling and monitoring experiments.

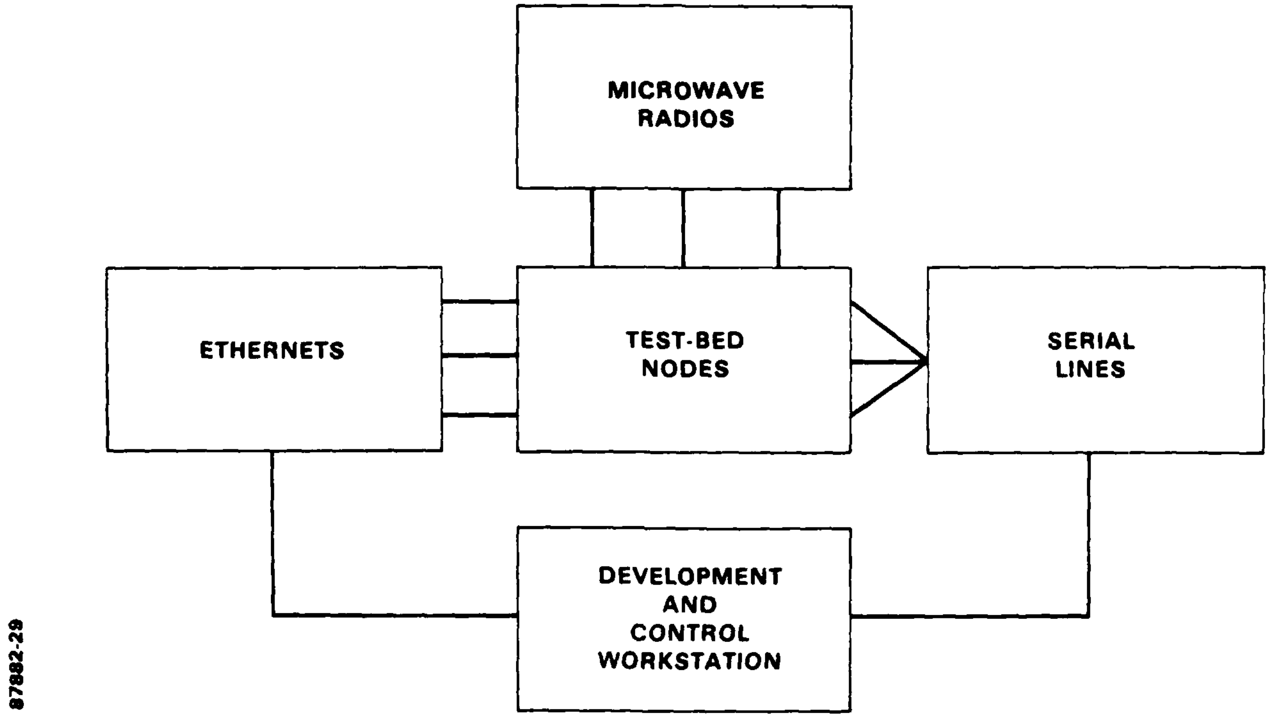


Figure 8.1. Test-bed communication subsystems.

The functions of the serial lines were discussed in Section 7.5. It is important to reemphasize that neither special serial lines nor operators located at nodes would be used for routine operation of larger scale experimental systems or an operational system. Such systems should be designed to use the primary DSN communication system to control remote node startup and test functions that are now accomplished using serial lines and human operators. This would not result in significant additional communication but would require specialized messages and node hardware. Among other things, it would require a design that provides for nodes to be deployed in a low-power state, capable of receiving and acting upon startup messages. This standby condition would be implemented with minimum complexity and maximum reliability.

Neither Ethernets nor radio links were part of the test bed in its early stages. The initial experiments utilized 9600-baud serial lines in a star configuration as communication links. The serial lines in the test bed are remnants of that initial star configuration. Although the lines no longer provide primary point-to-point and broadcast services their initial use has resulted in some communication system peculiarities, notably the use of ASCII rather than binary messages for communication between the nodes and the experiment control computer. This has persisted through the final implementation.

Sections 8.2 and 8.3 describe the communication hardware and the Ethernet and radio communication software. Section 8.4 briefly covers another topic; the development of an experimental digital radio system.

8.2. Communication Hardware

The primary communication medium for nodes located within the main DSN laboratory is an Ethernet. In addition to the nodes, the hosts on the network include a VAX 11/780 computer and two UNIX workstations for software development and experiment control. Communication between the laboratory and remotely deployed mobile nodes is by means of microwave radios. The remote nodes are installed in vehicles. Each vehicle contains two or more SNC systems and a short local Ethernet. The laboratory-based Ethernet and all of the remote Ethernets are physically separate. One of the SNC systems on the laboratory Ethernet and one on each of the remote Ethernets serve as communication gateway nodes. The gateways transmit messages between the Ethernets and the radio system in such a way that all elements, those in the laboratory and those at remote sites, appear to be on a single logical Ethernet.

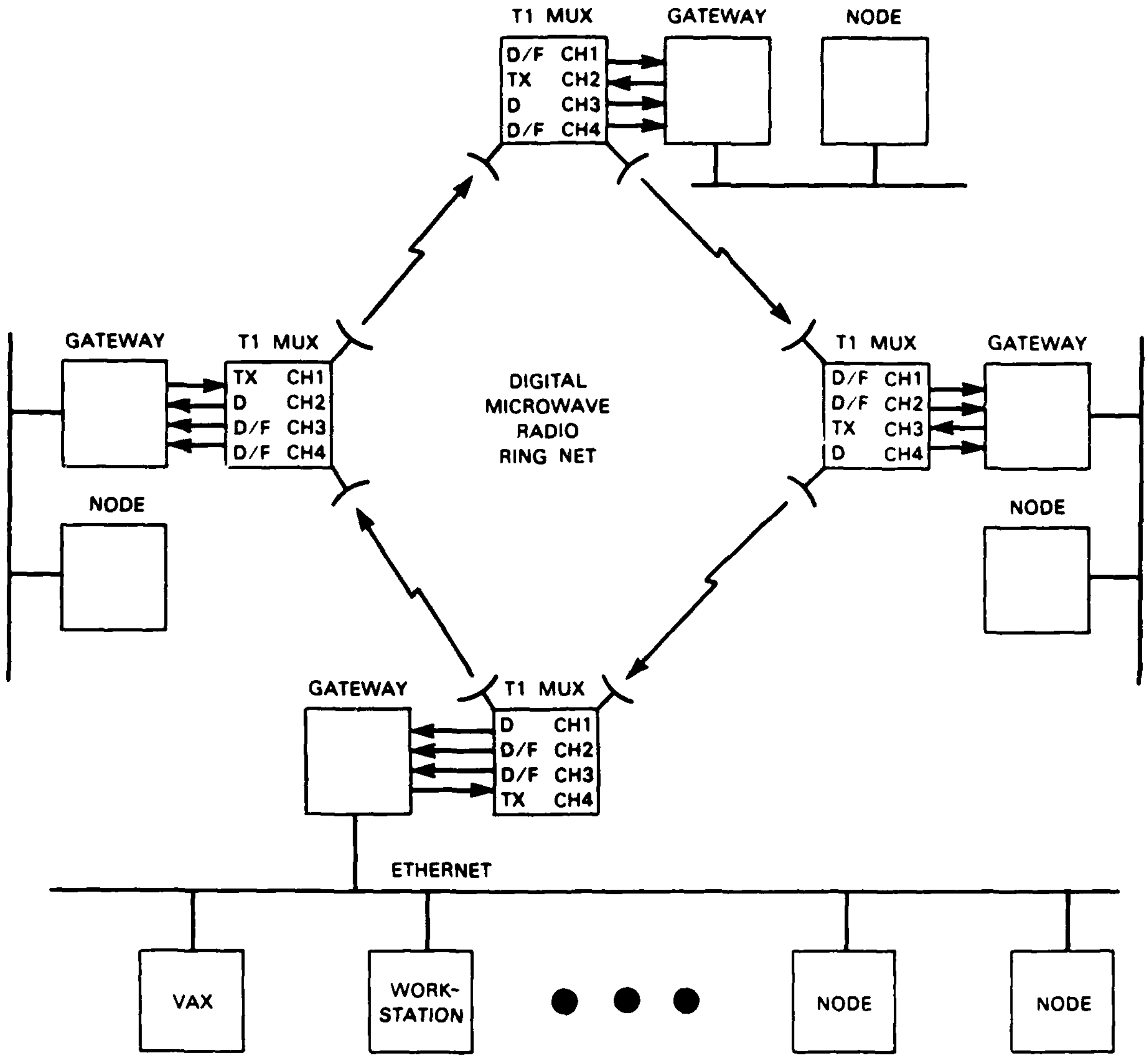
There are no restrictions to the type or number of hosts that can be attached to any of the Ethernets. Thus, one of the UNIX workstations can be physically moved to one of the remote sites and used from that location to control experiments in the same way that it would be used if located in the main DSN laboratory. TV subsystems can be attached to any of the Ethernets. The usual configuration is to attach one TV subsystem to the laboratory Ethernet and one to the Ethernet in a mobile unit. All mobile systems normally contain one SNC that is used for tracking and also serves as an interface to the acoustic subsystems.

The gateway SNC systems that interconnect the Ethernets are skeletal systems containing only a single processor board. They have hardware interfaces to microwave radio equipment and to the local Ethernet. Gateway nodes reformat messages received on the Ethernet and retransmit them to other parts of the test bed using the microwave radio. They also retransmit messages received from the microwave radio to the other SNCs attached to the Ethernet. The gateways utilize the standard NRTS operating system but, to avoid computational overload, do not run any of the tracking or video software. The gateway SNCs are dedicated to the communication task. The gateway and tracking functions have been kept in separate SNCs to ensure the availability of adequate resources.

All Ethernet hardware is standard commercial equipment. The interface between the Ethernet and the SNC multibus is an Execelan Ethernet Interface Board. This board interrupts the NRTS system whenever it has a message to deliver. Whenever NRTS has a message to send it passes parameters to the interface to accomplish the transmission. The interface board implements basic Ethernet protocols and relieves some load from the SNC processor. Specifically, in all non-gateway nodes, only messages requested by the attached SNC are passed on to NRTS by the interface. These are the point-to-point messages addressed to the specific SNC and broadcast messages that are received by all the SNCs.

The microwave radios that interconnect sites are Loral Terracom TCM-6 series radios operating in the 7.125 to 8.4-GHz band with a T1 carrier and a 1.544-Mb/s bipolar data stream. Radios are installed in three mobile units and on the roof of the main DSN laboratory.

There are two levels to the interface between the SNC and the microwave radio. The first level is the multibus board, which is the interface seen by NRTS. This board, furnished by SBE, Inc. provides four RS-449 56/64-kbits synchronous simplex communication channels. Three of



T1	MUX	UP TP 24 CHANNELS, TIME DIVISION MULTIPLEXED
TX	CHn	TRANSMIT ON CHANNEL n
D/F	CHn	DROP AND FORWARD CHANNEL n
D	CHn	DROP BUT DO NOT FORWARD CHANNEL n

87882-30

Figure 8.2. Microwave ring used to interconnect test-bed ethernets.

these are for output from NRTS, and one for input. This interface board was customized using ROM resident software to minimize the communication load on NRTS. All communication between NRTS and the board is in terms of messages, whereas characters are more natural for the basic synchronous channel. There is no provision for error checking or retransmission of damaged or lost messages. The interface board attaches to a DS-1 Format digital time division multiplexor that can multiplex up to 24 channels onto a single T1 carrier. This equipment, manufactured by Tau-tron Inc., is used to multiplex the four 56/64-kbit channels onto the T1 carrier. Only 4 of the available 24 channels are utilized.

This hardware is used to form a ring of up to four sites interconnected by microwave radios as illustrated in Figure 8.2. Each site contains a radio, a multiplexor with an interface board, a gateway, and a local Ethernet. The four sites consists of three mobile systems and the laboratory system. The number of sites can be reduced to three or two and need not include the laboratory site. The radio hardware and multiplexor are set up so that the transmission from any site is received by all three of the other sites. Each site has a separate receive channel for each of the other sites. The system routes a transmission from any site around the ring to the other three sites that receive it on a D or D/F channel, depending upon whether the site is the last of the three or not. The interface board operates in a so-called one-way drop and insert mode. For two of its receiving channels it forwards the message on the same channel. Messages on the third receiving channel are dropped and not forwarded, because if forwarded they would be sent to the node which originated them.

The radio link between any pair of nodes is unidirectional. Each of the up-to-four sites on the radio ring contains a single radio transmitter, a receiver, a transmit antenna, and a receive antenna. This is different from the way this equipment is normally employed for multihop communication. In standard applications each direct link between a pair of nodes is bidirectional, requiring additional receivers, transmitters, and antennas. By arranging the system in a ring and utilizing unidirectional links, we were able to reduce the amount of equipment by almost a factor of two.

The communication hardware can be scaled up to support more nodes, to a limit of about 20. This would require acquisition of radios and interfaces for each node and some upgrading of the existing interfaces to handle the additional channels. A small amount of additional NRTS software would also be required to handle the additional channels. However, the NRTS message processing load would increase substantially and probably limit the system to substantially smaller sizes, on the order of 10 nodes or less. This processing load limitation is not fundamental but is due to the details of our specific hardware and software implementation.

8.3. Distributed Surveillance Communication Software

The nodal operating system, NRTS, and its most basic I/O and communication services are described in Section 3.3. That section describes the basic NRTS interprocess message passing system based upon "ports" that are functionally similar to UNIX ports. As described in Section 3.3 the operating system provides Point-To-Point (PTP) and broadcast message types, which are the internodal communication mechanisms used throughout. The present section adds to that description with emphasis upon aspects related specifically to providing internodal communication with Ethernets and radios. In addition, special gateway software is described. That software uses the NRTS features and software drivers to actually accomplish the interconnection of the network. A simple user-level program in gateway nodes helps to identify possible problems with the overall communication system.

One requirement for the communication software was that user software should run without modification on any node, regardless of the communication media used to transmit messages between nodes. This requires that the communication system know about the physical communication media and how to reach any node from any other node. These multiple-media, multiple-network routing problems are very difficult in general. Fortunately, it has been possible to devise specialized solutions specifically for the test bed.

Not all nodes contain the same hardware components and devices. Some, but not all, SNC systems contain radio interfaces, Signal Processing Subsystem (SPS) interfaces, special TV subsystem boards, and multiple MC68000 processor boards. To avoid the confusion of many different, node-specific operating system configurations, NRTS performs a complete system generation at load time. All possible devices are checked to ascertain if they are present and functioning correctly. Bus errors generated by reference to nonexistent I/O devices are intercepted, avoiding a system crash when hardware is missing or malfunctioning. Missing devices are reported and the

I/O devices that are present are assigned an appropriate interrupt level and I/O transfer vectors are set up. I/O queues and routines to monitor the queues are established for each existing I/O device. Finally NRTS transfers control to the user level software.

The second, and much more difficult general problem, is how to route messages to their intended destinations. Two simple, but test-bed specific, message handling rules were programmed into the NRTS message system to accomplish this. These rules suffice because there are only two transmission media, Ethernets and microwave radios, and because an additional configuration restriction is enforced. The configuration restriction is that no more than one of the SNCs on an Ethernet can also be attached to a microwave radio. The reason for this restriction will be made clear shortly.

The first rule is that a message received on the Ethernet is retransmitted by the radio and vice versa. The second rule is that messages originating in a node are transmitted over all attached communication media: the Ethernet, the radio, or both. These rules are applied uniformly to both PTP and broadcast messages without checking the message type. No attempt is made to ascertain how to reach any particular node. As a result some of the transmissions of PTP messages may be unnecessary, but this approach results in a trivially simple system which is adequate for the existing test bed. The constraint that no single Ethernet have more than one attached radio avoids infinite message retransmissions. Without this constraint a message sent from one radio would appear at another on the same Ethernet and the system would become unstable.

The software interfaces between NRTS and the radio and between NRTS and the Ethernet are important because they determine the computational load placed upon NRTS for message handling. This is particularly important for gateway nodes containing both both radio and Ethernet interfaces.

The interface to the radio, consisting of one input and three outputs, was designed to be as simple as possible. The hardware interrupts the NRTS operating system in three cases. There is an initial interrupt to signal NRTS that the interface is initialized. Thereafter, NRTS receives an interrupt whenever a message is received or when a requested transmit is finished. The interface board provides ten 1024-word buffers in each direction for each of the four channels. This internal buffering provides a message at a time interface with NRTS, although the output of the board is a character at a time.

Communication between NRTS and the radio interface board is through a separate four-item data structure for each channel. The items are a channel status byte, a transmit/receive flag, a byte count, and a pointer to a NRTS data buffer. The interface board continuously polls the four status bytes. When a zero value is noted and the flag is TRANSMIT, the interface board proceeds to DMA the data into one of the on-board buffers. If no on-board buffer is available, the request is honored when a buffer becomes free. As soon as the data are moved, the appropriate status byte is set and an interrupt to NRTS is generated. When the interface board detects a zero status byte and the transmit/receive flag is set to RECEIVE, a message will be copied into the buffer provided by NRTS if a message is waiting. Otherwise, the transfer will take place as soon as one is available. The number of bytes in the message is entered into the count field, the status bits are set, and an interrupt to NRTS is generated.

The software interface to the Ethernet interface board is a conceptually similar, message at a time, interface although there is a single interface channel rather than four. The interface board allows the host computer to specify which incoming messages should be delivered to the host. All other messages on the Ethernet are discarded without alerting the host. NRTS normally uses this to minimize the message handling load on the SNC processors. But gateway nodes relay messages to other nodes and therefore must handle all of the Ethernet messages. This "promiscuous" mode is provided by the interface board but it requires that a gateway node handle a very large message volume. This load can become large enough to interfere with tracking functions. Therefore, it was decided to perform the gateway functions in separate nodes and to include radio software only in the nodes containing microwave radios. Because nodes other than gateway nodes have no gateway functions, their Ethernet interface boards can be set to transfer only a subset of all of the Ethernet messages to NRTS. The messages passed on to NRTS are broadcast messages and PTP messages addressed specifically to the node. The separate gateway nodes, by handling all message retransmission, leave more resources available for tracking and TV functions in the nodes that perform those functions. These are the only operating system differences between gateway and non-gateway nodes.

Since gateway nodes do not execute tracking or TV software there are some spare resources available that can be used for simple system monitoring. A user level program is run which prints out the traffic it has encountered every 2 s. This program will respond to a query from a UIP and provide traffic statistics to the user of the UIP. This feature allow the experimenter to ascertain if the gateway is running correctly.

8.4. Communication Network Technology Research

The initial DSN plan included integration of experimental packet radios into the DSN test bed to provide the internodal communication services that are now supplied by Ethernets and commercial microwave radios. The objective was to utilize advance radios being developed by the Communication Network Technology (CNT) project sponsored by DARPA. These radios were designed to experiment with and demonstrate many advanced features and capabilities [25]. Only a small number of the features were essential for the DSN application, but it was clear that the DSN test bed could provide a basis for experimentation with the radios.

The CNT radio is a pseudonoise spread-spectrum packet radio. The spreading code is changed for each bit in each message. The code for the first bit of each message can also be changed on a message-by-message basis. The primary data rates for the radios are 90.3-kb/s and 1.45-Mb/s. In both cases the first several bits of the message are synchronization and header bits transmitted at the 90.3-kb/s rate. The header indicates whether the main part of the message is at the 1.45-Mb/s rate so the receiver can switch to the higher rate when appropriate. At each of the two basic signaling rates the radio provides for several levels of error checking and correction encoding, resulting in several usable lower communication rates.

Another feature is that the radio can very precisely time stamp incoming messages and control the time of outgoing messages. Pairs of radios can use this capability to measure the distance between the two radios and to determine their clock offsets. Measurements from many node pairs can be used to determine the relative positions of the nodes and synchronize clocks. This is discussed more fully in Section 12.

These features, basic broadcasts and message timing, are the radio characteristics of primary interest for the DSN application. However, the radios incorporated many technical innovations to implement these capabilities and have many other special features. For example the high data rate is higher than that provided by any other packet radio. Adaptive notch filters are included in the radio front end to eliminate narrowband jammers. Three different options are provided to measure message arrival time. It can be measured without multipath corrections or, using two different methods of multipath processing, small direct-path precursors can be detected in the presence of strong multipath signals. Transmit power and a number of receiver thresholds are adjustable. A special mode is provided in which the body of a message is transmitted in a very high-gain low-data-rate integrating mode although the synchronization bits and header are transmitted at the 90.3-kb/s rate. The radio also can be used to communicate entirely in the very high-gain low-data-rate mode, but this requires precise synchronization of the radios before entering this mode. How to optimally utilize all of these capabilities remains a major resource allocation problem. The DSN objectives were to use a small subset of the options and to provide an environment in which the radio could be evaluated and experimented with to address the resource allocation questions.

While the CNT radio development was carried on as an independent project, part of the DSN effort was to develop a radio interface for the DSN Standard Nodal Computers. The interface, which was designed to support all of the features provided by the radio, was implemented on a single multibus board that can be accommodated in any DSN SNC system.

Figure 8.3 illustrates the radio channel structure supported by the interface board. The radio requires a new set of commands approximately every 10 ms. The host SNC computer is responsible for generating these commands. The host can initiate at most one transmission in a single 10-ms period. The transmission start time can be specified as any one of N possible start times within the cycle. The increment between possible start times is 90 μs. The possible start times must be restricted to provide a few milliseconds of free time at the end of the cycle during which the radio performs essential housekeeping functions. A transmission can extend over any number of cycles and, although only one transmission can be initiated each cycle, one or more messages can be received.

