ESD-TR-79-330

LEVEL III

Semiannual Technical Summary

Distributed Sensor Networks

30 September 1979

Prepared for the Defense Advanced Research Projects Agency
under Electronic Systems Division Contract F19628-78-C-0002 by

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-78-C-0002 (ARPA Order 3345).

This report may be reproduced to satisfy needs of U.S. Government agencies.

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

FOR THE COMMANDER

Raymond L. Loiselle

Raymond L. Loiselle, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

DISTRIBUTED SENSOR NETWORKS

SEMIANNUAL TECHNICAL SUMMARY REPORT TO THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

1 APRIL – 30 SEPTEMBER 1979

ISSUED 5 MAY 1980

LEXINGTON MASSACHUSETTS

ABSTRACT

Research in the areas of tracking, signal processing, and system software is reported. A review of possible tracking techniques from the area of artificial intelligence was completed, and an approach selected for further development effort. Signal processing algorithms for measuring acoustic azimuths were further developed. Progress with software and hardware development of an acoustic data acquisition system, which will evolve into an experimental DSN node, is reported.

CONTENTS

CONTRIBUTORS
TO
DISTRIBUTED SENSOR NETWORK PROGRAM

GROUP 22

Lacoss, R. T.

Donko, P., Jr.
Duckworth, G.
Green, P. E.
Hornig, D.
Landers, T. E.
Retzlaff, A. T.
Walton, R. L.

DISTRIBUTED SENSOR NETWORKS

I. INTRODUCTION AND SUMMARY

This Semiannual Technical Summary (SATS) for the Distributed Sensor Networks (DSN) program reports research results for the period 1 April through 30 September 1979. The DSN program is aimed at developing and extending target surveillance and tracking technology in systems that employ multiple spatially distributed sensors and processing resources. Such a DSN would be made up of sensors, data bases, and processors distributed throughout an area and interconnected by an appropriate digital data communication system. It would serve users who are also distributed within the area and serviced by the same communication system. The case of particular interest is when individual sensors cannot view the entire surveillance area and when they can individually generate only limited information about targets in their field of view. The working hypothesis of the DSN program is that through suitable netting and distributed processing the information from many such sensors can be combined to yield effective and serviceable surveillance systems. Surveillance and tracking of low-flying aircraft, including cruise missiles, using sensors that individually have limited capabilities and limited fields of view, has been selected to develop and evaluate DSN concepts in the light of a specific system problem. The research plan is to investigate these concepts and to develop a test bed and demonstration system.

Progress and results obtained during this reporting period are summarized here and presented in greater detail in subsequent sections of this report. Major topics covered include tracking methodology, acoustic array processing, software design and development, and hardware development for an initial experimental DSN node.

The first experimental DSN will make use of multiple small acoustic arrays to detect and track low-flying aircraft. Each node will make acoustic azimuth measurements and extract other target signatures. The tracking problem is how to use such multiple-site information for locating and tracking low-flying aircraft. The problem has been reviewed, and it has been noted that it is characterized by there being many possible interpretations of observational data and that the believability of various interpretations can vary a great deal. A review of techniques used in the area of artificial intelligence was completed, and a methodology, the Hearsay Methodology, was identified which might be adapted to the DSN tracking problem. List-processing software, important for the Hearsay approach, has been developed, and exploratory programming experiments conducted to verify that this software, in conjunction with the C programming language, will be a useful tool for further work in this area.

Frequency domain beamforming techniques continue to be the array processing methods we plan to use in all initial experiments. We have further decided that, because of its better resolution and suppression of false targets, the Maximum Likelihood Method will be the specific method to use. A review of that technique is included in this report along with progress in the selection of the proper algorithm parameter settings. The main parameters are the time interval to be used for analysis, the frequency resolution, and the stability of spectral estimates. In support of this, we developed software for simulating acoustic signals at arrays. The simulation is based upon autoregressive modeling of the signals.

Considerable progress was made in the area of software development during this reporting period. The design and coding of a real-time kernel for data acquisition was completed, and testing and debugging is about to start. This software will support acoustic data acquisition in the near future and will be extended to support real-time multiple-node tracking in the coming year. Actual data acquisition software has been designed and coding will begin shortly. Currently the PDP-11/34, which is the basis of the data acquisition node, is connected to our PDP-11/70 software development machine via a 9600-baud teletype connection. A simple downloading, dumping, and debug software package has been developed for that configuration and will also be used when the two machines are separated and interconnected by telephone lines.

The PDP-11/34 and all other hardware for the data acquisition node have been acquired and integrated in the laboratory. Testing and checkout is continuing.

II. TARGET TRACKING

The experimental DSN system that we are planning to deploy will use acoustic sensors to detect and track low-flying aircraft. Each node will use an array of microphones to determine the direction of arrival of the sound waves at that node. Azimuths measured by the different nodes will then be combined to determine the locations and tracks of the aircraft. The following sections describe some of the problems encountered in performing acoustic location and tracking of aircraft, and discuss the selection of an approach for solving these problems. In addition, it describes the selection of a programming language, development of a list-processing subroutine package, and the current state of our tracking algorithm development. The emphasis here is upon the possible application of Artificial Intelligence approaches to the acquisition and tracking problems. Similar problems also exist and solution techniques have been developed for acquisition and tracking in the more classical radar system context. We plan to incorporate classical approaches as appropriate but the emphasis presented here is upon a viewpoint which is not the classical one. In some of the following, the ideas discussed are very similar to classical approaches but the viewpoint is substantially different.

A. SUMMARY OF THE PROBLEM

From our signal processing algorithms we can obtain power as a function of azimuth of the sound waves arriving at a microphone array for a number of elevations and frequencies. These data can be available on the order of once a second with a delay from real time which is also on the order of once a second. The problem is to determine target locations and tracks from the observations taken at multiple sites.

We can plot power as a function of azimuth and normalized wavenumber as shown in Fig. II-1, where the normalized wavenumber is the wavenumber (radians per meter) parallel to the surface of the array, of a wave arriving from some elevation divided by the wavenumber of a horizontal wave at the same frequency with some nominal sound velocity. The radius from the center of the graph represents the normalized wavenumber, and the azimuth is represented by the angle from the axis. A value of unity for k indicates that the sound wave is coming in horizontally, i.e., that the target is on the horizon. A value of k of zero implies that the target is overhead.

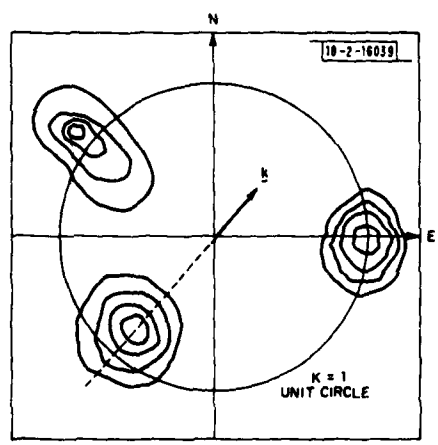

Fig. II-1. Iso power contours on a power vs azimuth and normalized wavenumber map.

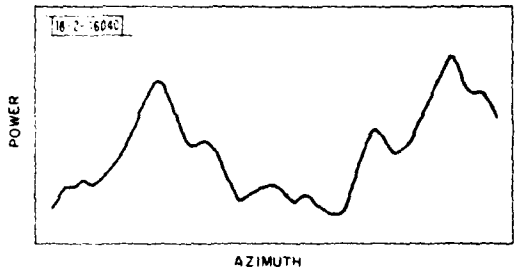

Fig. II-2. Power vs azimuth around the unit circle.

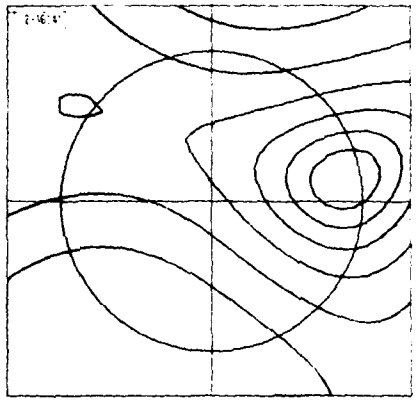

Fig. II-3. Iso power contours recorded during an A-7 jet flyby at Fort Huachuca, frequency = 50 Hz.

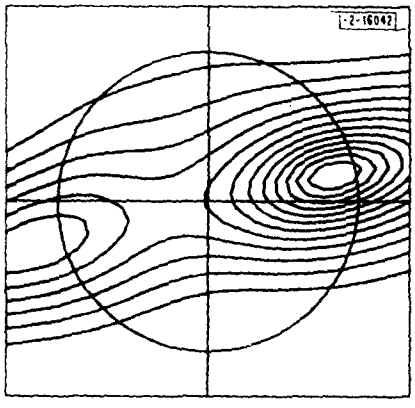

Fig. II-4. Iso power contours recorded during an A-7 jet flyby at Fort Huachuca, frequency = 60 Hz.

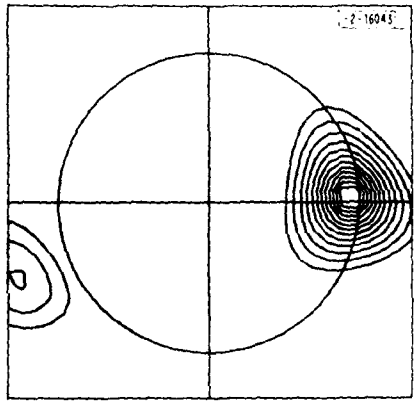

Fig. II-5. Iso power contours recorded during an A-7 jet flyby at Fort Huachuca, frequency = 75 Hz.