A DSN radio communication system design was developed around this basic 10-ms cycle structure [26,27]. The design organized the slots into groups of a few hundred, with subsets allocated to provide different communication services. Each subset could be used to provide a known level of service without interference from the traffic in other subsets. The design is flexible enough to accommodate experimental protocols that might exploit advanced radio features, while allowing for much simplified protocols for initial use. One such simplified broadcast

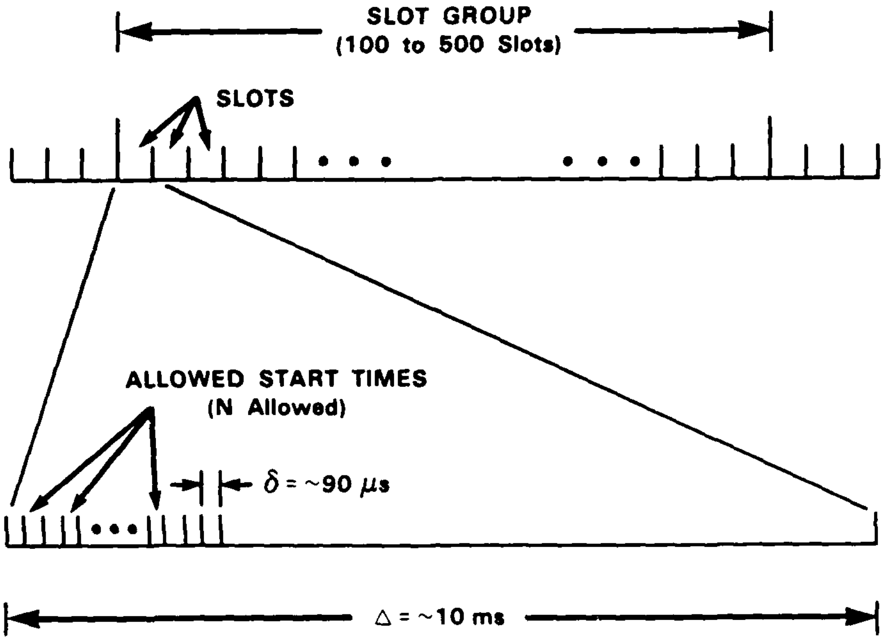


Figure 8.3. Organization of radio channel into slots with discrete packet start times within each slot.

protocol was designed and implemented for NRTS. The implementation utilizes the same pseudo-noise seed for each message, provides no automatic retransmission when transmissions are preempted by incoming messages, and randomly selects the transmission times within the 10-ms slots. This protocol was used in the CNT laboratory to perform preliminary experiments with broadcast communications between two CNT radios.

While no fundamental problems precluded integration of the radios into the test bed, it was clear that the radios, the multibus interface boards, and the NRTS operating system required substantial additional development if the radios were to be used extensively for DSN experimentation. At this point the projects proceeded separately, and the DSN project procured "off-the-shelf" microwave radios to provide radio communication for the test bed.

9. REAL-TIME TEST-BED EXPERIMENTS

9.1. Introduction

The Lincoln Laboratory DSN research effort culminated in a series of multisensor distributed tracking experiments in the Summer of 1986[14]. These final real-time experiments with helicopters are described in Section 9.1. Section 9.2 reviews system test and integration experiments leading up to the final real-time experiments. Other experiments and experimental procedures are described elsewhere. These include off-line acoustic array processing experiments (Section 4), TV cueing experiments (Section 5), real-time experiments with prerecorded data in (Section 6), and real-time tracking experiments with simulated data (Section 10).

9.2. Multisensor Distributed Tracking Experiments

A major goal of the DSN project at Lincoln Laboratory was the demonstration of real-time distributed aircraft tracking based upon the independent cooperative process paradigm, and using a combination of acoustic and TV sensors. This was accomplished, thereby successfully demonstrating the feasibility of distributed mixed-sensor surveillance and tracking systems.

The *experimental test-bed elements* used for final experiments and demonstrations are shown in Figure 9.1 They included five tracking nodes, four acoustic subsystems, and two TV subsystems, all interconnected by Ethernets and microwave radio links. The Ethernets and radios provided both broadcast and point-to-point communications for the demonstrations. This was done using special gateway nodes that created a single logical Ethernet as described in Section 8.

Three acoustic subsystems, a TV subsystem, and user workstations were installed on one Ethernet at the main Lincoln Laboratory building complex. These sensor subsystems and the user workstations were located within 500 m of each other. The other sensors, an acoustic subsystem and a TV subsystem, were located approximately 3 km distant at the Lincoln Flight Facility and at a Raytheon site on the other side of Hanscom Field.

Figure 9.2 shows the geographic deployment of sensor sites as well as some additional information about the demonstration scenario. A UH-1 helicopter was flown through the deployed sensor field as a test target. The demonstration concept called for the helicopter to be acquired and placed in track using microphone data. The initial tracks would then be used to cue

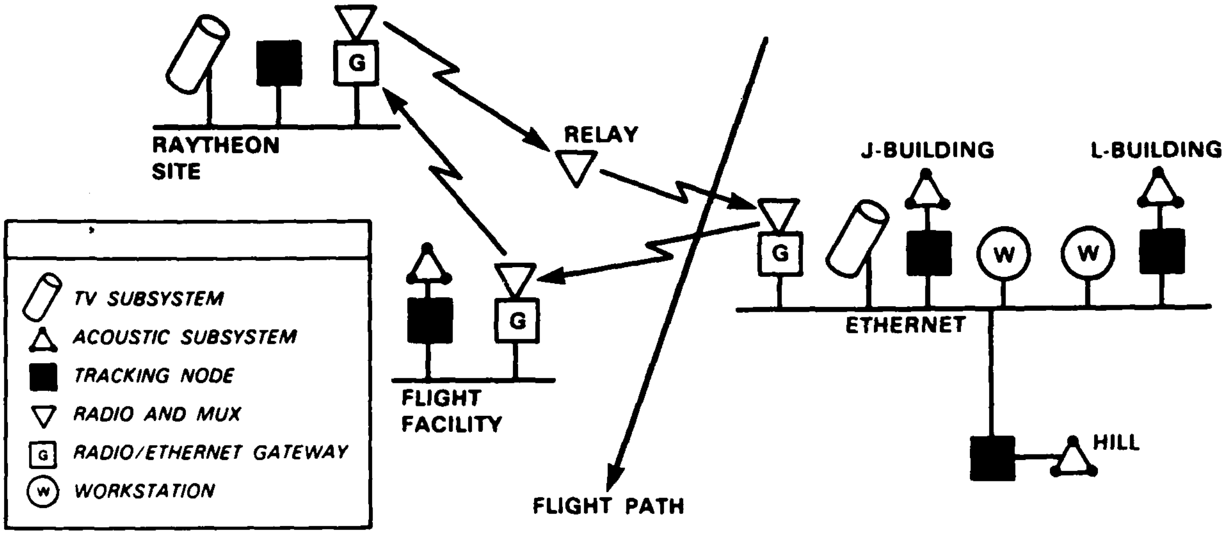


Figure 9.1. Experimental test bed for DSN demonstrations.

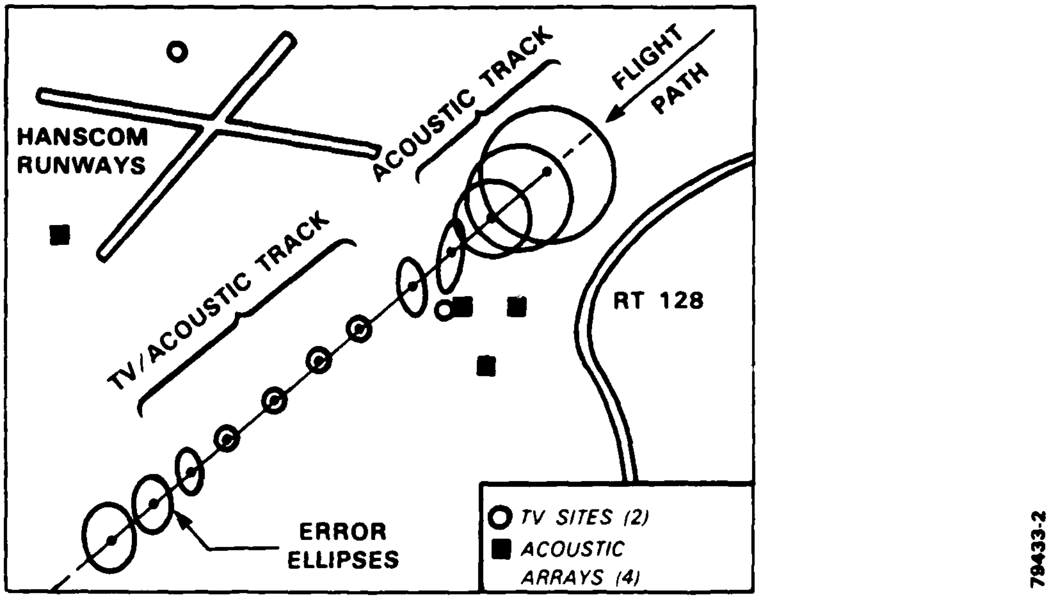


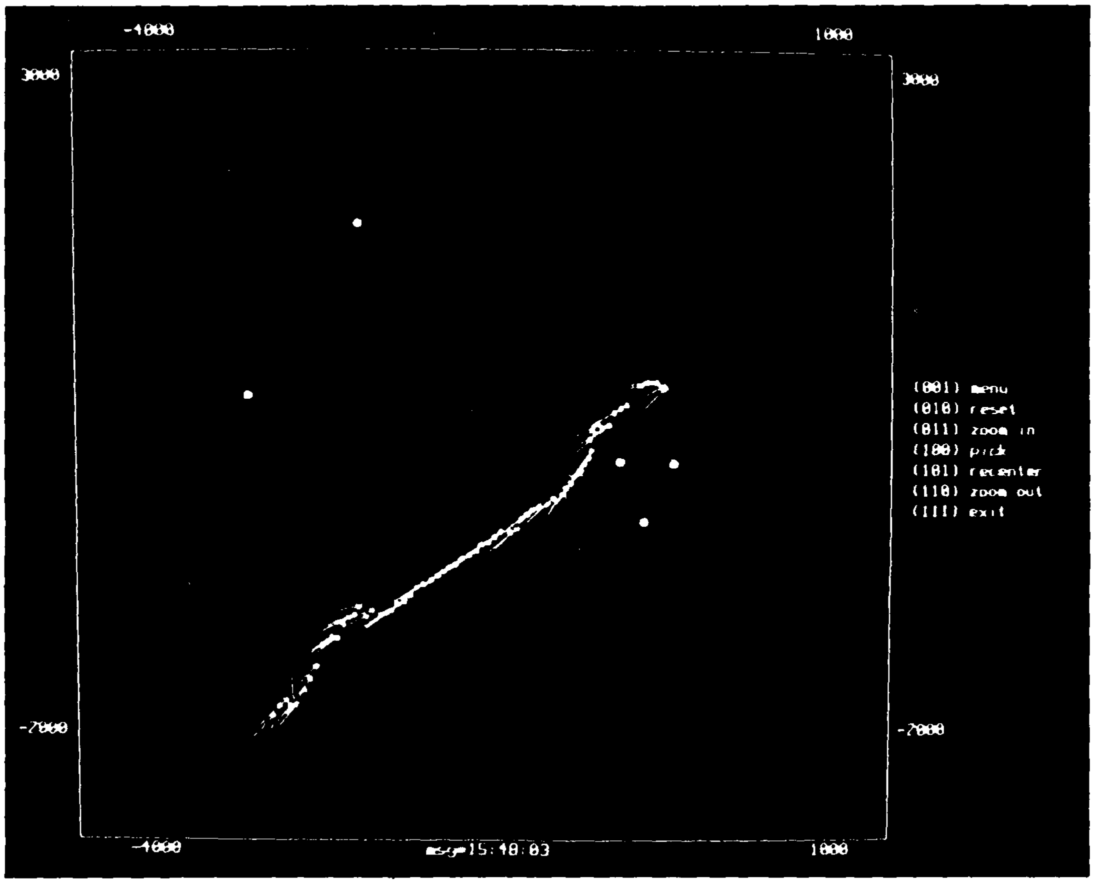
Figure 9.2. Demonstration scenario for distributed mixed sensor tracking.

the TV subsystems which would acquire the target and provide additional directional measurements to be integrated with acoustic measurements. Error ellipses, which are produced by the tracking algorithms, are indicated in the figure. Initially, the axes of the error ellipses are several hundred meters long. As more data are acquired, especially when the TV data are added, the error dimensions are substantially reduced to approximately 100 to 200 m. The expected large decrease in the error ellipses when TV measurements are added is largely because the TV measurements are accurate to a fraction of a degree whereas the acoustic measurements are accurate to only within a few degrees. Later in the track the error ellipses begin to increase agairı as the target leaves the sensor field and new measurements are no longer provided.

Figure 9.3 shows the track for one pass of the UH-1 helicopter during final demonstrations. Several similar passes were made, some from north to south and others from south to north, all with similar results. The choice of flight paths was limited by constraints imposed by air traffic control personnel at the Hanscom Field tower. A comparison of Figures 9.2 and 9.3 shows that the qualitative performance of the distributed system was as expected. The track was established using data from the acoustic sensors and was subsequently improved by jointly tracking with acoustic sensors and TV sensors.

The display of Figure 9.3 is an example of the real-time display provided for users during demonstration runs. Track points are added to the display in real time during the experiment. Each point shows position, direction, and an error ellipse. A finite history, typically 1 or 2 min, of each track is retained on the screen. Track data for the display are obtained via a point-to-point communication link between the user workstation and any of the tracking nodes. Although these data are obtained from a single node, they represent the overall track obtained using all sensors. This results from the nature of the distributed tracking algorithms and the fact that the test bed was operated with all nodes directly interconnected by broadcast communication links. Larger DSN systems, with many more nodes and less communication connectivity, would require additionai multisite data collection and integration. Methods to accomplish this, based upon track combining algorithms that are included in the distributed tracking algorithms, have been investigated as described in Section 13, but have not been implemented in the test bed.

Additional sensor specific displays were developed for the acoustic and TV subsystems. The acoustic displays are azimuth vs time histories. Data for these displays are obtained using

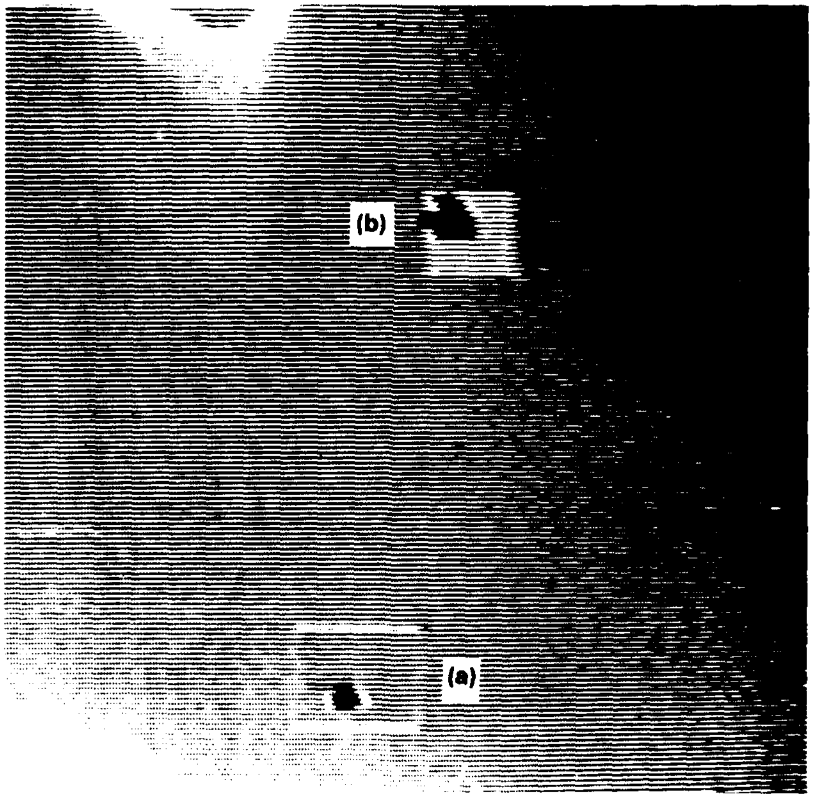


104069-26

Figure 9.3. Real-time helicopter track display obtained by tracking with both acoustic and TV nodes.

point-to-point communication links from any of the acoustic sites to a workstation where the display software is located.

The TV subsystem displays are shown in Figure 9.4. That figure shows the screen of a TV monitor attached to our J-Building TV subsystem. The display shows information from both the local TV on the roof of J-Building and the remote TV. The primary image is the last image from the local TV. If a target was detected in the image, then a box is superimposed upon the screen to indicate the detection [See box (a) in the figure]. In addition, if the remote site detected a target, a small, 2-cm image around the detection is saved, transmitted across the network, and superimposed upon the local TV screen. This image is the binary image labeled (b) in the figure.



104059-26

Figure 9.4. Local TV display with remote TV image overlay.

The 2-cm image from the remote camera is usually large enough to contain the target image because, as discussed previously in Section 5, the zoom control attempts to keep the target size between 1 and 1.5 cm. Although the image is binary, test-aircraft silhouettes have been found to be quite recognizable. The image is transmitted to the user in a 512-byte point-to-point message. The small image patch is superimposed upon the user TV monitor in the same position it had in the image of the remote camera. This window into the remote TV site is another example of the value of multihop point-to-point communications in a DSN.

9.3. System Integration and Checkout Tests

Final demonstrations of distributed multisensor tracking were preceded by a series of real-time system integration and checkout experiments which were carried out over a period of several months. Initial experiments involved the testing of the acoustic signal processing and TV signal processing software at single sensor sites. Subsequent experiments involved up to three acoustic sites and one TV site locally interconnected by an Ethernet cable. The last series of experiments involved integrating the microwave radio systems and expanding the experimental baseline to 3 km as well as adding other acoustic and TV site sites. Experiments utilized targets of opportunity as well as controlled experimental aircraft. Experiments with controlled targets utilized north-south and east-west target paths, depending upon wind conditions that determined which Hanscom runway was in use.

Experiments with controlled targets were performed on the average of one every two weeks for a period of several months. Many additional system integration tests were performed using targets of opportunity.

Algorithms, experimental procedures, and hardware reliability were all significantly improved during this integration and test period preceding final DSN demonstrations.

10. ALGORITHM AND SOFTWARE DEVELOPMENT TOOLS

10.1. Introduction

Several techniques and tools were used to aid the development, testing, and debugging of algorithms and software for the test bed. These aids fall into two classes. One emphasizes controlled experimentation with algorithms, and the other emphasizes general software development tools. Sections 10.2 through 10.6 deal with controlled experimentation techniques and tools. Section 10.7 provides information about software development tools.

The value of controlled experimentation as an aid to the development of a complex system was recognized early and several different forms of controlled experiments were used extensively. This experimentation made use of controlled data sources and several different forms of system simulation. The general technique was to use the same data set repeatedly to investigate the effects of software or algorithm changes and to use different data sets to stress different parts of the system.

Two types of controlled data sources were used as development aids. These are simulated data and prerecorded sensor data. For simulated data the experimenter completely controls the true situation, the number of targets, target trajectories, noise sources, etc., and can investigate algorithm behavior with absolute knowledge of the underlying truth. Section 10.2 describes an acoustic data simulator for the output of the signal processing subsystem. Section 10.5 describes a data simulator for TV-based azimuth measurements. These simulators give the system developer the significant advantage of knowing absolute truth and the ability to precisely repeat or modify experiments.

The other type of controlled data is recorded during live experiments with controlled aircraft. These data are more realistic than simulated data but knowledge of the truth is not as complete. For example, it is not practical to have complete knowledge of ground vehicles, other aircraft, and other noise sources in the experimental area. In addition, knowledge of the true track for the controlled aircraft may be precise or approximate, depending upon the details of the experiment.

Several other types of simulations were use to aid algorithm development. These include: a test-bed simulation environment that is described in Section 10.3, the use of the test bed as a simulation environment as described in Sections 10.4 and 10.5, and a symbolic tracking algorithm simulator that is discussed in Section 10.6.

The less extensive coverage of software tools in Section 10.7 reflects the fact that only minimal tools were used, although more powerful tools would have helped considerably. There was more emphasis upon controlled experimentation than upon software development tools for distributed software because it appeared to be the best utilization of available resources. Substantially better software development tools would have required a more capable distributed operating system. Developing such an operating system, along with more advanced tools, was the focus of more general DSN research at other organizations[28]. It was also clear that commercial workstation operating systems and local area network technology would quickly mature and provide much of what we would require in the future.

10.2. Acoustic Data Simulator

Two different kinds of acoustic data simulation were used in the development and testing of algorithms for the test bed. Time series simulations of individual microphones were used to test signal processing algorithms. Simulations of detections and direction estimates produced by the signal processing system (SPS) were used to test tracking algorithms. The latter were used more extensively and were a more essential part of the system development.

The fidelity and computational cost of simulation are important factors related to the use of simulators. When is a simulator good enough? Does it capture the essential features without paying too high a price in development cost or computational load? For example, the SPS output could be generated by simulating the acoustic time series inputs and applying the SPS algorithms to those time series. The resulting computational load would be considerable; so much so that it might become impractical to compute enough simulated data to effectly test tracking algorithms. Alternatively, a simple statistical model might be formulated for the SPS output simulation at far less computational cost. Based upon computational considerations and the recognition that absolute fidelity was not required, a simple statistical model was in fact used as the basis for simulating the SPS output.

A secondary advantage of the decision to model the SPS output without modeling the input waveforms was that the waveform simulation used for testing signal processing algorithms could be simple. The waveforms were modeled as plane wave signals plus additive white noise. Signals were assumed to be identical at all sensors in an array except for the plane wave delay.

Additive noise was assumed to be independent on the different microphone channels. Test signals were simulated for fixed directions and at several different signal-to-noise ratios. The software required was trivial. There was no need to model the acoustic source, the effects of propagation or the effects of target motion on the received signal.

The model and software to simulate the SPS output were made as simple as possible, but it was necessary to model the source, propagation effects, and target motion. Figure 10.1 shows the principal elements of the simulator and the data flow between them. The input to the simulator is a scenario. A scenario consists of one or more targets flying known trajectories, plus additional information concerning the location and characteristics of acoustic sensors and the background noise. The trajectories are described by piecewise continuous straight line segments with either constant velocity or acceleration along each segment. The outputs are simulated SPS target detections and true target positions as a function of time. These outputs are saved in files for later use. The detections are subsequently used to drive the tracking algorithms. The true target positions are used to evaluate the output from the tracker.

Two forms of interpolation are shown in the figure. Normal interpolation simply converts the description of target trajectories as a series of straight line segments into a sequence of aircraft positions at regular time intervals. The time intervals are the same as the processing cycle of the tracking algorithms, usually 2 s. Acoustic interpolation provides the location and velocity of the acoustic sources at the same regular time intervals. The acoustic position and velocity are node specific and must be calculated for each node location. The acoustic position of a target at time t is the position of the target at the time it emitted the sound received by the node at time t. The acoustic velocity is similarly defined.

The acoustic interpolation outputs are passed to an emission model and to a propagation model. The emission model calculates the spectrum shape and level radiated from the target in the direction of the receiving node. For this purpose the source is modeled by a sum of constant frequency tones. The propagation model adds Doppler shifts and reduces the power levels to account for spherical spreading of the signal. The structure will allow models of arbitrary complexity and fidelity, but the simplest workable models were used.

The propagation model output is labeled "ideal detections." If the SPS was perfect and there was no background noise the SPS output would consist of the ideal detections. The job of the

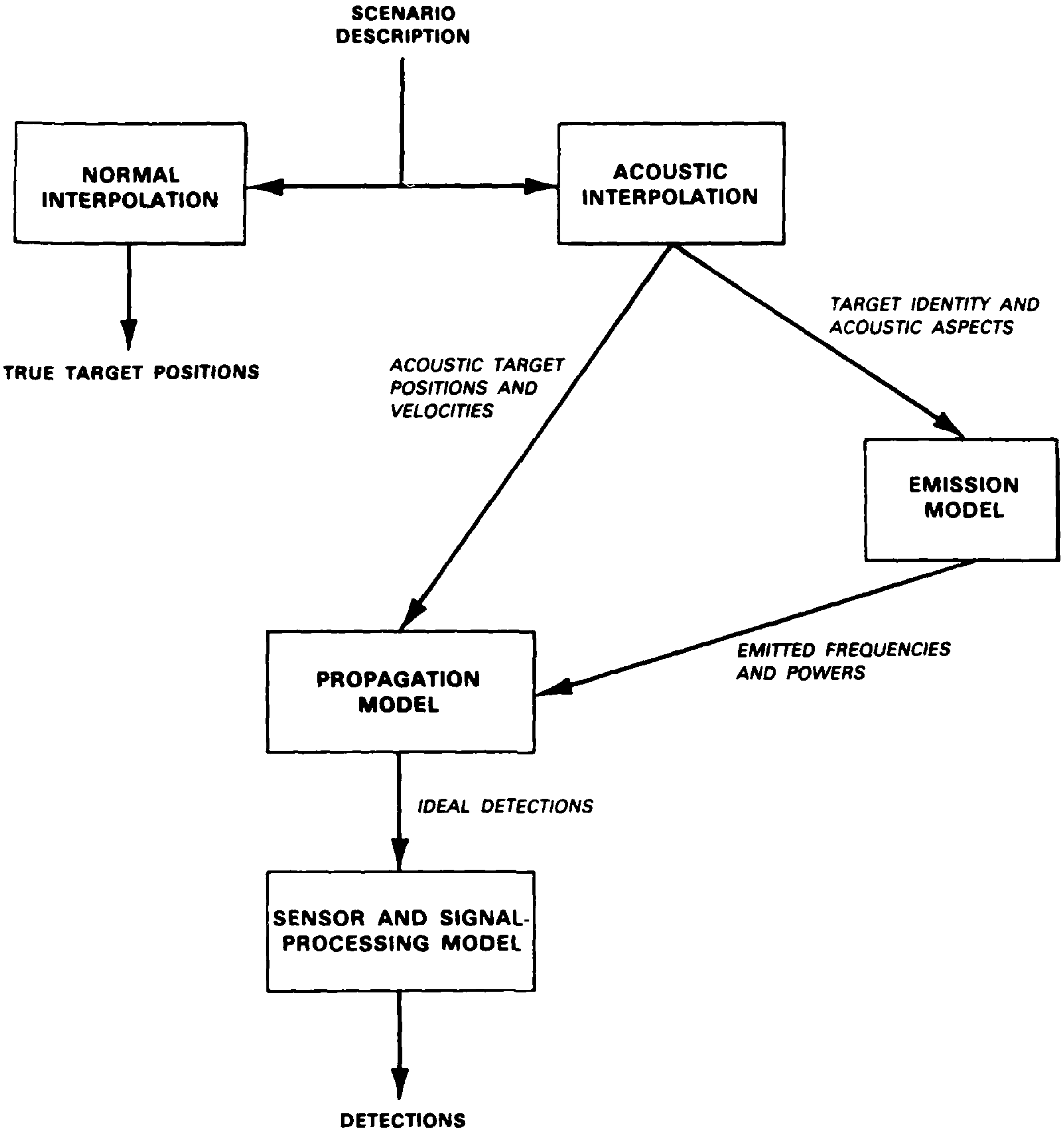


Figure 10.1. Elements of the SPS output simulation software.

sensor and signal processing model is to modify or discard the ideal detections and produce the final synthetic detection data.

Figure 10.2 provides more details for the sensor and signal processing model. Random measurement errors are added to ideal detections and false alarms are introduced. The measurement error module also discards data on the basis of signal-to-noise ratios. The frequency processing module limits the number of frequencies and the band of frequencies. The resolution module combines detections that are not resolvable in frequency or direction.

The sensor and signal processing simulation software were first developed when frequency domain beamforming was being employed and before the wideband algorithm described in Section 4 had been developed. Fortunately, the simulator has served its main purposes, preliminary testing of algorithms and software. It has not been necessary to revise it to better model the SPS output for the newer SPS algorithms. More quantitative testing of tracker performance would probably require revision of the simulator. The revised simulator, while more accurate, would most likely be simpler than the current one because the source and propagation models could be simplified.

10.3. Network Simulator

A software simulation of the test bed was used as a tool for debugging application software before the test-bed nodes and operating system became available. The major elements of the acoustic tracking algorithms and the user interface program were initially checked out using the simulator and simulated acoustic data. Even after the nodes and operating system became available the network simulator continued to be used because it was a more convenient environment for debugging.

Figure 10.3 illustrates the basic application software configuration on test-bed nodes and the UIP computer that controls experiments. The computers are interconnected by an Ethernet or, if radio links are used, by a virtual Ethernet. Each node contains a Sound Processing Subsystem (SPS) and a Standard Nodal Computer (SNC). The SPS executes signal processing software to produce acoustic detections and azimuth estimates. The SNC executes tracking software. It is also possible for a node to have no SPS and for some nodes to be configured as TV subsystems, but these are unnecessary complications for the present discussion.

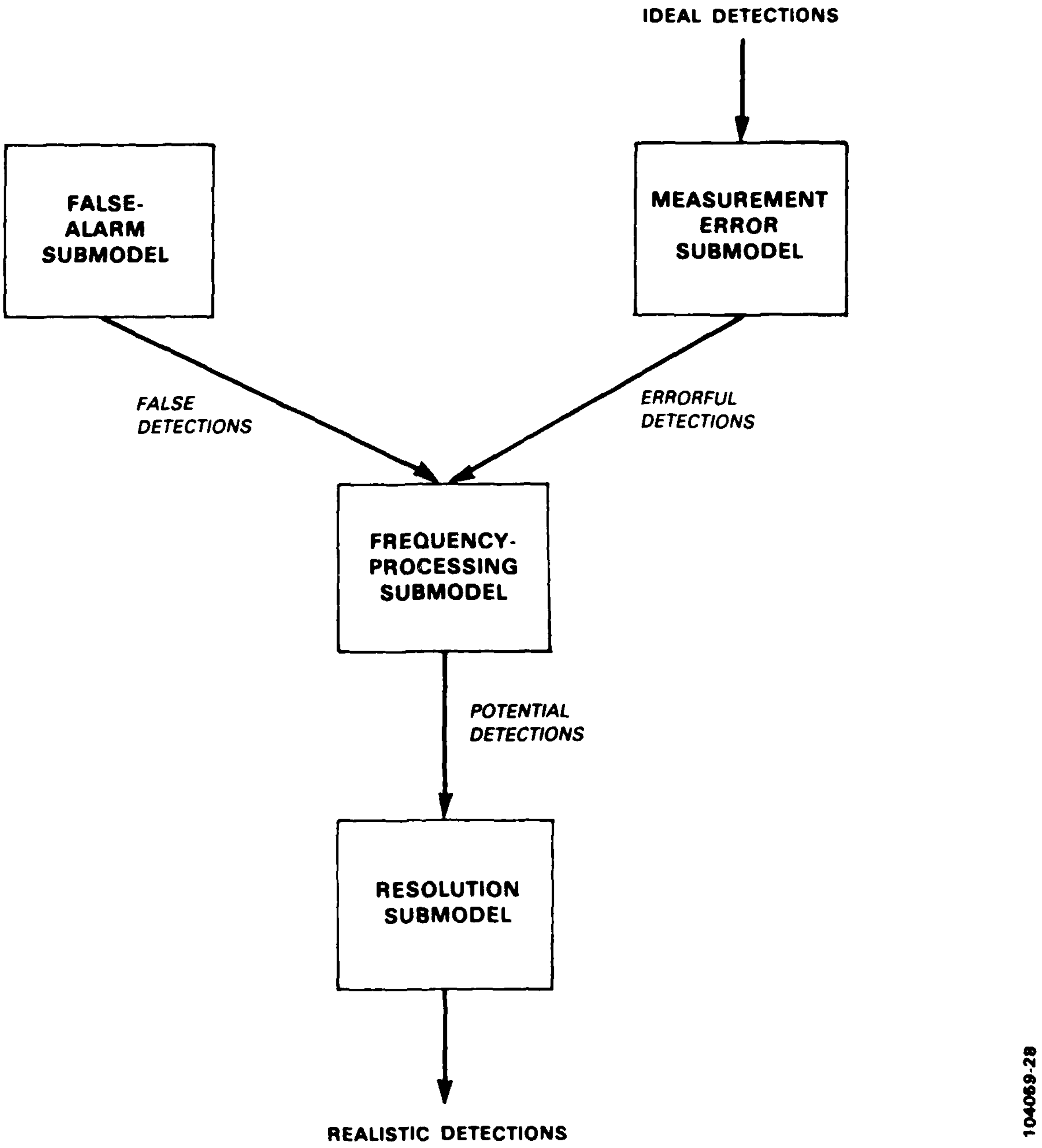


Figure 10.2. Elements of the acoustic sensor and signal processing model.

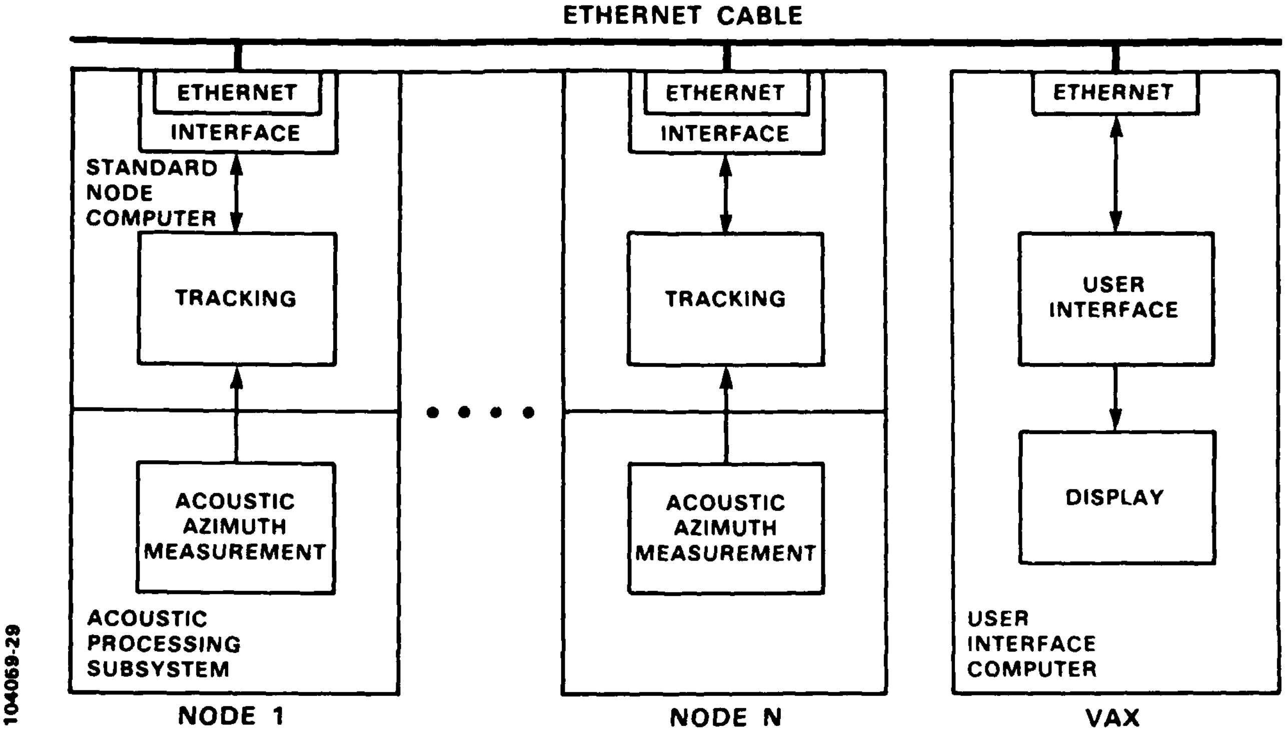


Figure 10.3. Organization of application software.

Within each SNC, the NRTS operating system provides a two-layer interface to the Ethernet, which links the nodes with each other and the User Interface Computer (UIC). One layer formats messages while the other transmits and receives them.

A VAX-11/780 is shown as the UIC although other UNIX workstations with Ethernet capabilities can be used for this purpose. The application software on this computer consists of the User Interface Program (UIP) and display programs. The UIP controls the tracking software and spools its performance. The display programs are used to display tracks and supporting data . The single layer of Ethernet interface in the VAX sends and receives messages. Message formatting is done in the UIP. The UIP and the control of experiments are described in depth in Section 7.

Early in the DSN project it was clear that the test bed would provide limited support for software debugging and that we would begin debugging and algorithm testing before the nodes and NRTS were fully operational. This prompted the development of a simulation of the test bed with enough fidelity to allow most of the debugging and some algorithm testing to be done using a general-purpose computer. The simulation was developed for the VAX computer running the UNIX operating system.

Figure 10.4 illustrates the simulation configuration. The most obvious feature is the Ethernet and communications interface simulation, indicated by the odd-shaped box in the figure. This component of the simulation mimics the Ethernet distribution of point-to-point and broadcast messages, plus the formatting of messages by the Nodal Run-Time System (NRTS) in each node. Another obvious feature is the acoustic measurement interface. The simulation mimics the SPS system by reading simulated or preprocessed SPS output data from a file into the appropriate port of the SPS interface in the tracking software.

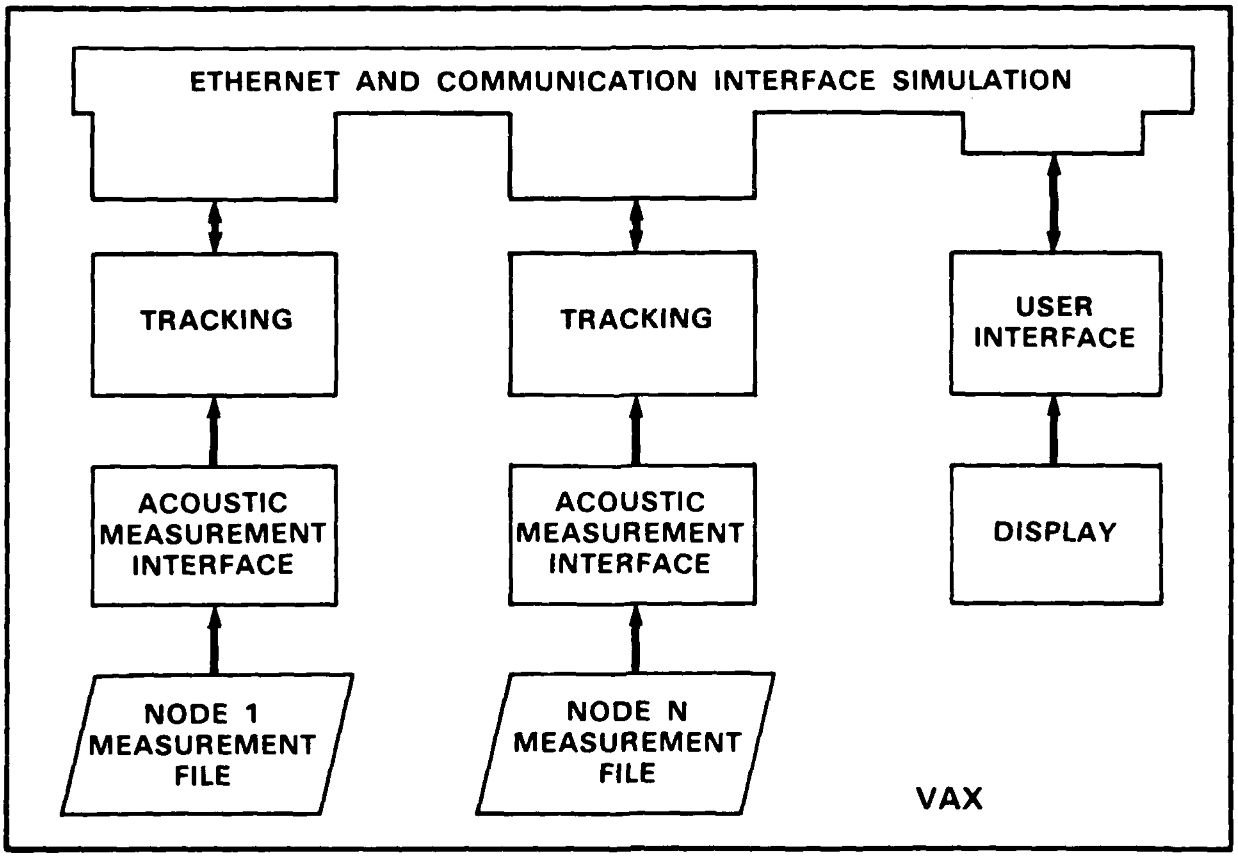

ETHERNET AND COMMUNICATION INTERFACE SIMULATION
TRACKING
TRACKING
USER INTERFACE
ACOUSTIC MEASUREMENT INTERFACE
ACOUSTIC MEASUREMENT INTERFACE
DISPLAY
NODE 1 MEASUREMENT FILE
NODE N MEASUREMENT FILE
VAX
104069-30

Figure 10.4. Organization of test-bed simulator software.

In the simulation environment, NRTS operating system services are emulated by subroutines with identical parameters to the NRTS calls and by following a few special conventions when writing the tracking software. Differences between the simulation and node environment are isolated in a few subroutines. Because of such residual differences, some debugging must still be done in the nodes, bu. the basic tracking algorithm was completely debugged in the simulation environment.

10.4. The Test Bed as an Acoustic DSN Simulator

The test bed can be viewed as a simulator for a DSN system. It is not necessary to deploy nodes and sensors and fly aircraft to perform experiments. Specifically, the test bed can use simulated SPS data as input but operate in every other way as a deployed system. The locations of the nodes in the simulated system can be arbitrary. These locations are used to simulate the input data and are provided to the test bed as input parameters at the start of an experimental run. The physical nodes can be located within a single room while the simulated system is deployed over an area of many kilometers. Also, the communication connectivity can be arbitrary. The test bed provides complete communication connectivity between nodes, and software is used to modify that connectivity. The test bed has been used extensively in this way as a real-time simulator of deployed DSN systems. Only occasionally has it been deployed as a geographically dispersed tracking system for live targets.

The test bed has been used as a simulator for DSN systems of up to eight acoustic nodes. The simulated data are read from floppy disks located at each node. Tracks and other experimental output are logged on floppy disks at the nodes and transmitted by Ethernet to the UIP for real-time display and recording in a spool file. The data logged at the nodes are always complete while the data in the spool file may be incomplete due to temporary overload of the Ethernet software on the computer supporting the UIP.

Figure 10.5 illustrates the procedures for running an experiment using floppy disks for input and for data logging. At the start of the experiment, separate input data and logging disks are mounted on each of the nodes. At the end of the experiment, the floppy disks are removed and transported to a VAX where they are read into files for post-experiment analysis. Several experimental runs can be completed with the same physical disks and the results stored on different partitions.

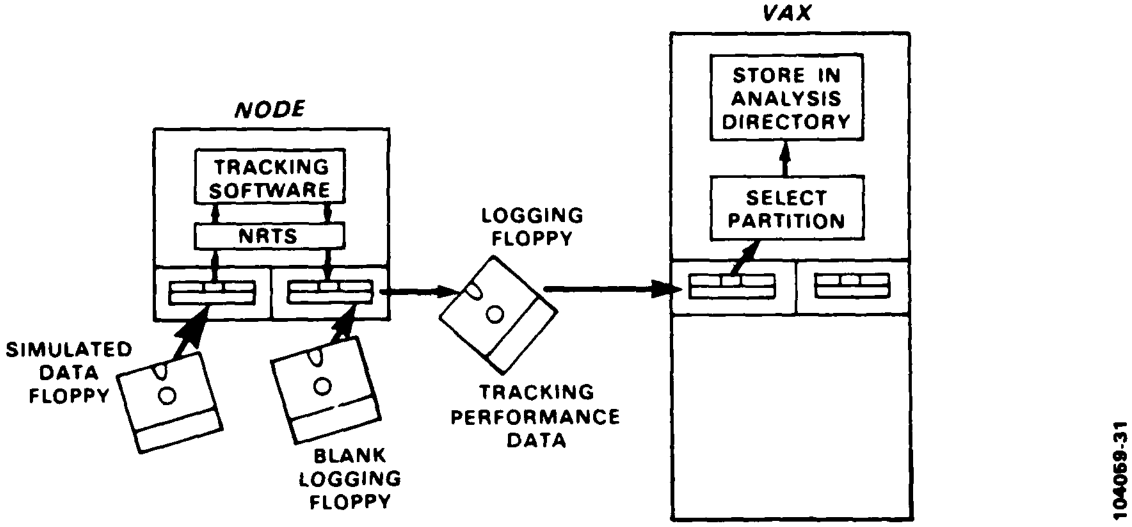


Figure 10.5. Procedures for using simulated data and logging performance.