Values of k = 1 only have meaning for sound velocities differing from the nominal. More discussion of signal processing involved with the generation of such power maps is contained in Section III of this SATS.

Figure II-1 is a contour plot and has a number of peaks. Each of these peaks corresponds to a possible target. By determining the azimuths of these peaks, we can determine the possible angles of arrival of the sound waves. It should be noted that not all the peaks lie on the unit circle. Some are inside, indicating target elevation above the horizon. Some are outside, indicating that our assumed value for sound velocity was in error.

We expect low-flying aircraft to first appear close to the horizon. By looking at plots of power around the unit circle, as shown in Fig. II-2, we expect to find a significant maxima at an azimuth close to that of the true peak. Selecting each of these maxima, we can then find the peak to which it belongs by searching for the closest maxima in the full power map.

The different frequencies contain separate and potentially useful information. For example, Figs. II-3 to -6 show that power maps for an A-7 jet recorded at Fort Huachuca. (See Refs. 1 and 2 for details of this experiment.) In these figures, other interfering sources of sound are clearly delineated at some frequencies and not at others. One of the problems that requires further research is how to select the frequencies at which to analyze the data.

When we have determined the azimuths of wave arrival at multiple sites, the problem remains of how to correlate these to find target locations. The first step is to form azimuth tracks vs time for each peak, at each site, as shown in Fig. II-7. We can translate these azimuth tracks to a curve of possible loci for the target at some time in the past T, as shown in Fig. II-8. We make use of the fact that the sound emanating at T will travel C meters in one second, where C is the velocity of sound (approximately 330 m/sec). If the target was C meters away at T, then its sound would arrive at T + 1 for which we have recorded the azimuth Θ_1. Thus, we have a possible location for the target at time T, which is C meters away at an azimuth Θ_1. Alternately, the target could have been 2C meters away when the sound would have arrived at T + 2, for which we have recorded Θ_2. By applying this technique for all possible times between T and the present, we are able to generate the locus of possible positions of the target at time T as shown in Fig. II-8.

If we form these loci at two nodes for the time T, we can determine the location of a single target by determining the intersection of the curves (Fig. II-9). Repeating this procedure for a sequence of times, we obtain a sequence of target locations which can be associated together to form a track. This method was presented in detail in our 31 March 1978 SATS.

For a single well-defined target, this location and tracking procedure would appear to have no problems. However, if we have multiple peaks then several problems arise. First there is the problem of associating peaks with azimuth tracks. With many peaks, a particular peak may seem to belong to more than one track, and also the tracks may cross over as shown in Fig. II-10(a). This results in ambiguities in possible loci and hence in the resultant positions and tracks. Another problem occurs when we have well-defined azimuth tracks but the existence of multiple-target loci at two sites results in ghost targets as shown in Fig. II-10(b). In this figure, we show two nodes, along with their possible location curves for each of three targets. It is apparent that there are nine possible target locations resulting from the intersects. Three of these are real, and six are ghosts.

In resolving these ambiguities, we need to make use of knowledge, such as prior observations from other sites, knowledge of the frequency spectra observed at each site, and knowledge

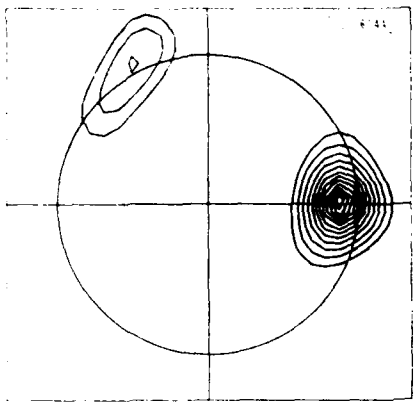

Fig. II-6. Iso power contours recorded during an A-7 jet flyby at Fort Huachuca, frequency = 100 Hz.

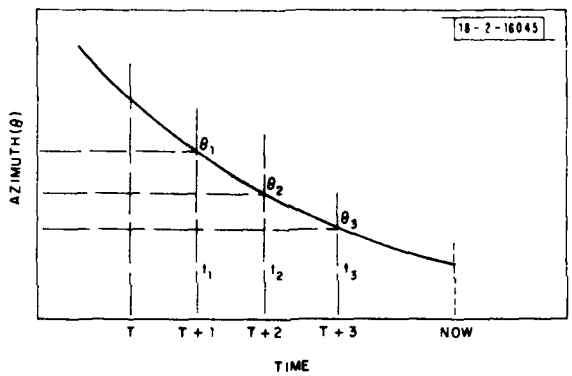

Fig. II-7. Azimuth vs time for a single target.