Following are two examples using the test bed as a DSN simulator. The first is an example of an experiment involving two targets. The simulation of two target situations is not difficult, much less difficult than executing two-aircraft experiments with live aircraft. The second example, alluded to in Section 6, revealed circumstances under which the tracking algorithm fails to recognize when two tracks correspond to the same target.

Figure 10.6 shows tracks obtained from the simulated flight of two helicopters from left to right through a rectangular grid of nodes. The experiment involved six nodes arranged in three pairs and separated from each other by 5 km. Each node could only communicate directly with its nearest neighbors. An acoustic detection range of 5 km was used for the simulation. Since the scenario, detection ranges, and communication ranges are all known with certainty it is often easier to understand the behavior of the system from such experiments than it is from live experiments.

Only the first two pairs of nodes are shown in the figure, displayed as yellow triangles. The figure is a photograph of the display generated during the real-time DSN simulation run using the same display program that generated Figure 9.3 during live real-time experiments. The extra blue lines on Figure 10.6 are the true tracks for the two helicopters. These are known for simulated scenarios and the display program provides an option to display them. The photo was taken just before the lead helicopter left the field-of-view of the display.

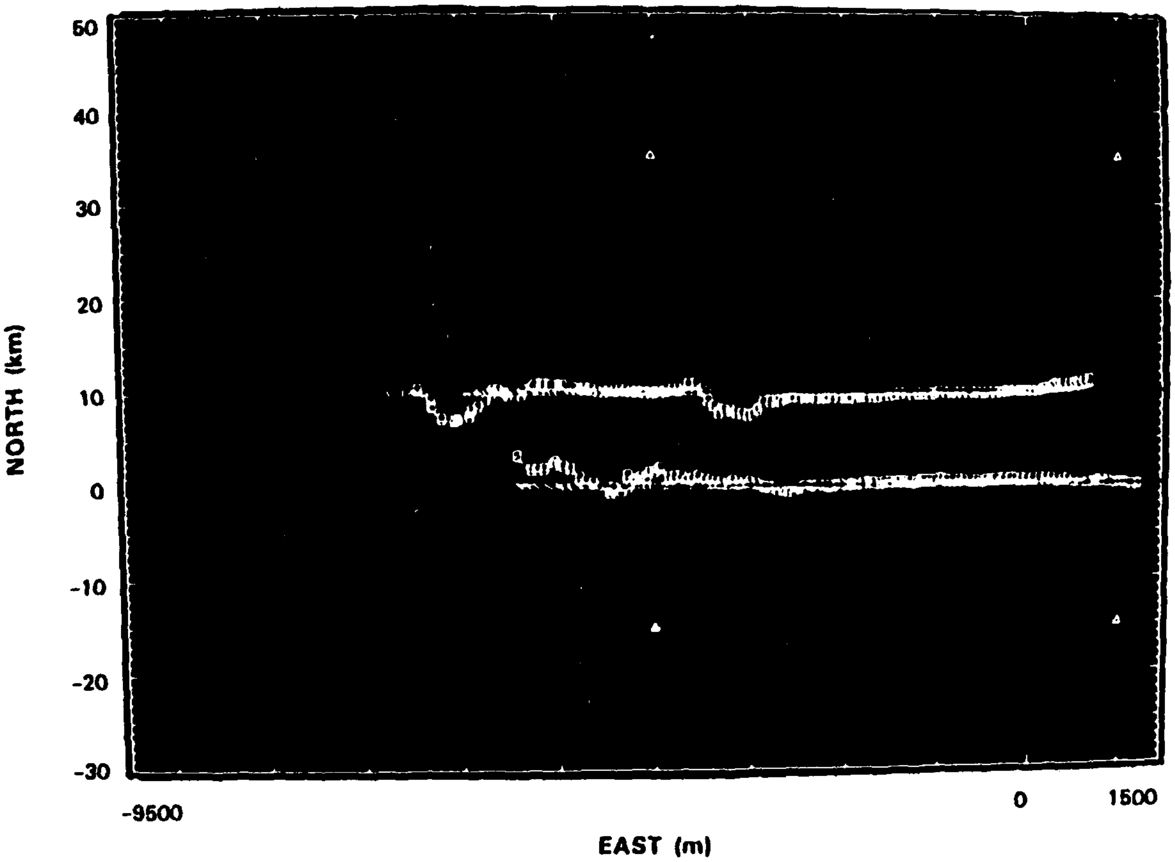


Figure 10.6. Simulated track of two helicopters through six nodes. The six-node system was simulated in real time using the test bed.

Although the nodes have a simulated 5 km detection range, the initial location estimate for the upper helicopter is only about 3 km from either node. This is caused by two kinds of delay: algorithm delay and acoustic propagation delay. The algorithm delay results because position track initiation requires the use of two well-established azimuth tracks and this requires several 2-s observation cycles. Acoustic propagation delays cause the target to be closer than its acoustic detection range when it is first detected. For example, the helicopter is only 4.5 km from the nodes when the nodes receive the sound emitted at a range of 5 km. The acoustic delay effect appears even more obvious for the lower target because the target is further away from the upper node for this case.

The location uncertainty of both tracks first decreases, then increases slightly as the aircraft pass between the first two nodes, and decreases substantially shortly thereafter. Additional azimuth measurements and a good geometrical configuration explain the initial decrease. The increase when the targets pass between the first two nodes is because the measurements provide very little location information along the line joining the two nodes. The sudden large decrease occurs when the targets come within range of the next two nodes and azimuth measurements from a total of four nodes contribute to the position tracks.

Other experiments with the same data were used to test the robustness of the tracking algorithm when faced with communication failures.

Figure 10.7 shows the tracks obtained during a simulation experiment that revealed one particular problem with the tracking algorithm. The display is similar in form to that of Figure 10.6 but is a black-and-white version and does not show ground truth. This experiment involved only the four nodes shown as triangles on the display. These have been labeled 1, 2, 3, 4 for reference. A single helicopter flew from left to right midway between the bottom row of nodes and the single node at the top. Communication and detection ranges were the same as for the previous experiment.

An obvious problem with the tracks shown in Figure 10.7 is that there are two tracks and only one target. The two tracks begin close to each other, diverge somewhat and, with the exception of one of them toward the end of the run, never become high quality. A detailed analysis of the data showed that one was initiated by nodes 1 and 2 and the other by nodes 2 and 3. Both

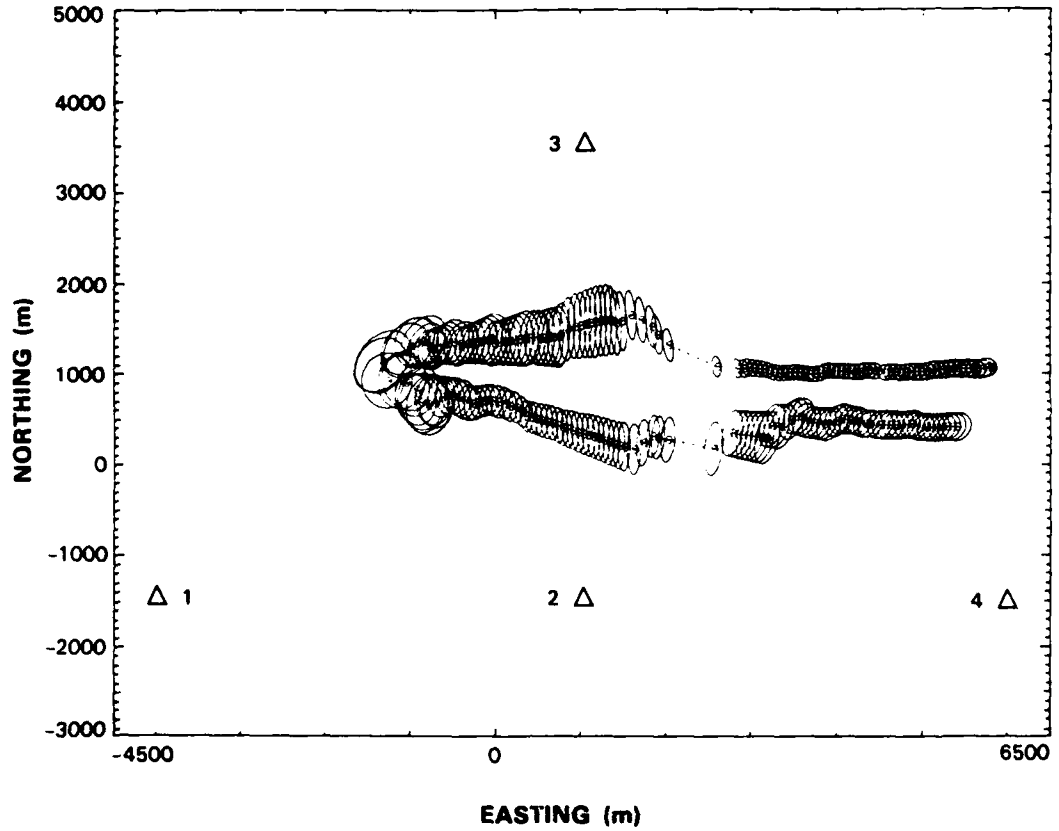


Figure 10.7. Duplicate tracks obtained during the simulated flight of a single helicopter. This resulted because two different pairs of nodes simultaneously initiated track. Methods to correct this problem are known but not implemented.

were initiated during the same 2-s processing cycle. Because of this, track initiation and naming mechanisms (discussed in Section 6) which attempt to avoid multiple track names for the same target, failed to avoid multiple names in this case.

Fortunately, this failure is a low probability phenomenon. It occurred because nodes 2 and 3 began detecting the target at exactly the same time but after node 1. The result was that node 3 initiated a track using its data and data from node 2, while node 1 did a similar thing using the same data from node 2. Subsequent to the duplicate track formation each new azimuth measurement was used to update one or the other of the tracks, but not both. Thus, neither track obtained full benefit of the available data.

The specific problem illustrated by this experiment is well understood. Simple tests involving the appearance of the same azimuth track in two different position track identifiers would detect its occurrence and the tracking algorithms could be modified to eliminate the problem. But, as discussed in Sections 6 and 13, the general problem of detecting and combining multiple tracks for the same target remains an unsolved problem.

A portion of each of the two tracks shown in Figure 10.7 is missing. This is due to a temporary overload of the Ethernet connection used to collect the data in real time from the nodes.

10.5. The Test Bed as an Acoustic and TV DSN Simulator

Simulation software was developed for real-time tracking experimentation using simulated TV measurements in addition to simulated acoustic measurements. For experiments involving TV as well as acoustic data, the test bed is used as described in Section 10.4, with the addition of TV measurement simulation software for TV nodes. The TV software simulates the image processing and target detection modules of the TV subsystem. The rest of the TV subsystem is exactly the same as for experiments with live aircraft, including target selection algorithms, camera control commands, and camera motion.

Figure 10.8 shows the elements of the simulation module for the image processing and target detection functions. The simulation parameters, including a description of the true aircraft track as a function of time, are provided to the module by the user before the start of an experiment. Target selection and camera pointing are performed by the camera pointing module as for any live experiments. These functions are not simulated. Mount pick-off points provide real-time camera orientation information to the TV simulation software. The outputs from the simulation module are zoom and elevation control commands, azimuth measurement messages, and a real-time display of the simulated target superimposed on a TV monitor.

The major elements of the TV simulation software are indicated in Figure 10.8. The target dynamics simulator computes target position and orientation from the same scenario description used by the acoustic data simulator. The plane-of-view (POV) simulator calculates target features as they might appear on a TV frame. These include the target azimuth and elevation, the target position in the POV, and the target image size. The size is based upon target dimensions provided to the simulator by the user. The measurement generator produces an azimuth measurement message if the simulated target position is within the TV POV and if the target projection onto the POV is large enough. Random errors, with statistics selected by the experimenter, are

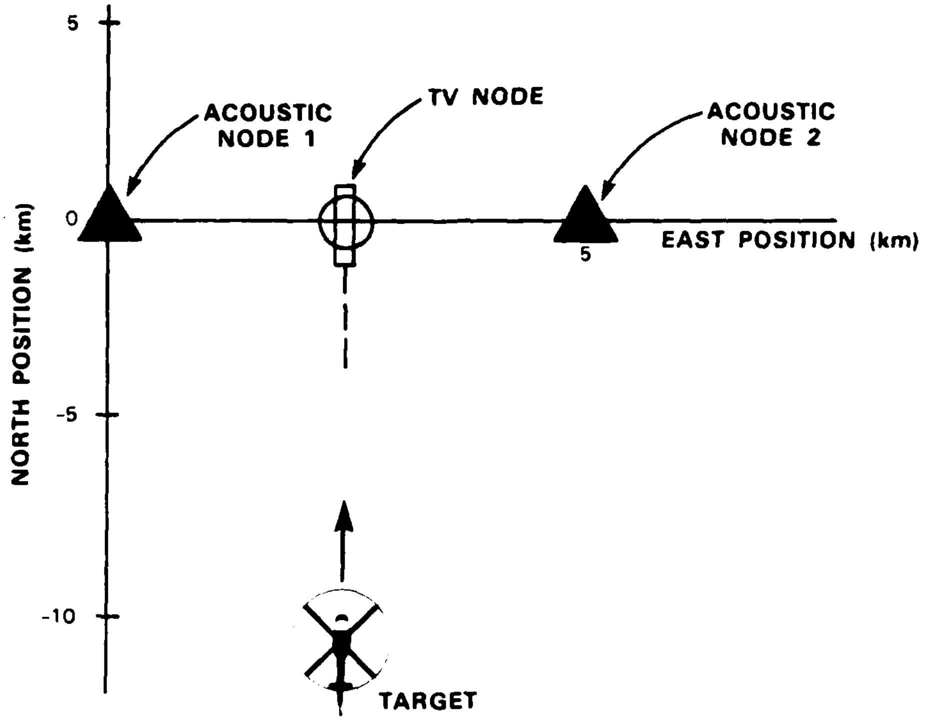


Figure 10.8. TV simulation module for real-time experimentation.

included in the azimuth measurement. The measurement generator also provides zoom and elevation control messages to control the camera. The TV display generator superimposes on the TV image a simulated target image of the right size if it is within the camera field-of-view (since there is no real target).

An example illustrates how the TV simulation software was used to test software, to tune algorithms, and to demonstrate tracker performance differences with and without a TV sensor. Figure 10.9 illustrates the simulated node locations and target track for one series of experiments in which three test-bed nodes were used to simulate two acoustic nodes and one TV node. The simulated target trajectory was south to north at a speed of Mach 0.1 and an altitude of 350 m. The camera, when it was included, was oriented due south at a 10° elevation with the zoom set for a 15° field-of-view. The pointing algorithm should choose to follow the target as long as possible while the target is coming toward the TV. Once the target rises above the field-of-view of the camera, the camera should slew 180° and wait for the target to re-enter the field-of-view from above.

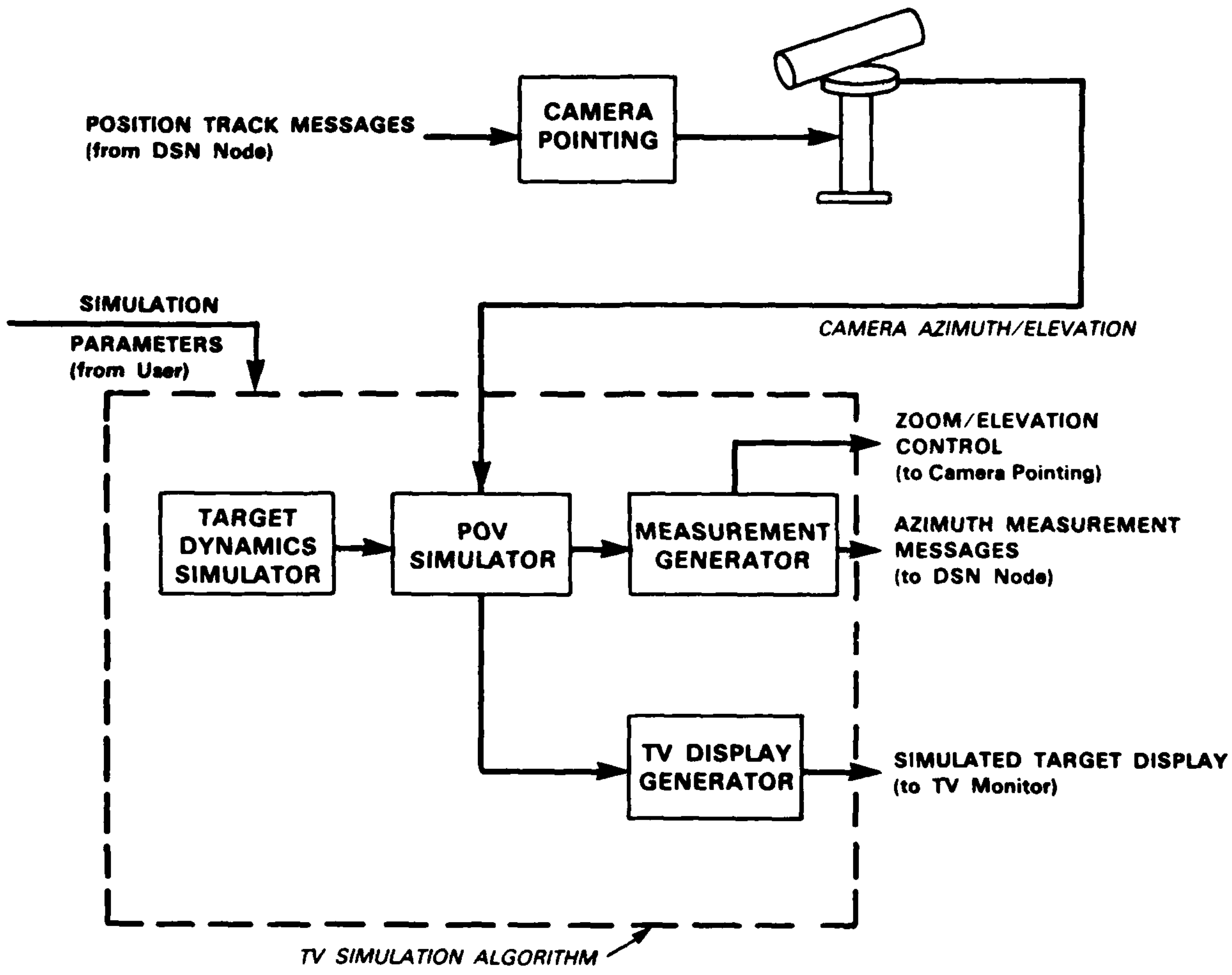


Figure 10.9. Simulated combined TV/acoustic tracking scenario.

Figure 10.10 shows an example of tracks obtained with and without TV azimuth measurements. The initial error ellipses are large in both cases, on the order of several hundred meters. Without TV measurements, the error ellipses remain sizable throughout the entire track, especially in the east-west direction because of the sensor/target geometry. With TV measurements the performance, after the first few track points when only acoustic measurements are used, is changed markedly. After the first TV measurement is processed (Point 3), the error ellipses show a significant reduction because the TV measurements have no propagation delay and are more accurate than the acoustic measurements. Once the target leaves the camera field-of-view the

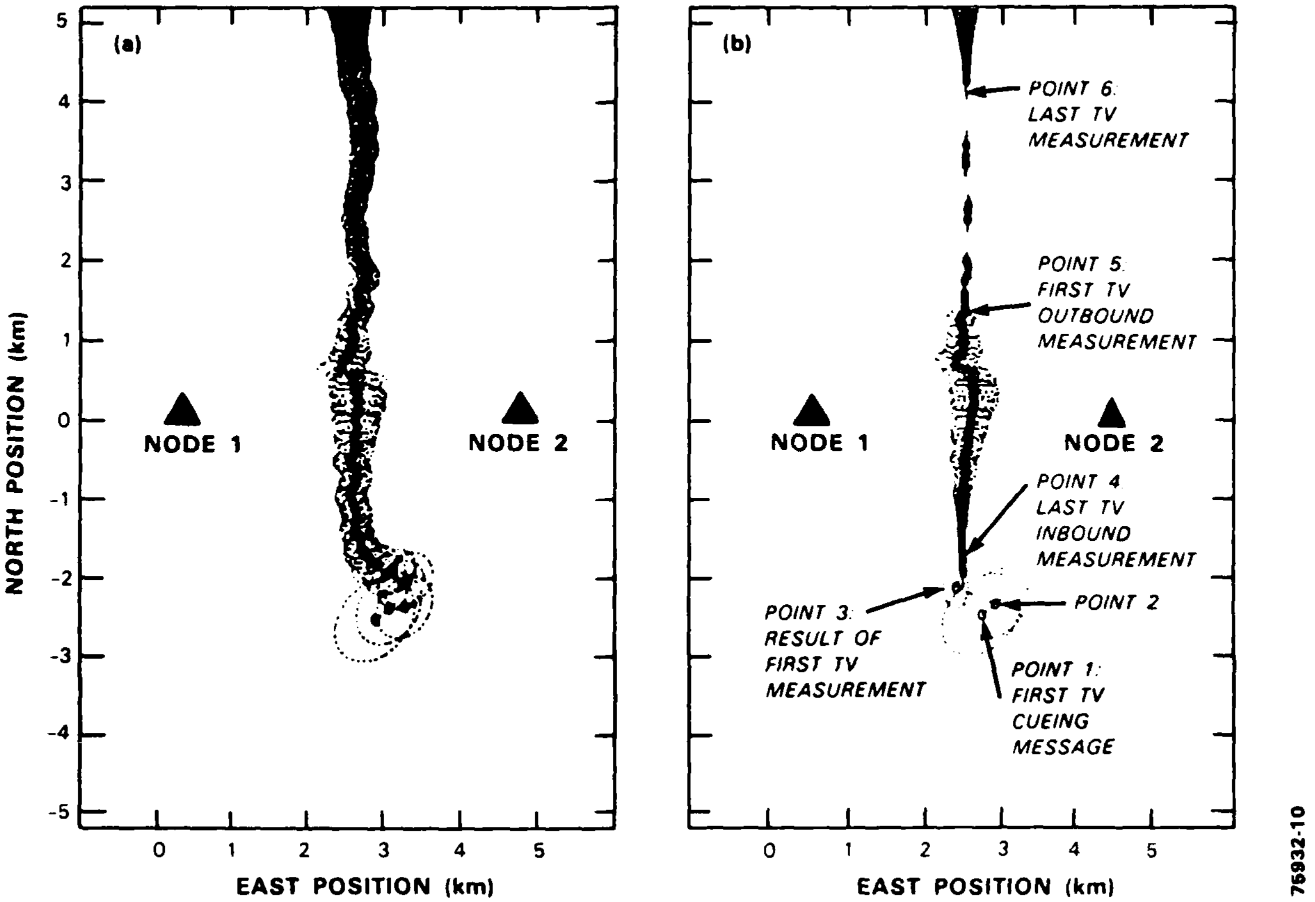


Figure 10.10. Position track and error comparisons: (a) with and (b) without TV measurements.

camera slews and waits to acquire the target on its outbound path. During this time the error ellipses grow to the levels obtained with acoustic-only tracking. When the outbound target enters the top of the field-of-view of the camera (Point 5), error ellipses decrease again and remain small until after the last TV measurement is obtained (Point 6).

These results were an initial demonstration of how distributed sensors with complementary characteristics can be used to achieve improvements in system performance. The field-of-view limitations of the TV sensor are complemented by the omnidirectional acoustic sensors that are used to initiate tracks, to provide cues to the TV subsystem, and to maintain tracks while the TV sensor slews and the target is passing overhead. The precision limitations of the acoustic sensors are supplemented by the TV sensor which provides high-accuracy azimuth measurements even at long distances. The result is longer and more accurate tracks.

These experiments were an important milestone in the development of the test-bed system. They were the system integration tests for the tracking software and algorithms before they were integrated with radio communication, acoustic signal processing, and TV signal processing software.

10.6. Symbolic Information Flow Simulation

Section 6.2.5 discusses the internodal information exchange needed for position track combining. The discussion is illustrated with diagrams showing the symbolic flow of information between nodes and how different track combination procedures modify that flow. The figures were obtained by symbolically simulating the procedures and the information flows. The simulation was developed as an aid to understanding some unstable tracking behavior that was observed in an initial version of the tracking algorithm. The simulator helped show that the behavior resulted from a subtle flaw in the position track combination procedure. The simulator was also found to be generally helpful in understanding how track combination procedures could fail due to lost or delayed messages. The development and use of the simulator were inspired by diagrams, similar to those in Section 6.2.5, that have been used by ADS Inc. in describing their distributed tracking algorithm framework[30].

The simulator represents information abstractly and symbolicly. Algebraic expressions are used for the combination of information. No numerical calculation of estimates or covariance matrices is performed. The use of symbolic rather than numeric values makes the sources and flows of information explicit. Numeric calculations do not give any indication of how the results are obtained, whereas the symbolic simulation focuses upon exactly that question.

The symbolic simulator used during the development of the track combining procedures includes four basic operations: (1) initialization of target information; (2) addition of new information directly from sensors; (3) transfer of information by broadcast, including the possibility of message loss or delay; and (4) addition and subtraction of information by a position track combination procedure. These operations were implemented as a few functions using the symbolic algebraic manipulation system, MACSYMA[29]. Different position track combination procedures corresponded to slightly different functions. A particular pattern of broadcasts, lost messages, and delays was realized by the arrangement of the functions into a simple program. The simulator did not require the full power of MACSYMA. It merely exploited the MACSYMA capabilities for forming, representing, and displaying symbolic sums and/or differences.

Such hand tailoring of programs on a vastly overpowered system was adequate for our limited use of symbolic simulation techniques. But a dedicated simulator, probably written in LISP, would be a more effective tool if symbolic simulation was to be used more extensively. A dedicated simulator could allow one to describe the effects of a position track combination procedure more explicitly and to specify the situation (pattern of broadcasts, lost messages, etc.) more succinctly. A good graphical interface could serve the latter purpose.

In retrospect, a dedicated system should probably have been built and used in the early stages of algorithm development to test algorithm concepts. It is difficult to anticipate how complex distributed systems with asynchronous components will behave, particularly in the face of phenomena like lost broadcast messages. And it is very difficult to diagnose unexpected behavior after a system is fully implemented. Subtle design flaws may not be readily distinguished from subtle implementation errors. Abstract simulations of a system, such as our symbolic simulator, might be used to identify design problems early. More important, they could be used to test design alternatives when paper-and-pencil analyses cannot capture all the complexities of the system and without incurring the expense of fully implementing each alternative.

10.7. Software Development Tools

The primary software development environment for the DSN project was the UNIX environment on a VAX 780 computer. Source code was written, compiled, and link-edited under UNIX. Load modules for the other computers were transferred using tape or floppy disks or downloaded using serial lines connected to the VAX. Most code was written in the C language with a small amount of assembler code. Software development was carried as far as possible within the VAX environment because of the available software support.

Program development steps for test-bed software are standard. First, the source files are prepared using an editor. The source files are compiled or assembled using an appropriate translator to produce object files. The object files are link-edited, possibly with some libraries, to make a load module. For the SNC computers the load module is converted into a special loadable format used by the resident monitors on the SNC boards. In the case of the PDP-11/34 computers in the SPS systems, this step was not required because of the similarity between the 11/34 and VAX systems. The final product of this sequence is a file that can be downloaded or booted into a test-bed processor.

As noted above, the test bed includes PDP-11/34 computers as well as the MC68000 processors in the SNCs. Some of the code for the SPS, notably the acoustic data acquisition and recording code, was developed completely under UNIX in much the same way as the code for the SNCs. The code developed later, which implements the signal processing algorithms described in Section 4, was developed partly under UNIX and partly on a PDP-11/34 under the RSX11 operating system which supports the real-time signal processing code. Address space and memory limitations on the the PDP-11/34 computers and memory limitations in the attached FPS-120 signal processors complicated the development of the real-time signal processing system. But, compared with the NRTS system, RSX11 is a mature product with better tools for program testing and debugging. Because of this, and because the signal processing code could be tested without interaction between nodes, the discussion below emphasizes the SNC systems.

SNC processors are loaded in one of two ways. A load module for any of the processors in an SNC can be downloaded to the processor on a serial line between the SNC and the VAX. Alternatively, the load module can be copied onto a floppy disk and transferred to an SNC where it is used to boot one or more of the processors in the SNC. The floppy disks eventually superceded the serial lines for booting purposes. The main reason was that the floppy disk mechanism performed the task faster and easier for the user.

Although booting from floppy disks displaced downloading over the serial lines, the lines continued to serve usefully in the software development process. They provide a mechanism for conveniently resetting and rebooting the SNC processors. Thus a software developer, or other test-bed user, can perform these functions with several nodes from a single VAX terminal serving as the system console for each of the systems in turn. In addition there may be several software developers using different nodes at the same time. A mechanism is needed to control node access. This is done using the serial lines and a simple "lock/unlock" command. A user wishing to use a specific node requests a "lock" on it. This is granted if the node is available. Otherwise the user is told who is using the node. Users are expected to "unlock" nodes when they are finished using them.

The lock/unlock commands maintain an information file for each node. In addition to providing information about node usage, these files store various status information about the nodes such as broken or missing hardware and a list of the programs resident in the nodal processors. This allows users to avoid nodes with hardware problems and often saves download or booting time. Much of this information must be entered manually by users or service personnel.

Each of the SNC processors provides console facilities that permit users to examine and modify memory and registers, and to start and stop programs. The standard console command set was modified slightly. One command was added to load programs from a floppy disk. Other features were specifically motivated by the distributed nature of the test bed. Features were added to reset all nodal processors with a single serial line. The primary SNC processor P1 is attached to that line and is reset automatically when a special reset message is received. Other processors in the node can be reset indirectly by using P1 to relay reset commands. Figure 3.5 shows the remote reset device used to monitor the primary node serial line for reset codes as well as the secondary reset lines to the other processors in the nodes. The same paths also provide a remote user access to the monitor in any of the processors in the node. New I/O features that were added to the monitor included a transparent mode and an option to use a general-purpose serial input as the console interface as an alternative to the dedicated console interface on each board. Resetting a SNC processor clears its on-board memory. In practice, this means when a processor is reset it must be reloaded with a program.

User interaction with SNC systems for software debugging is limited to the primitive interaction provided by the monitor program plus any windows that the programmer provides at the application software level by including debug messages in the user code. The result is that it is difficult to debug programs in the nodes.

11. DISTRIBUTED SURVEILLANCE SYSTEM DEVELOPMENT LESSONS

Several expectations were confirmed and lessons learned regarding the development of a distributed DSN system. These concerned: (1) the role of experiments, (2) operating systems and debugging, (3) internodal communication, (4) physical distribution, and (5) system complexity. Following is a summary of observations in each of these areas, based on our experience, that we hope will be of value to future developers of distributed surveillance systems.

Controlled and repeatable experimental capabilities are essential for the development and test of any complex system or set of algorithms. This is especially important for distributed systems. They are more likely to exhibit unexpected behavior than traditional systems. Simulated data, raw sensor data, and partially processed data recorded by several nodes during live experiments were all used extensively for controlled and repeatable testing. The test bed was designed to support real-time experiments using these data. This allowed well-controlled algorithm and system testing that would have been impossible to achieve with live experiments.

A related fact is that when live experiments are performed it is essential to analyze and fully understand system behavior, especially unexpected behavior. This is possible only if the system saves enough data and provides the capability to reprocess the data to test if problems have been correctly diagnosed and corrected. Detailed data logging and the reprocessing might not be important for a fielded system but are essential for a development system. This is an example of the general rule that the development system requires more capability than the fielded system. This is even more the case for distributed systems.

Neither a distributed operating system nor well-developed remote debugging tools were available at the inception of the DSN effort. Thus, it was necessary to develop an operating system and interprocess communication mechanisms for the test bed. The resulting software is necessarily simple and provides limited support for remote debugging. The lack of a fully functional distributed operating system and remote debugging tools substantially increased the difficulty of developing application-level DSN software. Although operating systems and tools for distributed systems are currently areas of active research it is also clear that distributed systems, often with sophisticated workstations and file servers interconnected by a local area network, are becoming relatively common. A new DSN test bed could make use of these advances and would be far more flexible and easy to use for research and development of future DSN systems.

The DSN approach is to use broadcast communications for distributed tracking and to use point-to-point communication for other functions such as collection of data from areas and to support system operation and development. No satisfaciory communication system existed for the test bed at the inception of the DSN effort. Experimental Packet Radio systems could probably be modified and used for DSN applications but this was not possible in the DSN time frame. Therefore, part of the DSN effort included the procurement of microwave communication and Ethernet hardware to interconnect nodes and development of special-purpose software to provide internodal communication services. Although this required considerable effort, it is a minimal system without the ease of use and all of the features that one would want for a DSN or for extensive DSN experimentation in the field. The communication system is an essential tool for DSN development as well as being an element of the final system. Communication system requirements are more stressing during the research and development phase than for the final system. For example, the development process requires the collection of detailed information for test and evaluation and frequent changes in the network software stress the communication system. A next-generation DSN experimental system should make use of advances in local area network and packet radio technology.

Another important point is that the communication system performance is typically limited as much by software and protocol processing as it is by more traditional physical link parameters. Throughput in the test bed is adequate to support simple distributed tracking experiments. But larger scale experiments or experiments requiring substantially more data to be exchanged would be limited by the communication system. The communication protocols and message processing loads are the limiting factors, not the physical communication media. This is common for computer-to-computer communication systems and should be kept in mind for future DSN system development.

Software development and debugging typically involve an ongoing process of compilation, changing load modules, and testing. This process is often more difficult in a distributed environment than in a centralized one. This was certainly true for the DSN test bed which was constrained by operating system features, available software tools, and the communication system. Whenever possible the development of all software was done in the centralized environment of a single computer. This included some simulation of the distributed environment. Future system development should probably exploit modern local area network technology to provide a good DSN development environment. The test bed utilizes Ethernet hardware, but advanced general-purpose workstations and well-supported network software were not widely available in the earlier stages of the project.

Distributed software testing in the nodes was deferred as long as possible because of the difficulty of working with a distributed system with available tools. During early stages of development, 9600-baud serial lines were used for communication between nodes and between the software development computer and the nodes. Distributed testing required lengthy downloading to the nodes. The final test bed includes floppy disks at each node. Disks are written on the central software development machine and physically loaded on the nodes to make software changes. It would also be possible to download using the test-bed communication system, but the necessary software has not been developed. In addition, the process might still be a bottleneck because a single machine would perform all downloading. This could be somewhat alleviated by an incremental downloading system that provides for modular downloading of only small portions of the software that change.

Distributed systems must be designed from the start for remote operation of system elements. This includes remote hardware startup and diagnosis as well as distributed software features. The test bed used board and higher level commercial products that were not designed for use in a distributed system. The scale and nature of the DSN effort forced us to take this approach and to minimize attention paid to designing for remote operation and test. As a result it is clear in retrospect that the test bed is more difficult to operate and use than it need be.

These are examples of difficulties that arise in the development and testing of distributed systems. A good infrastructure to support the development of distributed systems was nonexistent when the DSN project was started. The situation is improved but there is still much that could be done to make the development, testing, and use of DSN systems easier.

And, finally, DSN systems are complex and this requires extensive test and maintenance, automated tools for system operation, and automated tools for software development. Hardware and software modularity is essential to keep complexity under control. There are two aspects to this modularity. One relates to how to build DSN systems that are robust and reliable; the other relates to the development process for complicated systems.

One advantage of DSN systems should be their insensitivity to failures and ability to adapt to the addition of new nodes. The autonomous cooperating process implementation helps to make this possible and is an example of using modularity to achieve robust and reliable operation. This was the approach taken for the DSN tracking system and should also also be applied to other system components including the operating system, communications, self-location, etc.

Autonomous processes require modular hardware to be most effective. A DSN system automatically provides some hardware modularity; individual nodes are physically separated. But even a single node can be a complex system performing many different functions. The design should minimize the number of single point failures that can incapacitate a node. A simple problem that could be easily repaired in a centralized system may be more difficult to repair in a remote node. System reliability requires that failure modes be as small grain as possible. Thus, a single powerful nodal computer is less desirable than a multiprocessor design.

Software modularity as an aid to system development and testing is not unique to DSN systems. But the large number of software components in a DSN and the fact that it is a distributed system make it even more important that modular designs and automated system development tools be used. Even the simple experimental DSN test bed is complex, containing hundreds of modules. A modular design and automated tools were employed and we believe this contributed substantially to the success of the project.

12. SELF-LOCATION ALGORITHM STUDIES

12.1. Introduction

The locations of the nodes in a DSN system must be known. Without knowledge of these locations, the measurements from different nodes cannot be combined to locate and track aircraft. The accuracy to which aircraft can be located cannot be any better than the accuracy to which the node locations are known.

Many options exist for the determination of node locations[30,31]. These include manual surveying; hyperbolic "time difference of arrival" systems such as LORAN and DECCA or the OMEGA system that uses multiple satellite beacons; the more modern satellite-based Global Positioning System, GPS; triangulation from nodes to known beacon positions and generalized triangulation systems such as JTIDS and EPLRS in which already located elements serve as additional beacons. Any of these, or variants, could be used for the DSN application. Different approaches might be called for, depending on the specific DSN under development.

The DSN project concentrated on using internodal range measurements for self-location. The idea was to use the radios in each node to make the internodal range measurements between pairs of nodes and to develop distributed algorithms to provide each node with its own location as a consequence of interactions with nearby nodes.

Of the node location options listed above, the approach emphasized during the DSN research has been most similar to the triangulation approach and generalizations such as JTIDS and EPLRS. However, a major difference is that a goal for the DSN work was to handle situations where triangulation would not work. Figure 12.1 illustrates a simple case of this. The solid lines show radio communication connectivity. Lines radiating from a node terminate at those nodes that can receive direct radio broadcasts from the node. These lines correspond to the only distance measurements that can be made between pairs of nodes in this network. Starting with those measurements and any three of the nodes, it is impossible to unambiguously construct the geometry of the network using triangulation, adding one node at a time. Such problems can often exist in a ground-based DSN system where line-of-sight limitations restrict connectivity. The DSN research objective was to devise techniques that would accommodate these cases as well as the simpler cases when there is sufficient connectivity to support triangulation.

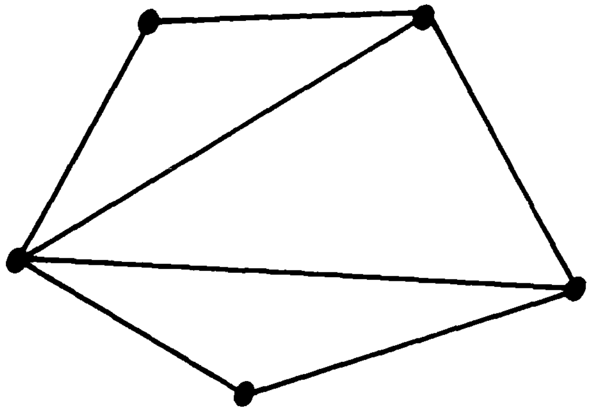


Figure 12.1. Network that cannot be configured by triangulation.

The research performed at Columbia University [30] emphasized how to determine when the set of measurable internodal ranges is sufficient to unambiguously establish the network configuration and how to calculate an estimate of the configuration. They gave less attention to optimal handling of range estimation errors than to obtaining at least approximately correct position estimates. They were less concerned with optimal handling of range estimation errors than with finding configurations consistent with the measurements and detecting when multiple configurations are possible.

Complementary research performed at Lincoln Laboratory emphasized the refinement of node location estimates. The initial estimates might be obtained using algorithms such as those developed at Columbia, although any other mechanism could be used. For example, rough map coordinates might be recorded during node deployment. These estimates, perhaps accurate to only within several hundred meters, could be used as initial estimates.

Section 12.2 summarizes our work on distributed self-location algorithms. Details of the algorithms and further discussion are contained in Reference 32. Section 12.3 describes how radios such as the experimental radios described in Section 8.4 can be used to estimate the ranges required by the node location algorithms.

12.2. Location Algorithm

The DSN network self-location problem has been formulated as a distributed estimation problem, and distributed estimation theoretic algorithms have been developed to estimate the network geometry. The algorithms accept internodal range measurements as inputs, calculate position estimates as outputs, and optimally account for measurement errors. Initial esimates of nodal positions are treated as unbiased random variables with known error covariance matrices.

Several simplifying assumptions have been made to minimize complexity during the development and testing of initial versions of the algorithms. First, nodes are constrained to lie in a two-dimensional plane and do not move. Nodes in three dimensions or on a known two-dimensional nonplanar surface could be treated but involve unnecessary complexity. Second, there are no biases in range measurements. As discussed in Section 12.3, the experimental CNT radio incorporated anti-multipath features to satisfy this assumption. Third, as mentioned above, initial estimates of node locations are available. These must be close enough to the true values so that the algorithms, which maximize a function of the node positions, will not become trapped in a local minimum. Fourth, there is enough communication connectivity to support a solutic. This is less restrictive than requiring that the range measurements are sufficient to determine the network configuration. One way to visualize this is to imagine that the initial node position estimates are equivalent to many additional internodal range measurements.

Subject to these assumptions, optimal distributed algorithms have been developed. The approach is Bayesian. Statistical models for the initial location estimates and the range measurements were used to derive the probability distribution of the node locations. This distribution is then maximized to obtain the location estimates. For these purposes the initial estimates of node locations were assumed to be independent unbiased Gaussian random variables, and the range measurement errors were modeled as zero mean Gaussian random variables added to the square of the true range. This measurement model is not as realistic as adding the errors to the range values, but it was used because we initially thought we could derive closed-form solutions for the node position estimates based upon this approach. It has only been possible to develop numeric algorithms. Since this is the case there is no longer any reason to prefer the less-realistic model.

The DSN self-location problem is complicated because it is nonlinear and a distributed solution is required (i.e., an algorithm that can be executed independently in each node without any centralized control). To develop an understanding of different aspects of the problem a series of sub-problems, leading to the solution of the DSN self-location problem, were investigated. These were centralized and distributed versions of the problem with two forms of measurement equations, linear and nonlinear. The distributed problem with nonlinear measurement equations is the actual DSN self-location problem.

Part a of Figure 12.2 shows a portion of a large network, where each dot is a node and the x_i and y_i are the position coordinates to be estimated for node i. The contour line N_i encloses a set of neighbor nodes for node i, i.e., those nodes that can communicate directly with node i. This contour is determined by the type of radio at the node, the electromagnetic environment and the local geography, and is not known *a priori*. Parts b and c of the figure illustrate the two types of internodal measurements that have been considered. Linear measurement equations are obtained when we assume that the vector distance between a node pair is measurable. The coordinates of the vector difference, with noise added, are the measurements. The linear measurement problem is not of much interest but was an important step toward the solution of the nonlinear problem, which is of interest. Nonlinear measurement equations are obtained when we assume that the distance or the square of the distance between nodes is measurable after the addition of random noise.

The centralized linear problem was easily solvable using Kalman filter methods and was of little general interest. However, the distributed linear problem provioed insight into how to distribute the solution and what information to exchange between nodes. A decomposition of the overall *a posteriori* probability function was found that allowed nodes to locally maximize a portion of the probability function as a function of their own location. The resulting estimate depends on the estimates from other nodes in the network. Each node then continues to revise its own estimate as long as it continues to receive revised estimates from its neighbors and these continue to significantly affect its own estimate. Convergence is assured if the local maximization is completed one node at a time in the network. A convenient property of the solution is that a node need know only the position estimates of nodes with which it can communicate directly, assuming the node can communicate with every node for which there is a position measurement relative to itself. The initial work with the linearized problem provided the insight needed to develop the solution to the more important and difficult nonlinear problem.