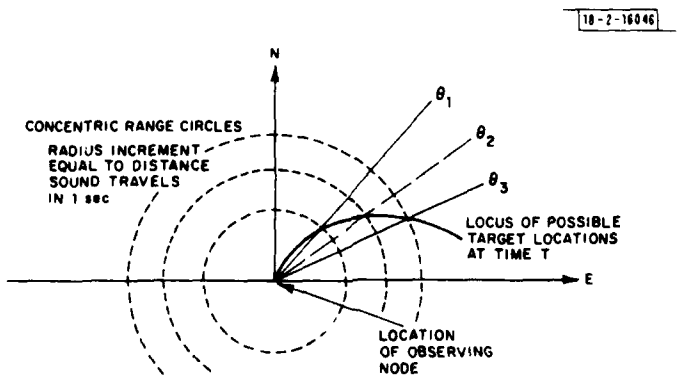

Fig. II-8. Possible location curve derived from azimuth vs time curve.

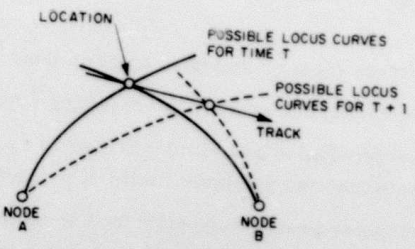

Fig. II-9. Conversion of possible loci into locations and tracks.

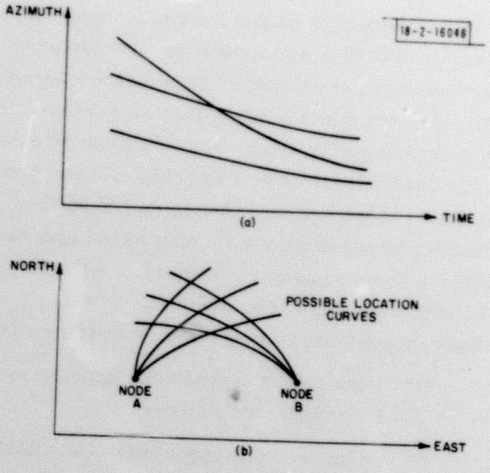

Fig. II-10. Effect of multiple targets on azimuth/time curves and location curves. (a) Multiple azimuth vs time for a single site. (b) Multiple possible position curves leading to ambiguity.

of possible target motion. One of the major challenges of tracking-algorithm develcpment is to achieve good detection and tracking of targets in the presence of ambiguities and multiple interfering sources of sound.

B. METHODOLOGY

In selecting a methodology with which to approach the tracking problem, we are confronted with one major characteristic of this problem. That characteristic is that there is no absolute right or wrong answer, but simply that one result may seem to indicate the presence of a target more strongly than another. In selecting possible alternatives, we are able to make use of many clues, but none of these is absolute in differentiating right from wrong, especially in the presence of systematic as well as random errors.

In many cases, we can determine statistical measures on the relationships between peaks, locations, and tracks. However, we are left with a considerable amount of heuristic decision making. While not strictly necessary, it is highly desirable to have a formal methodological structure into which to embed such decision making. Such methodologies exist in the field of Artificial Intelligence (AI).

A literature review of various AI methodologies has been completed with the intention of selecting one for further development and evaluation. The requirements for the methodology were that it should be able to:

(1) Develop competing hypotheses from an ambiguous set of data.

(2) Have a framework for combining and evaluating competing hypotheses.

(3) Be selective in hypothesis generation and elimination so as to keep the data space for hypotheses within the available limits of physical memory.

(4) Be selective as to the computer processing performed so as to keep the processing power required within the bound of that available in real time.

(5) Be suitable for distribution over multiple processors with limited communications bandwidth between them.

The methodology chosen for preliminary implementation and further evaluation was that embodied in the Hearsay speech recognition program (Refs. 3-13) and which has become known as the Hearsay methodology. This methodology was the only one that met all of the above requirements and, while needing further research into its use for real time and continuous hypothesis space applications, appears to be an excellent starting point for DSN.

The basic elements of a Hearsay structure are Knowledge Sources (KSs) that generate, strengthen, or weaken hypotheses; a Blackboard on which to record the state of the hypotheses; a Scheduler to schedule the running of the Knowledge Sources; and a Pruner to eliminate weak hypotheses so as to keep the number of hypotheses being processed to within the number that can fit within the available memory.

Some possible Knowledge Sources within a DSN context are:

(1) Peak picker – generates hypotheses as to azimuths of arrival of sound from different sources.

(2) Azimuth