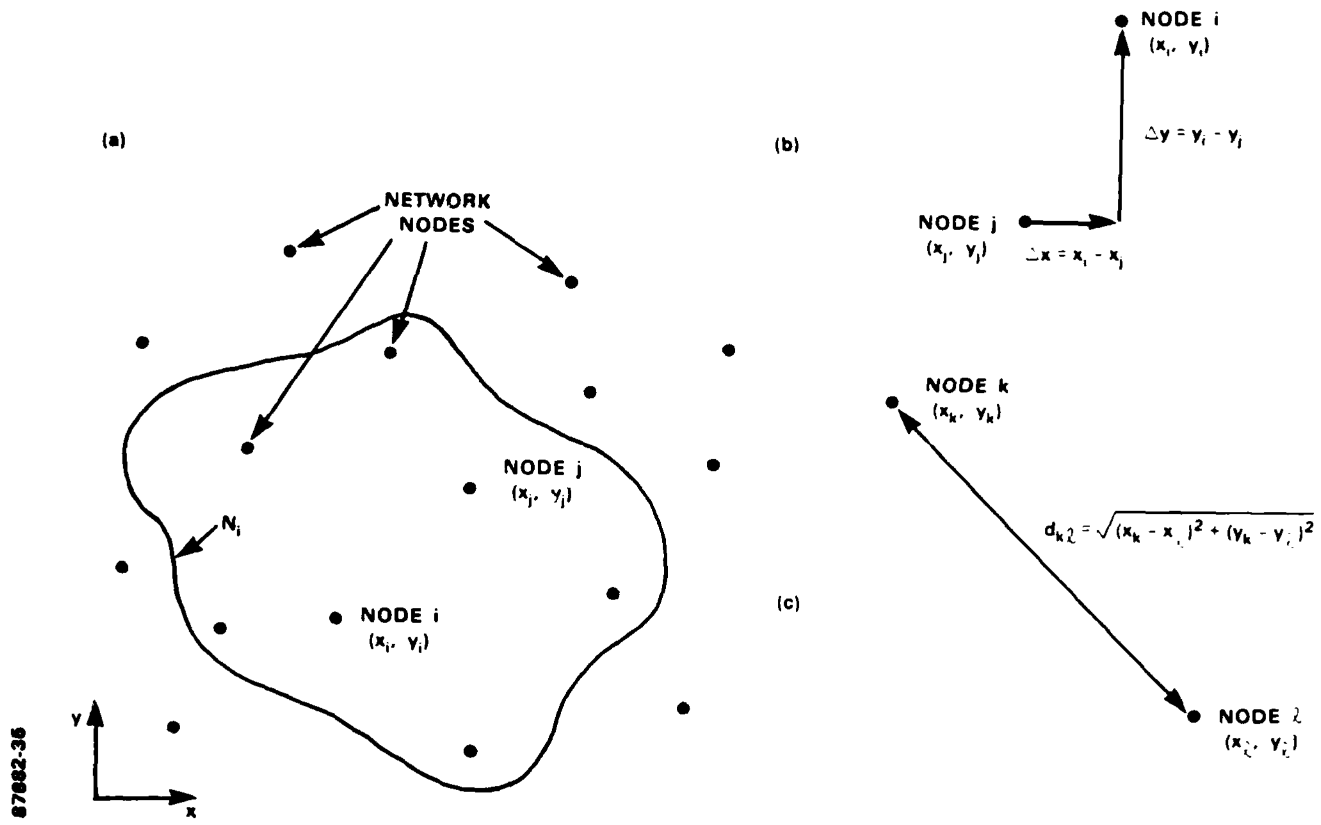


Figure 12.2. Network layout and measurement options. (a) Definition of neighborhood; (b) linear measurement; (c) nonlinear measurement.

As noted previously, we used the square of range measurements as the observations for the nonlinear self-location problem. This results in a fourth-order cost function to minimize with respect to node coordinates. As for the linear case, it was possible to devise distributed algorithms to accomplish this optimization. At each node the derivatives of cost function are taken with respect to its two position coordinates, and set to zero to find the minimum. Other node locations are assumed known for this purpose. This results in two simultaneous cubic equations that must be solved at each node.

Three methods were tried for solving these equations. The most successful was to solve for each coordinate separately with the other held constant at its last value, then iterating between the two until they both converged. The second was Newton's method, which was less successful because it often found local rather than global minima. The third approach was to find a closed-form solution to the equations with the help of MACSYMA [29], but the resulting formulas were too complex to be practical.

The overall procedure is for nodes to exchange location estimates with neighboring nodes and repeat the above process until the solutions converge. As in the linear case, convergence is assured only if estimates are revised one node at the time. Once again it is convenient that the set of neighboring nodes, from which node i must obtain location estimates, is the same set for which it is processing internodal measurements.

Distributed self-location algorithms, including the communication policy, have been implemented under UNIX on a VAX computer to investigate their behavior. Both linear and the non-linear distributed algorithms were implemented. The behavior of the algorithms was tested as a function of the standard deviation of initial node position estimates, the standard deviation of internodal measurements, and a parameter controlling convergence. Random ten-node networks were used to test the algorithms and performance statistics were averaged over ten runs with each network for different values of prior and measurement error standard deviations. For the linear problem one node was taken as a reference node to avoid drifting in absolute coordinates. For the nonlinear problem two reference nodes were needed to avoid translation and rotation. (There was nothing special about the reference nodes other than that very small standard deviations were assigned to their position guesses.) All cases behaved much as one would intuitively expect, taking more time to converge when connectivity was low, or when errors were large.

Table 12.1 is an example of the results. It shows the performance statistics for the network in Figure 12.3, a network in which none of the nodes can communicate with all of the reference nodes, so no node is performing triangulation from three known positions. The trends toward longer convergence times and larger final errors for larger initial position and measurement error covariances are clear. The apparent exception in the bottom line is due to statistical fluctuations; the results of only ten runs were averaged to obtain the statistics shown.

TABLE 12.1

Algorithm Performance for Network in Figure 12.3

Measurement Standard Deviation (m)	Initial Position Standard Deviation		
	5 m	50 m	500 m
150	3[1] 6.4[2]	4.6 57	25.2 300
15	4 5.8	17.6 35	37.2 48
1.5	11.4 3.5	29.2 5.3	43.8 4.4

NOTES: (1) Average number of algorithm cycles per node
(2) Final position error standard deviation (in meters)

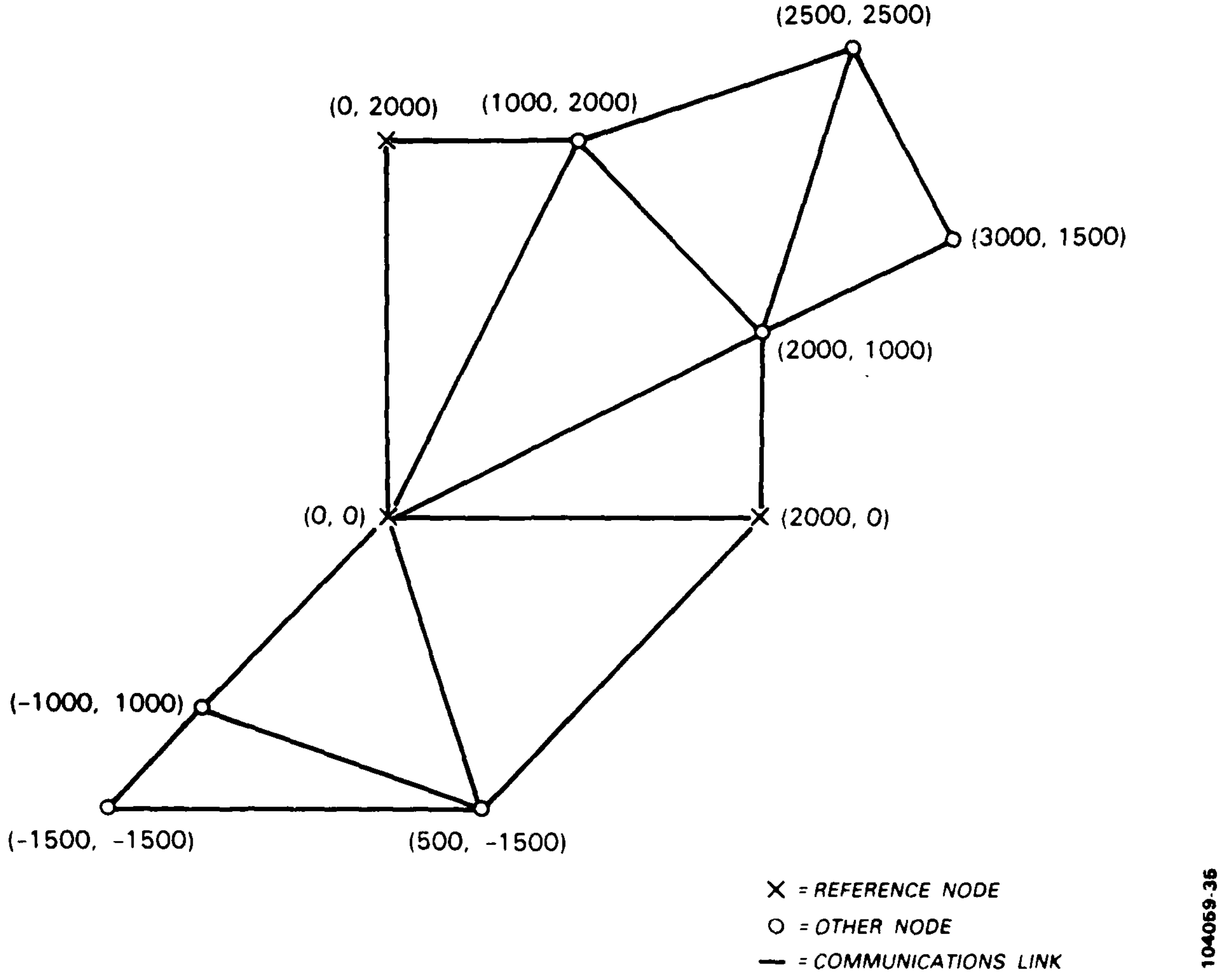


Figure 12.3. Sparsely connected network used for self-location tests.

12.3. Internodal Range Measurement

Distributed network self-registration using internodal range measurements requires that each node know the number and identities of its neighbors and that there be a way to measure the distance between all neighbor pairs. Mechanisms for obtaining this information were not incorporated into the DSN test bed but are available.

How to determine the number and identities of neighbors is a common problem for all packet radio networks. The information is used to route data packets through the networks. The mechanism that has been developed for packet radios networks is for each node to occasionally broadcast a Packet Radio Organization Packet (PROP) [33]. The PROP identifies the sending node and certifies that it is operational. Based upon the PROPs that it has recently received, each node can maintain up-to-date tables listing its neighbors.

The PROP mechanism, in some form, will be required in DSNs to support the communication network, self-location and, indirectly, tracking algorithms. One option might be to employ the periodic tracking message broadcasts for this purpose since each node transmits a tracking message every few seconds. This is probably not a good approach since it intimately links distinct system functions and violates the design principles of modularity and layering. The PROP messages, although this adds to the communication traffic, should be distinct from all other messages in the system. The cost will be more than compensated for by the resulting system simplicity and reliability.

Basic protocols for collecting and managing network node and communication connectivity information are known. Nevertheless it is worth noting one issue that arises when the radios are pseudonoise spread-spectrum radios that can be operated with different code seeds for each message. (The experimental radios described in Section 8.4 are of this type.) At any given time a radio can "listen" for incoming messages using only one code seed. If all radios in the network use the same seed this causes no problem and all PROPs will be received, subject to failures resulting from message collisions. If the code-changing feature is utilized, then PROPs will be missed unless the sender and the receiver are using the same seed during the transmission time interval.

The code changing problem is most severe when a node first enters a network. At that time there is no established communication link and the nodes may be poorly synchronized in time. Options available are to not change code seeds, to change them on a coarse time scale commensurate with the poor time synchronization or to require the established network to occasionally listen for PROPs from new nodes using a standard code that is agreed upon for the purpose of finding new network members. Once the first PROP is received from a new node the problem is substantially reduced. Time stamp information in the PROP can be the basis for bootstrapping the node into the network with any degree of time and code synchronization that is desired.

Once communication links have been established, and a node has a list of neighbors, it can measure one-by-one the distances between itself and each of the neighbors. This can be done by measuring the time for a communication signal to pass between nodes, provided the time for a direct line-of-sight communication path can be measured.

A signal transmitted by Node A may arrive at Node B by several paths, a direct path plus multipaths corresponding to reflections and refractions of the transmitted signal. The signal along the direct path will arrive first. The others will arrive later by an amount proportional to the extra distance traveled by the wave. A multipath signal is shown in cartoon form in Figure 12.4.

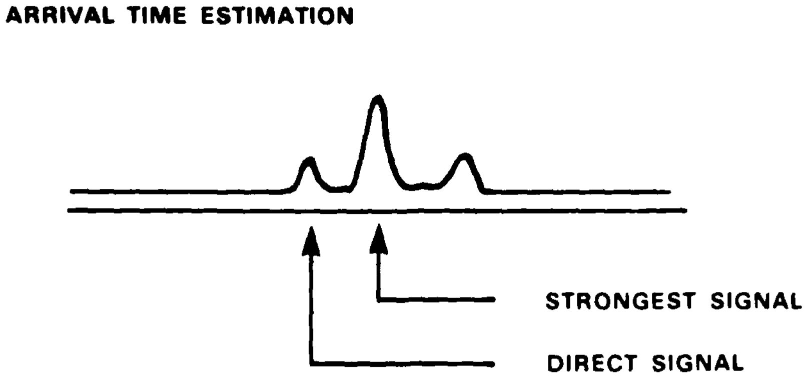


Figure 12.4. Illustration of strong late multipath arrival.

12.3. Internodal Range Measurement

Distributed network self-registration using internodal range measurements requires that each node know the number and identities of its neighbors and that there be a way to measure the distance between all neighbor pairs. Mechanisms for obtaining this information were not incorporated into the DSN test bed but are available.

How to determine the number and identities of neighbors is a common problem for all packet radio networks. The information is used to route data packets through the networks. The mechanism that has been developed for packet radios networks is for each node to occasionally broadcast a Packet Radio Organization Packet (PROP) [33]. The PROP identifies the sending node and certifies that it is operational. Based upon the PROPs that it has recently received, each node can maintain up-to-date tables listing its neighbors.

The PROP mechanism, in some form, will be required in DSNs to support the communication network, self-location and, indirectly, tracking algorithms. One option might be to employ the periodic tracking message broadcasts for this purpose since each node transmits a tracking message every few seconds. This is probably not a good approach since it intimately links distinct system functions and violates the design principles of modularity and layering. The PROP messages, although this adds to the communication traffic, should be distinct from all other messages in the system. The cost will be more than compensated for by the resulting system simplicity and reliability.

Basic protocols for collecting and managing network node and communication connectivity information are known. Nevertheless it is worth noting one issue that arises when the radios are pseudonoise spread-spectrum radios that can be operated with different code seeds for each message. (The experimental radios described in Section 8.4 are of this type.) At any given time a radio can "listen" for incoming messages using only one code seed. If all radios in the network use the same seed this causes no problem and all PROPs will be received, subject to failures resulting from message collisions. If the code-changing feature is utilized, then PROPs will be missed unless the sender and the receiver are using the same seed during the transmission time interval.

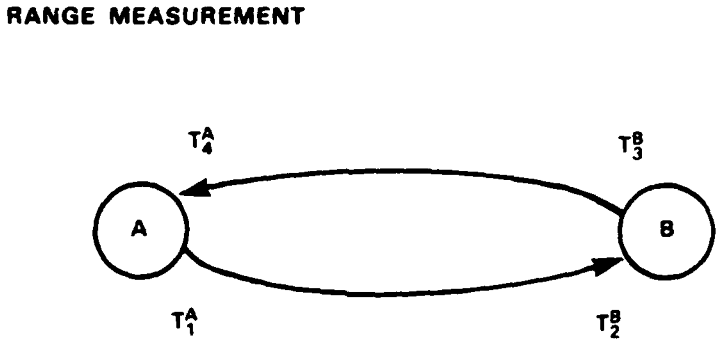
It shows the received signal strength time history from a single transmitted bit. There are three distinct arrivals, corresponding to the direct path and two multipath arrives. Moreover, the strongest signal is a multipath signal, not the direct path signal. Using the time of the strongest signal to calculate the range would introduce a bias which would make the distance between the two nodes appear to be larger than it really is.

The radios described in Section 8.4 contain many features designed to detect and measure the time of the direct signal in the presence of larger multipath signals. There may still be situations in which the direct path is very weak, or even nonexistent, but those should be exceptional situations, probably occurring only in urban environments where buildings block and reflect signals. In other environments the direct path should be detected. The derived internodal range measurements will then be unbiased, as we assumed for our work on self-location algorithms. But there still is a possibility of situations that create multipath errors. This must ultimately be acknowledged when developing self-location algorithms based upon internodal range measurements. It might be possible to formulate a model for the multipath errors and derive optimal self-location algorithms. It is probably adequate to introduce heuristics to detect multipath measurement errors and to discard the effected measurements. Since the errors will be large, their detection should be straightforward and the loss of precision from eliminating the measurements will be small.

If nodes are precisely synchronized in time, the time of flight from Node A to Node B can be obtained by transmitting a time-stamped message from A to B. B can subtract the reception time from the transmission time contained in the message and obtain the travel time. B can then provide this information to A via a second message or the process can simply be repeated. DSN nodes will not generally be synchronized to the degree required for this simple approach.

If nodes are not precisely synchronized there are two options for range measurement. Suppose that Node A initiates a range measurement with Node B. One option is for B to act as a repeater for the message received from A, with a fixed known delay between reception and transmission. The other is for B to behave as a more sophisticated repeater for which the delay between reception and transmission is not fixed. In this case, B does not behave strictly as a repeater but transmits a message containing delay time information.

Neither of these options has been implemented for the DSN test bed or the Communication Network Technology (CNT) radios (Section 8.4), but it is clear that there are no fundamental difficulties. Figure 12.5 illustrates a range measurement protocol for the CNT radios. The radio design does not permit them to be used as simple repeaters with a known and fixed delay but message arrival times can be measured. Transmission times can be specified precisely, but in terms of clocks that are not synchronized. The time between a reception and a transmission triggered by the reception is variable, depending upon the time required to process the incoming message and prepare the outgoing one. As shown in the figure, the range R is estimated from the initial transmission time at A (T_1^A), the reception time of that message at B (T_2^B), the later time at which B responds (T_3^B), and the still later time when A receives the response (T_4^A). C is the speed of light. An important thing to note is that times with a superscript A are measured by the clock at Node A and times with a superscript B are measured by the clock at Node B. No synchronization is required, although the two clocks must tick at approximately the correct rate. The rate must be accurate enough so that the total round-trip time from A back to A will be measured with an accuracy commensurate with the desired accuracy for the range measurement. This is easy to achieve.



$$\frac{2R}{C} = (T_4^A - T_1^A) - (T_3^B - T_2^B)$$

Figure 12.5. Determination of range using time-stamped messages.

13. MULTISITE DATA-INTEGRATION STUDIES

13.1. Introduction

Each node in a DSN system maintains tracks for targets within range of its own sensors. This area of responsibility may extend somewhat beyond the sensor range but will not normally extend to include the sensor coverage areas of all nodes in the network, except perhaps for small networks containing only a handful of nodes. Whatever the size of the area of responsibility, adjacent nodes may contain tracks for the same target. Some of these redundant tracks may differ from each other in detail and the set of nodes with knowledge of the target will change as the target moves through the network. Redundant tracks must be merged together and users should be provided with nonredundant "full-network" tracks. This requires collecting and merging information from multiple sites and providing the integrated result to the user in a useful form.

A multisite track integration algorithm and a user query and display system have been implemented and used for preliminary investigation of multisite integration issues. The multisite integration algorithm, which has been used with prerecorded helicopter data and with simulated data for one and two helicopters and up to eight nodes, was implemented as multiple communicating processes on a single UNIX system. The user query and display system was implemented on a Silicon Graphics, Inc. (SGI) UNIX workstation with a color display. The objectives were to experiment with a simple approach to multisite integration and, based upon that experience, to formulate multisite integration ideas for future DSN systems.

13.2. Multisite Integration Algorithms

An acyclic data integration tree, rooted at a network user, can be used to collect surveillance information in a DSN. The nodes of the tree are the nodes that might provide useful information to the user. The branches represent information flow. Tracks are sent inward from the leaves, with nodes combining redundant tracks as redundancies are detected. Track files presented to the user have had all redundancies removed and contain tracks from all the nodes in the network.

Our experimentation with multisite integration has concentrated on exploring the use of acyclic data integration trees . Standard track identifiers (see Section 6.2.8) are used to determine when tracks are redundant. The experimental multisite integration algorithm uses the position track combining algorithm described in Section 6.2.5. The track combining algorithm is used at each node to merge local track information with information received from other nodes. The result is sent out to the next node in the routing tree. Tracks that are new to the node are passed on without modification. Eventually, the information reaches the user.

Like the tracking system, the multisite integration system functions on a fixed time cycle. The tracking nodes operate with update periods of 2 s. The multisite integration algorithms operate with an update period that is an integer multiple of the tracking period. Integration processing is set in motion during the same tracking cycle at all network nodes. The start time and length of the integration period are parameters of the integration processes. The integration process in each node gathers tracks from the tracking process in the node at the start of the first integration cycle, and multisite data collection is begun at the leaf nodes of the multisite integration routing tree. Multisite integration messages propagate inward from the leaf nodes to the user. The combining algorithm at each node is triggered by the arrival of an integration message from another node. As soon as all the expected integration messages have been received and processed, the node transmits an integration message to the next node in the tree. Time-outs are used to trigger outputs to the next node when expected inputs have not been received within a reasonable time. When a time-out occurs, the node sends along whatever information it has and waits for the start of the next multisite integration cycle. Integration messages summarize the aircraft traffic status at the start of each integration cycle.

The structure and contents of data integration messages are similar to those of normal target tracking messages, with additions to provide extra multisite integration information. Normal tracking messages contain the identity of the sending node, the message origin time at the transmitting node, local and foreign state vectors, and error covariance matrices. The message origin time for normal tracking messages is also the time corresponding to the position and covariance estimates in the message. Multisite integration messages contain separate message and track times. The message time for a multisite integration message need not be the time corresponding to the position and covariance estimates in the message; generally it does not. The time corresponding to the track estimates is explicitly contained in each message. It is the start time of the multisite integration cycle during which the tracks are reported to the user.

The multisite integration messages also contain lists of nodes contributing to each track. At the start of an integration cycle each node initializes its multisite integration message. For each track, the list of contributing nodes is initialized with one element, the identity of the initializing node. A list is initialized for each track reported to the multisite integration process by the tracker. Subsequently, whenever two multisite integration messages are combined, the lists of contributing nodes are merged.

In addition, at the start of each multisite integration cycle, a list is initialized of nodes expected eventually to contribute information for the user. These lists describe the multisite data integration routing tree. Nodes are deleted from the lists as they contribute messages. When a multisite integration message eventually reaches the user, the list will have been reduced to a list of nodes failing to report during that multisite integration cycle. Note that a failure to contribute a data integration message is independent of whether a node detects or tracks any specific target. Even if a node has no targets to report, it should provide a timely data integration message if the node is part of the multisite integration tree.

Integration algorithms have been implemented as a main control process and a group of other processes, each representing a process at a different node in the multisite integration tree, that all execute on a single UNIX system. The node processes are interconnected by pipes to form the multisite integration route tree. The main process reads parameter files, translates parameter files into messages, and executes the node processes. The main process also sends messages to the node processes to specify algorithm and route parameters. Algorithm and route parameters are input to the main process in the same format as the parameters sent from the UIP process to the tracking process. Each line in the parameter file contains a node number, the process it is used in (in this case multisite integration), a keyword, and a value or values corresponding to the keyword. The algorithm parameters specify the timing of the multisite integration algorithm as well as uncertainty and threshold values required for track combining. The routing parameters specify the network route for multisite data collection and integration. In addition, the main process creates the interprocess pipes according to the route information.

The multisite integration algorithms were tested using data previously recorded on nodal floppy disks during multinode tracking experiments. The data from the disks were transferred to UNIX files, one for each node in an experiment. The node processes read track information from these files and also receive multisite integration messages from other nodes through the pipes. Each node reads its input pipes, performs multisite integration, and writes to its output pipe. Multisite integration communication failures are simulated by filtering the internodal pipes to remove some of the messages. The nodes detect missing messages by monitoring message time stamps and take the appropriate action, which includes no further waiting for old data and adding the missing message information to its output integration message. The last node in the integration route produces the multisite integration messages for the user. The user messages are saved in a file and later transferred to an SGI UNIX workstation where they are displayed.

13.3. User Display and Multisite Integration Examples

A multisite query and display system has been implemented on a SGI UNIX workstation. It provides several useful displays and options. These include displays of present target locations and past histories, node locations, the data integration routing tree, network communication connectivity, and nodes contributing to specific tracks. The display can be zoomed in or out on areas of interest. Track points are centered within error ellipses derived from track covariance matrices contained in the multisite integration messages. The ellipses indicate the size and shape of estimated track uncertainties. The display is similar to that used for real-time test-bed experiments but has several additional features.

The display system requires several input files in addition to the multisite integration message stream. These include files that contain messages that specify node locations, communication ranges, and sensor detection ranges. These messages were previously defined and used in the tracking software. It also requires multisite integration route specification messages.

The display program is interactively controlled using menus and a three-button mouse. The buttons invoke functions such as changing menus, selecting objects, recentering the screen, and zooming in and out. They are also used to toggle options, such as track vectorization, communication range, detection range, track history, and route depiction. The program can display normal target tracking messages during tracking experiments, as well as display multisite integration messages.

Figure 13.1 shows an example of the situation display. Node locations are shown as small yellow dots. The data integration tree is indicated by the long blue isosceles triangles. Each triangle is an arrowhead pointing from a sending to a receiving node in the multisite integration route

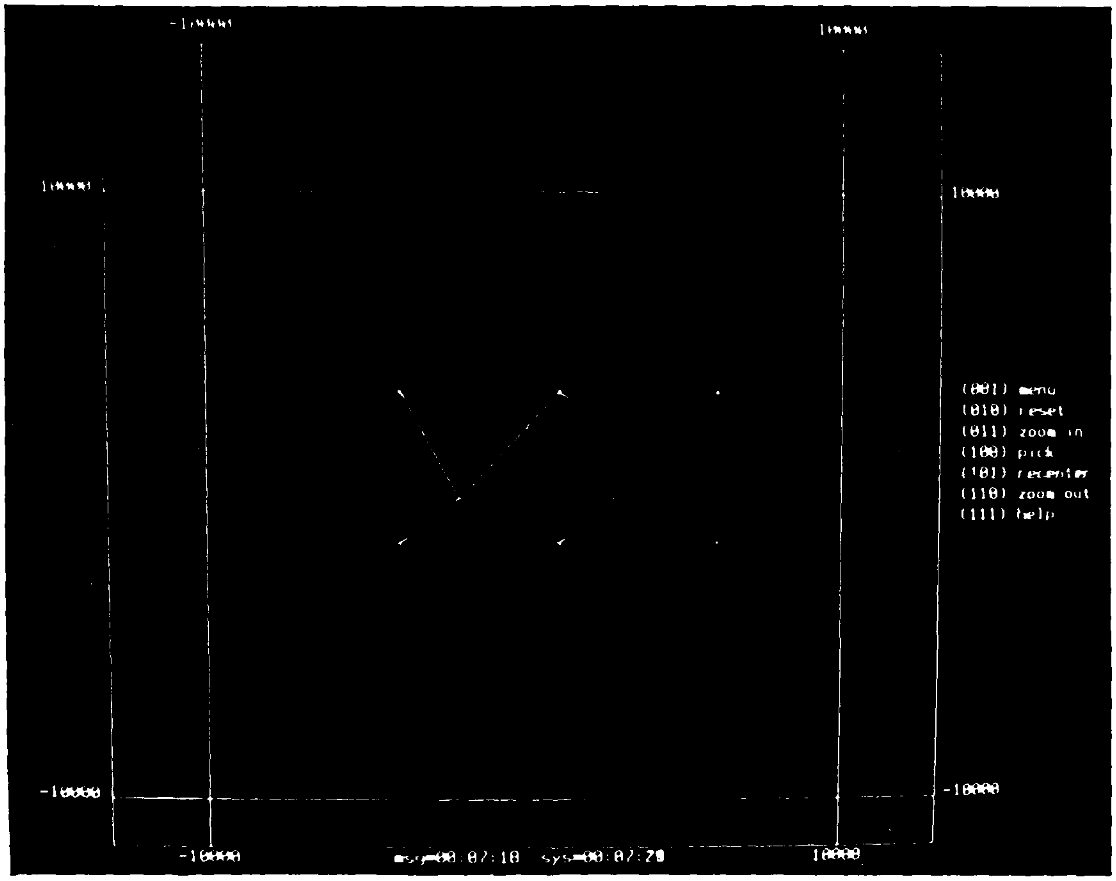


Figure 13.1. Example of a situation display showing six nodes, two tracks, data-collection route, and the nodes contributing to the tracks.

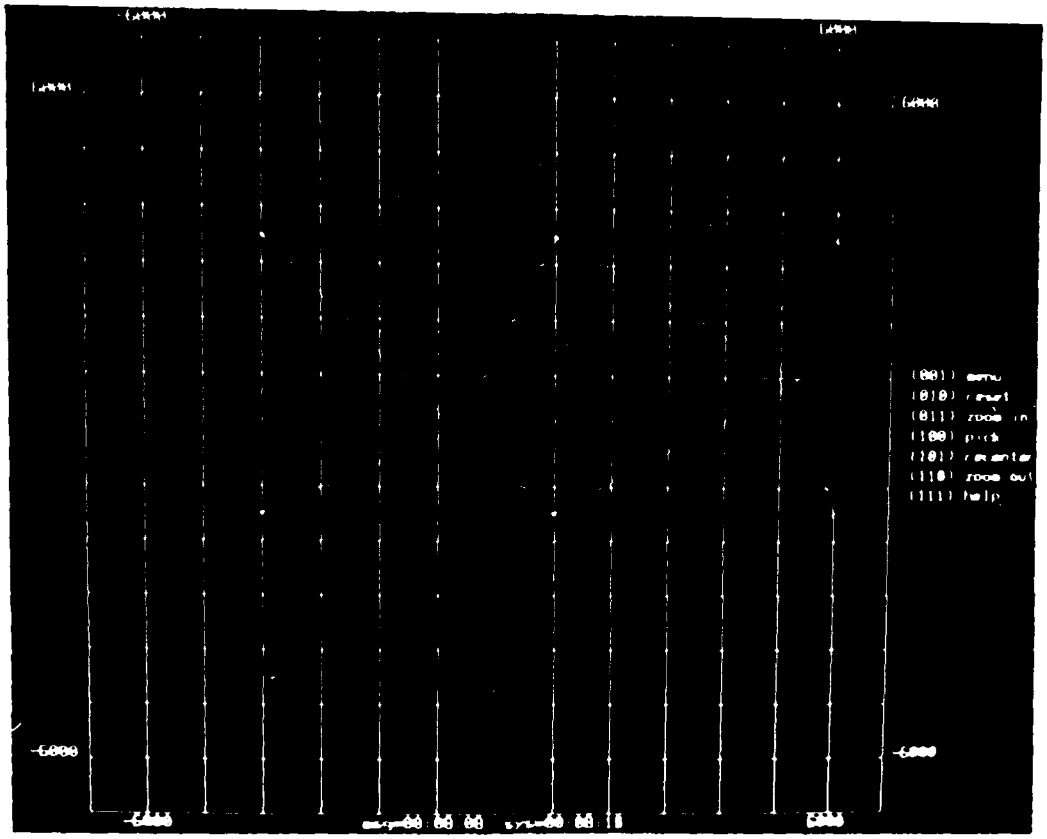
tree. Also shown are 30-s track histories for two targets flying from the left to the right. These are the two series of four small red error ellipses. In the case shown, the integration cycle was five times the basic tracking cycle, so the time between ellipses is 10 s. The straight yellow lines radiating from tracks to nodes identify which nodes provided information for each integrated track. The text at the right side is a help menu to remind the user how to use the mouse buttons. Other information on the display includes a labeled geographic grid and two alphanumeric strings that show the present time and the time of the last received track integration message. The small red arrow is a cursor used to pick options from other menus and select targets or nodes. Its position is controlled by the mouse.

Figure 13.2 shows a display similar to that of Figure 13.1 except that the display has been zoomed in and is for a later time during the same experiment. Note that the node in the upper-right corner is not contributing to the right-most track although it is close enough to be within detection range. Although it is not shown on this display, the user could also request the detection range of the sensor, shown as a circle centered on the node, to confirm that the target is indeed within normal detection range.

The display in Figure 13.2 can be used to infer that there is a malfunction in the tracking system at the upper-right node, although the node is performing its multisite data collection and integration functions. This requires knowing that as long as messages are being received from a node, the route triangles are solid as illustrated in Figures 13.1 and 13.2; but, when messages are not being received the route specification triangles are shown only in outline as in Figure 13.3.

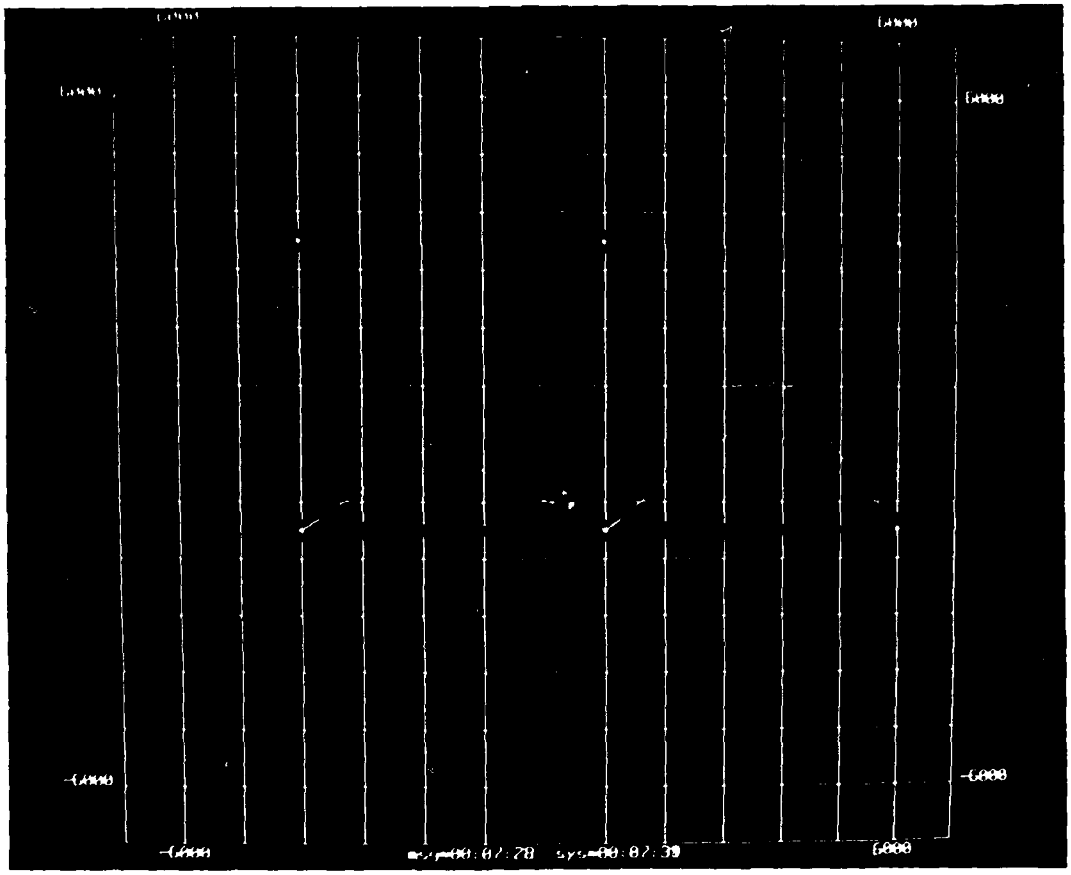
The situation depicted in Figure 13.3 is more catastrophic than that in Figure 13.2. The user, located at the lower-right-hand node, is receiving no multisite integration messages from the node at the upper-right. Therefore, no information can be collected directly from any of the upper three nodes. The user knows only that the multisite integration messages from the upper right-hand node are not being received. This is true even though two more of the yellow lines from tracks to nodes are missing. We cannot infer from the missing yellow lines that the other two nodes have failed. The tracks from the lower nodes may contain information from the upper nodes. We cannot know for certain because the information transfer from the upper to the lower nodes, if any, is through normal tracking messages, not multisite integration messages.

Returning to Figure 13.2 we see that all communication links are working, that two of the nodes are contributing normally to the tracks, and one node is not. Thus, we conclude that there



104059-38

Figure 13.2. Situation display example at a later time and zoomed in.



104059-40

Figure 13.3. Situation display showing a communication failure.

is a problem with the tracking process in the noncontributing node. This is a simple illustration of how a well-designed user interface will help detect and diagnose failures as well as provide surveillance information.

13.4. Discussion

Multisite integration is an important and complex problem for large-scale DSN systems. Our preliminary experimentation is only a first step toward the development of a practical system. But based upon that work a few observations can be made. First, it appears that the algorithm used for track combining within the tracking system can also be used for multisite integration. Second, users and system operators should be provided convenient access to information such as node positions, sensor capabilities, communication connectivity, routing trees, contributing nodes, etc., in addition to tracks. Such information may increase user confidence in results, identify erroneous results, and help operational personnel identify system faults. Third, acyclic multisite integration routing trees are the only practical multisite integration approach that we have been able to identify, and there remain many unanswered questions about how to establish and modify these trees, about the timeliness of the information provided to the user, and about the interplay between multisite integration and the underlying communication system. Following is additional discussion of the first and third of these points.

It is not surprising that the track combining algorithm used by the tracker can also be used for multisite track integration; the problems are similar. But, just as the tracker may make track association errors because it relies only on track identifiers for that purpose (see Sections 6.2.10 and 10.4.), so will the multisite integration procedure. More sophisticated track association algorithms are required for multisite integration as well as for basic tracking. Every trick or method used by the tracker should be included in the multisite integration process.

Imagine a situation in which an aircraft can be tracked for a long period of time but, for some reason, the track is broken into two or more segments, with different track identifiers. The real-time tracker will probably not be aided much nor its real-time tracking output improved much if, in retrospect, it becomes possible to associate the fragments with a single aircraft. But a user might be interested to know that there is a single target rather than several. If this is the case, higher-level track association algorithms will be required for future DSN systems. Both statistical and knowledge-based techniques might be used as the basis for the algorithms.

Data integration trees appear to be a practical and direct approach to data collection and integration. But several important unanswered questions remain concerning the automatic use of routing trees and the relationship between the multisite integration process and the DSN communication system.

The preceding discussion of multisite integration assumed that the routing tree was static but subject to damage resulting from nodal failures. No algorithms for selecting integration trees or mechanisms for reacting to failures were discussed. Both of these issues need more attention.

Given knowledge of the nodes of interest to the user and the communication connectivity of the network, it is not difficult to devise reasonable algorithms to select routing trees. The problem is more difficult if there are multiple users within the network and either processing or communication resources are constrained. It is in this context of constraints that algorithms to select data collection trees are still an issue. Such algorithms are also required if the system is to react automatically to failures requiring rerouting of information.

The situation illustrated in Figure 13.4 is an example of the complex relationship between multisite integration and communication. The solid lines represent a multisite integration routing tree. The two hash marks on the link between nodes 2 and 5 on Figure 13.4(a) represent a failure of the link. The dotted links represent an automatic response of the communication system to the failure. The communication system has automatically invoked a two-hop link to continue to supply the service from node 2 to node 5. This may be reasonable for the communications system, but a better solution would be to change the tree to the one shown in Figure 13.4(b). This configuration will require less communication resources. Automatic communication rerouting can be thought of as a temporary measure to use until the multisite integration routing tree is reconfigured. But there is a possible trap. If the communication rerouting is too effective, the multisite integration system might never be aware of the communication problem and never make any changes. The multisite system should be informed of changes in the communication system so that it can reconfigure itself in response to the changes.

Another issue is the timeliness and quality of the track information provided to users. In our experimental system the user receives track estimates that can be delayed by as much as a full data integration processing cycle; the report to the user is always a snapshot of the system traffic

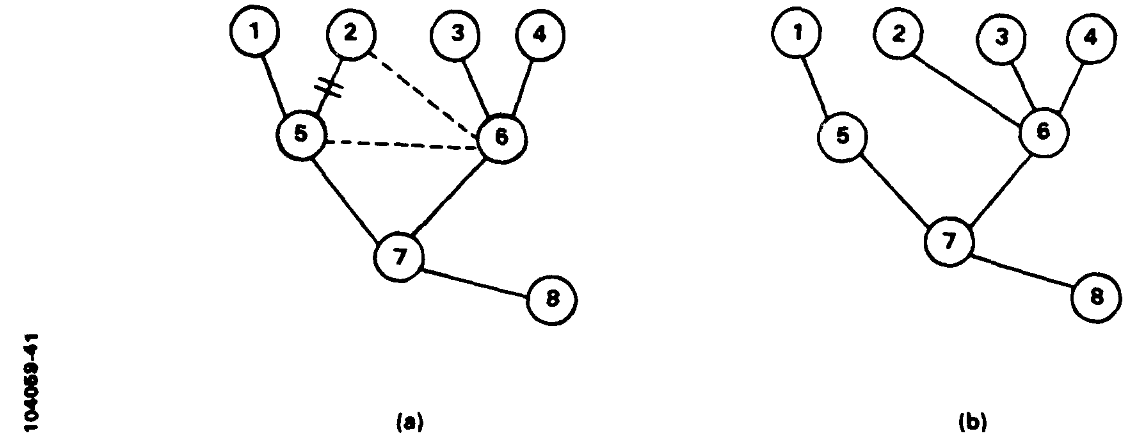


Figure 13.4. Illustration of communication and integration interrelation. Solid lines are integration routing trees. (a) Communication link failure between 2 and 5 has caused the communication system to route integration traffic from 2 through 6 to 5. (b) Redesigned integration routing tree that accounts for the link failure between 2 and 5 and reduced overall traffic.

at the start of the cycle in which the report is received. If the cycle is large, say 10 or 20 s, this delay may be unacceptably large. The track can be extrapolated forward to the present but the quality will suffer. An alternative would be to make the report to the user as current as possible, subject to the constraints of processing and communication delays. This can be easily accomplished. Rather than collecting track information from the tracker at the same time at each node, the integration process at a node could obtain information from its own tracker only when it has received all incoming integration messages and it is prepared to calculate and transmit its outgoing integration message. The most recent information would then be included in the report delivered to the user. The information for nearby targets would be as up-to-date and accurate as possible, a generally desirable feature.

Because of an implementation detail, our experimental multisite integration system would introduce substantial delays in the delivery of information from remote nodes in the network. The detail is that the tracking system, the multisite integration system, and all internodal communication are driven by the acoustic signal processing cycle, usually 2 s. The node output triggered by the arrival of multisite integration messages can occur only after the start of the next signal processing cycle. The result is that a report from a node N communication hops distant from the user is delayed by at least N acoustic signal processing cycles; usually 2N seconds. Also note that the integration cycle must be at least N acoustic processing cycles in duration or the information will not arrive at the user.

Although our implementation exacerbates the delay problem for distant nodes, additive multihop delays are a fundamental consequence of multihop communication and serially linked computational processes in a DSN. If the computation and message handling time at a single node is T seconds, then the information delay incurred when passing though N nodes is at least NT seconds. The result is that the delay grows rapidly. The growth in T could be reduced substantially by breaking the hop-by-hop link to the next acoustic processing cycle. The growth can probably be reduced to acceptable levels, even for very large networks, but it is important to realize that multisite data integration delays must be considered during the design of large-scale DSN systems.

We have not attempted to address all the alternatives or issues related to multisite data integration in large-scale DSN systems. The discussion here has been limited only to a few specific issues that arose during our preliminary multisite data integration investigation.

14. KNOWLEDGE-BASED SYSTEM DIAGNOSIS

A future DSN might contain cooperating knowledge-sources distributed among network nodes, interacting to diagnose system failures and to control the system. The Lincoln Laboratory DSN effort included preliminary research aimed at using artificial intelligence methods to automate system diagnosis and control in response to evolving situations. However, our research focused only on knowledge-sources for individual nodes, with the mechanisms for cooperation between knowledge-sources left as a subject for future research as well as being the topic of separate distributed problem solving research by another DSN contractor[34]. Even more specifically, we considered only the problem of reasoning about the adjustment of signal processing parameters in the acoustic signal processing subsystem. The work was done in close cooperation with the knowledge-based signal interpretation research [35,36] undertaken by the Digital Signal Processing Group within the M.I.T. Electrical Engineering Department. Each group investigated the use of multiple levels of signal abstraction, but in different ways.

To gain a perspective for the reasoning required to adaptively adjust signal processing parameters, we first examined how we selected parameter settings during the development of the DSN test bed. We observed that parameter selection generally involved an initial guess, followed by adjustments derived from analyzing the discrepancies between the system output and our knowledge of the actual scenario. An important part of this process was reasoning about the relationship between the system output, the specific system parameter settings, and the test scenario. We hypothesized that this diagnostic reasoning process could be an important element of an automatic adaptive system and decided to investigate it further.

Diagnostic reasoning using discrepancies requires mechanisms for detecting discrepancies between the system output and the true situation. Once a discrepancy has been detected, a knowledge-source can be triggered to perform the diagnostic reasoning to determine what is responsible for the problem, but first the the existence of a problem must be detected, even when there are uncertainties in the knowledge of the true situation.

An experimental system using this approach, discrepancy detection followed by diagnostic reasoning to identify parameter settings responsible for the discrepancy, was implemented on a Symbolics 3600 LISP Machine. The primary system inputs are the outputs from the DSN signal processing system and an approximate description of the true situation. If a discrepancy is detected, the system produces an explanation. The explanations are similar to those that would be provided by a human expert familiar with the underlying theory for the DSN signal processing algorithms.

The experimental system assumes the existence of an independent information source that provides scenario descriptions for discrepancy detection. There would not be such an external information source in a real DSN environment. Fortunately there are alternative, internal DSN sources of scenario information that can be used for discrepancy dete^tion in that case. Four have been identified. These make use of: (1) alternative views, (2) alternative processing, (3) comparison with predictions, and (4) empirical consistencies. None requires human intervention or an independent information source distinct from the DSN system itself.

The discussion of the experimental diagnosis system, which requires an external information source, is continued below following a brief description of the four discrepancy detection alternatives that do not require an external source. This digression is to show how the ideas embodied in the experimental system might be used in a DSN.

(1) *Alternative Views:* Nodes in a DSN have overlapped areas of coverage and can provide different nodal views within an area of common coverage. The output of one node can be used as the source of scenario information to detect discrepancies in the output of another node. The role of the two nodes can also be reversed and heuristics used to decide which of the resulting diagnoses is more plausible.

(2) *Alternative Processing:* The signal processing system developed for the DSN test-bed nodes is an acoustic direction finding system that does not exploit the spectral features of the signals. Other algorithms, such as recognition algorithms for identifying the number and types of aircraft, might also be included in a node. Their outputs could be compared with outputs from the direction finding system for the purpose of discrepancy detection.

(3) *Comparison with Predictions:* Past system outputs can be used to predict future outputs for comparison with actual future outputs. The predictions may be based on knowledge about typical flight characteristics for various types of aircraft in different situations, or may be simple extrapolations of aircraft position using models such as those described in Section 6 for tracking algorithms.

(4) *Empirical Consistencies:* Simple empirical knowledge can also be used to identify unlikely output patterns as discrepancies requiring explanation. For example, short-duration azimuth tracks often do not correspond to real sources. Instead, they can be artifacts of the processing that produces the tracks. Thus, one might assume that the longer tracks represent true tracks while the shorter ones are discrepancies needing explanation.

The experimental diagnosis system models the DSN signal processing system as a set of processes that transform a qualitative description of the actual scenario into a description of the signal processing output [37,38]. In the absence of a discrepancy between the scenario and the signal processing output, each of the processes reduces to an identity transformation. System faults are viewed as being caused by one or more of the processes becoming nonidentity transformations. Diagnosis involves finding a set of nonidentity processes responsible for discrepancies. The processes each embody theoretical knowledge of some phenomenon and its relationship to specific algorithm parameters.

The diagnosis system makes extensive use of knowledge from the underlying Fourier theory of the DSN signal processing system. This knowledge is organized as a *search-space* and a set of *operators* . The search-space is a set of abstract *states* which represent possible DSN system outputs. The operators represent the processes that can affect the DSN signal processing output. The diagnosis system uses means-ends analysis to select operators that together can transform a state consistent with the actual scenario into a state representing the observed DSN signal processing output. The collection of operators and the intermediate data states generated in this process constitute the explanation [37]. The explanation is presented diagrammatically to the user as well as in text form. Figure 14.1 shows an example of the graphical output.

The experimental system represents each state as a collection of *signal-objects* , each corresponding to an acoustic source at a particular time. Each of the signal-objects is further characterized by its spectral characteristics (e.g., bandwidth, amplitude, energy) as well as dynamic characteristics (e.g., velocity, track history). Each state can also be described at four different levels of abstraction; the highest level of abstraction hides the most detail. State parameters are specified by a range of values rather than by a single value. Thus, for example, an aircraft whose direction is unknown can be represented as having a direction in the range from 0 to 360°.

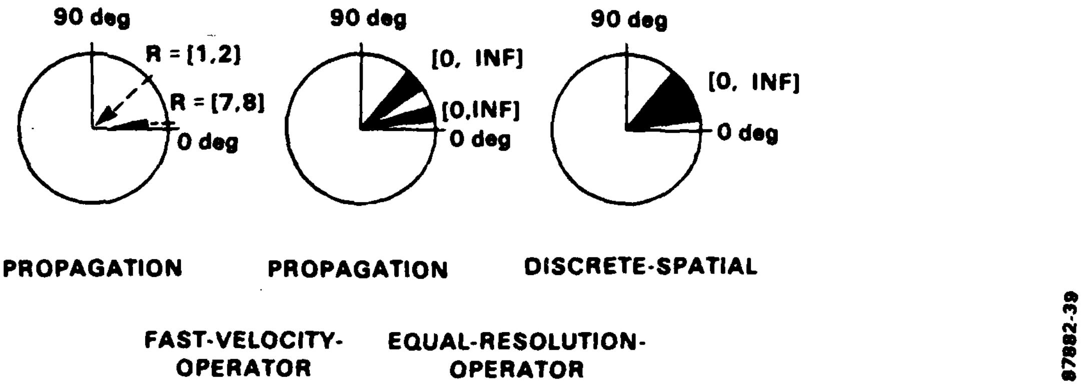


Figure 14.1. Sample output from diagnosis system. Shaded sectors represent signals. Angular extent represents angular uncertainty. Radial extent represents signal bandwidth. Signal amplitude range is given by bracketed numbers. Figure illustrates an initial state transformed first by a fast-velocity operator and then by an equal-resolution operator.

Each operator in the experimental system is a LISP-function which manipulates the data structures representing the states. Since state parameters are specified as ranges of values, the operators are general enough to manipulate parameters that are specified as ranges. Furthermore, an operator can carry out the desired action at whatever level of abstraction is specified. In addition to the LISP-function that carries out the action of an operator on a state, each operator also consists of a symbolic description of the conditions under which it can be applied to a state. An example of an operator is given in Table 14.1.

We have also experimented with key portions of a qualitative reasoning system to prescribe parameter changes intended to overcome problems identified during diagnosis. This system utilizes qualitative models of how parameter changes affect the actions of the operators. We have implemented and experimented with such models for some DSN operators.

It is clear from the results of our investigations that system diagnosis and control of the DSN signal processing subsystem at each node is feasible with current AI technology. In particular, we implemented an experimental diagnosis system which identifies the processes and parameters responsible for DSN signal processing faults. We also designed and implemented portions of a control system for prescribing parameter changes intended to overcome the problems identified during diagnosis.

TABLE 14.1

Equal-Resolution Operator

Input Signal Type	Propagation, continuous-temporal, discrete-temporal, continuous-spatial
Output Signal Type	Continuous-spatial
Differences Reduced	Resolution
Operator Parameters	
Direction	Array-aperture
Power	Array-aperture
Frequency	Array-aperture, epsilon
Band	Array-aperture, epsilon
Gaussian	Array-aperture, epsilon
State Preconditions	Per pair of input signals
Direction	Direction difference intersects [0,100/array-aperture]
Power	Direction level preconditions Power in [0,inf]
Frequency	Minimum frequencies intersect Maximum frequencies intersect Direction difference intersects [0,(1000*epsilon)/(array-aperture*0.0001 *maximum-frequency)]
Band	Power level preconditions Frequency level preconditions Amps in [0,inf]
Gaussian	Frequency level preconditions with Gaussian model
Scenario Preconditions	None
State Postconditions	Per pair of input signals
Direction	Delete input signals Create signal whose direction is the cover of the two input directions
Power	Direction level postconditions Power of input signal in [0,sum of maximum powers in signals]
Frequency	Direction level postconditions Minimum frequency of output same as input Maximum frequency of output same as input
Band	Frequency level postconditions Power level postconditions Amps of output signal in [0,sum of maximum amps in signals]
Gaussian	Band level postconditions with Gaussian model

Although our investigations were limited to the signal processing subsystem, we believe that the diagnosis and cure approach that we have taken may be applicable to other functions of a DSN node. However, a substantial research effort might be needed to obtain positive proof for such a speculation. Furthermore, since many of the higher level functions of a DSN node depend upon interaction with other DSN nodes, the diagnosis and control problem should be viewed within a more general framework of distributed problem-solving. Our preliminary study of some specialized knowledge sources for a single DSN node will hopefully provide a small experiential base for future exploration of problems requiring the consideration of multinode issues.

15. REMAINING RESEARCH AND DEVELOPMENT

Although the DSN program has successfully achieved the goal of demonstrating the feasibility of DSN systems, it is clear there are many areas where additional work is needed. Following is a summary of such items in three categories: (1) distributed acoustic surveillance, (2) experimental system and subsystem prototypes, and (3) applications of artificial intelligence. Much of the work could be general, but some would benefit from emphasizing specific system applications.

There has been a lot of emphasis on acoustics during the DSN project. The feasibility of using acoustics for detection and tracking has been established but there are several remaining research questions. First, what are the limits of acoustic tracking for the case of multiple and maneuvering targets? Analysis, algorithm development, and experimentation are required to answer this question. Second, passive target identification can probably be accomplished acoustically. How well it can be done and how to do it should be investigated. Third, acoustic performance will vary considerably with environmental conditions. This needs to be understood, quantified, and analyzed in the context of specific system requirements.

There appear to be no fundamental impediments to the development of a DSN system. However, many engineering problems must be addressed in the context of a DSN system. A fully functional experimental prototype system should be developed, demonstrating all essential DSN features. This would include modular prototype nodes and a communication system with remote start, test, and operation features. Unlike the existing test-bed system, the experimental prototype would be functionally complete. For example, it would maintain time synchronization, provide multi-hop point-to-point communication, and maintain network configuration tables. Each function would be performed as it might be in a real DSN system. This differs from the test bed in which, for example, nodes are manually synchronized, communication services are specialized, network configuration is manually maintained in static tables, and no multisite data integration service is offered during real-time operation. A prototype system would establish that no unsolved system problems remain. This probably would require operating the experimental system for extended periods in a manner consistent with operational requirements.

Development and evaluation of a prototype system would would include assembling system development tools, demonstrating that they are sufficiently mature to support full-scale system development.

DSN systems offer many opportunities for the application of AI approaches to automated system operation and sensor data interpretation. Automated system test and control capabilities will be needed to effectively operate DSN systems. It may be feasible to exploit knowledge-based methods for the development of such diagnosis and control capabilities.

Finally, acoustic surveillance provides at least three interesting data-understanding problems. One is to develop acoustic tracking systems with improved performance by explicitly integrating knowledge into the system. This would have broad application beyond acoustic distributed systems. Target recognition is a second area where knowledge-based methods could be applied. The third area is knowledge-based adaptive signal processing.

ACKNOWLEDGEMENTS

The successful completion of the Lincoln Laboratory DSN project is the result of hard, dedicated, and competent work by many people.

John Delaney played a crucial role. Working with Bob Tenney he initiated the model-based approach to multisensor distributed tracking. He developed the algorithms, designed the application software, implemented much of it himself and oversaw the implementation of the rest. Sections 6 and 7 are based primarily on his work. He wrote the drafts for those sections as well as portions of Section 10.

Jim Needham was responsible for the test-bed hardware during most of its life and also helped to solve many software-hardware interface problems. His was a difficult and often thankless job that he carried out very well with help from many people including Larry Sargent, Joe Racamato, Bob D'Ambra, Bob Gallagher, Dick Simard, and Royce Brown. Hal Heggestad helped with the test-bed microwave radio system.

Joft Saks designed and implemented the NRTS software system. Connie McElwain made improvements to NRTS and developed radio communication software for the test bed.

Hamid Nawab did the initial work on the wideband direction finding algorithm and explored the area of knowledge-based system diagnosis. Others involved in the implementation and testing of the wideband algorithm include Tami Peli, Glenn Adams who also provided general system programming support to the project, and Bob Walton who also developed digital-data acquisition software and the initial delayed-time target location algorithms. Jorge Galdos developed and experimented with TV algorithms, building upon work by Nawab and aided by Royce Brown.

Lynn Feddem assisted with the implementation of tracking software and performed multisite data integration studies. She and Jim Needham were instrumental in test-bed experimentation and tuning that led to the final real-time tracking demonstrations.

Liz Hinzlemann developed distributed self-location algorithms.

Additional past contributers included Peter Green, Kieu Hua, Ted Retzlaff, Jeff Kurtz, Dave White, Dorothy Curtis, Mike Humphrey, Russ Dionne, Bob Ganley, Julie Rowhein, and Paul Demko.

REFERENCES

[1] D.E. Dudgeon, "Fundamentals of digital array processing," Proceedings of the IEEE, Vol. 65, No. 6, pp.898-904, June 1977.

[2] D.H.Johnson, "The application of spectral estimation methods to bearing estimation problems," Proceedings of the IEEE, Vol. 70, No. 9, pp. 1018-1028 Sept. 1982.

[3] S.W.Lang and J.H.McClellan, "Spectral estimation for sensor arrays," IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-31, No. 2, pp. 349-358, April 1983.

[4] J.S.Lim and N.A.Malik, "A new algorithm for two-dimensional maximum entropy power spectrum estimation," IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-29, No. 6, June 1981, pp. 401-413.

[5] M.Wax, T.Shan, and T.Kailath, "Location and the spectral density estimation of multiple sources," Proc. 16th Asilomar Conf. Cir. Syst. Comp., 1982.

[6] J.Capon, "High-resolution frequency-wavenumber spectrum analysis," Proc. IEEE, Vol. 57, No. 8, pp. 1408-1418, August 1969.

[7] S.H. Nawab, F.U. Dowla and R.T. Lacoss, "Direction determination of wideband signals," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. 33, pp. 1114-1122, October 1985.

[8] S.H. Nawab, F.U. Dowla, and R. T. Lacoss, "A new method for wideband sensor array processing," Int'l. Conf. on Acoustics, Speech and Signal Processing, San Diego, CA, March 19-21, 1984.

[9] A.V.Oppenheim and R.W.Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975, p. 358.

[10] J.H.McClellan, "Multidimensional spectral estimation," in Proceedings of the IEEE, Vol. 70, Sept. 1982, pp. 1029-1039.

[11] R.T.Lacoss, "Data adaptive spectral analysis methods," Geophysics, Vol. 36, Aug. 1971, pp. 661-675.

[12] R.R. Tenney and J.R. Delaney, "A Distributed Aeroacoustic Tracking Algorithm," American Control Conference, San Diego, CA, June 6-8, 1984.

[13] J.R. Delaney and R.R. Tenney, "Broadcast Communications Policies for Distributed Aeroacoustic Tracking," American Control Conference, Boston, MA, June 12-14, 1985.

[14] R.T. Lacoss, "Distributed Mixed Sensor Aircraft Tracking," American Control Conference, Minneapolis, MN, June 10-12, 1987.

[15] A.H. Jazwinski, *Stochastic Processes and Filtering Theory* (Academic Press, New York, 1970).

[16] Semiannual Technical Summary, Distributed Sensor Networks Program, M.I.T. Lincoln Laboratory, 30 September 1985

[17] A.S. Willsky, *et al.*, "Combining and Updating Local Estimates and Regional Maps Along Sets of One-Dimensional Tracks," IEEE Trans.on Automatic Control, Vol. AC-27, pp. 799-813, August, 1982.

[18] R.E. Kalman and R.S. Bucy, "New Results in Filtering and Prediction Theory," Trans. of the ASME: Journal of Basic Engineering, Vol. 83, pp. 95-108, March 1961.

[19] Y. Bar-Shalom, "Tracking Methods in a Multitarget Environment," IEEE Trans. on Automatic Control, Vol. AC-23, pp. 618-626, August 1978.

[20] Chee-Yee Chong, Kuo-Chu Chang, and Shozo Mori, "Distributed Tracking in Distributed Sensor Networks," Proceedings of 1986 American Control Conference, Seattle, WA, 1986.

[21] A.S. Willsky and H.L. Jones, "A Generalized Likelihood Ratio Approach to the Detection and Estimation of Jumps in Linear Systems," IEEE Transactions on Automatic Control, Vol. AC-21, pp. 108-112, February 1976.

[22] Semiannual Technical Summary, Distributed Sensor Networks, M.I.T. Lincoln Laboratory, 30 September 1982.

[23] Semiannual Technical Summary, Distributed Sensor Networks, M.I.T. Lincoln Laboratory, 31 March 1980.

[24] Semiannual Technical Summary, Distributed Sensor Networks, M.I.T. Lincoln Laboratory, 3O September 1979.

[25] J.H. Fischer, J.H. Cafarella, D.R. Arsenault, G.T.Flynn and C.A. Bauman, "Wide-Band Packet Radio Technology," Proc. IEEE, Vol. 75, No. 1, pp. 100-115, January 1987.

[26] Semiannual Technical Summary, Distributed Sensor Networks, M.I.T. Lincoln Laboratory, 31 March 1982

[27] Semiannual Technical Summary, Distributed Sensor Networks, M.I.T. Lincoln Laboratory, 31 March 1982

[28] R. Rashid, "Accent: A network operating system for SPICE/DSN." Technical Report, Computer Science Dept. Carnegie-Mellon Univ., May 1981.

[29] MACSYMA Reference Manual, The MathLab Group, Laboratory for Computer Science, MIT, Version Ten, First Printing, January 1983.

[30] Y. Yemini, D. Bacon, and A.Dupuy, "The Distributed Incremental Position Location System," Technical Report, Columbia University Department of Electrical Engineering and Computer Science, 1986.

[31] "Position Location and Navigation Symposium," IEEE, Atlantic City New Jersey, 1982.

[32] E. A. Hinzelman, "A Nonlinear Distributed Algorithm for Self-Location," MIT MSc Thesis, June 1985.

[33] J. Jubin and J.D. Tornow, "The DARPA Packet Radio Network Protocols," Proc. IEEE, Vol. 75, No. 1, pp. 21-32, January 1987.

[34] V.R. Lesser and D.D. Corkill, "Functionally Accurate, Cooperative Distributed Systems," IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-11, No.1, pp. 81-96, January 1981.

[35] E.E. Milios, "Signal Processing and Interpretation Using Multilevel Signal Abstractions," MIT PhD. Thesis, May 1986.

[36] E.E. Milios and S.H. Nawab, "Multilevel Signal Abstractions in Signal Processing," Proc. 1986 Digital Signal Processing Workshop, IEEE Acoustics, Speech and Signal Processing Society, 20-22 October 1986

[37] H.Nawab, V. Lesser, and E. Milios, "Diagnosis Using the Formal Theory of a Signal-Processing System," IEEE Trans on Systems, Man, and Cybernetics, Vol. 17, No. 3, pp. 369-79, May/June 1987.

[38] S.H. Nawab, V. Lesser, and E. Milios, "Conceptual Diagnosis of an Algorithmic Signal Processing System," Int'l. Conf. on Acoustics, Speech and Signal Processing, Tokyo, April 1986.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
ESD-TR-88-175

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Lincoln Laboratory, MIT

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Electronic Systems Division

6c. ADDRESS *(City, State, and Zip Code)*
P.O. Box 73
Lexington, MA 02173-0073

7b. ADDRESS *(City, State, and Zip Code)*
Hanscom AFB, MA 01731

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Defense Advanced Research Projects Agency

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F19628-85-C-0002

8c. ADDRESS *(City, State, and Zip Code)*
1400 Wilson Boulevard
Arlington, VA 22209

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
62708E			AO3345

11. TITLE *(Include Security Classification)*
Distributed Sensor Networks

12. PERSONAL AUTHOR(S)
Richard T. Lacoss

13a. TYPE OF REPORT
Final Report

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Year, Month, Day)*
1986, September, 30

15. PAGE COUNT
240

16. SUPPLEMENTARY NOTATION
None

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
multiple-sensor surveillance system
distributed tracking
target surveillance and tracking
communication network
acoustic sensors
video sensors
low-flying aircraft
acoustic array processing
digital radio
distributed estimation

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This is the final Lincoln Laboratory technical report for the DARPA Distributed Sensor Networks (DSN) project. The report reviews the DSN concept and documents the accomplishments, technical results, and lessons learned during the project. Topics covered include: acoustic direction finding, distributed tracking algorithms, hardware and software elements of an experimental test bed, network communication and self-location, multisite data-integration, and experimental demonstrations of tracking low-flying aircraft using multiple distributed acoustic and TV sensors.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
Lt. Col. Hugh L. Southall, USAF

22b. TELEPHONE *(Include Area Code)*
(617) 981-2330

22c. OFFICE SYMBOL
ESD/TML

83 APR edition may be used until exhausted.
All other editions are obsolete.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE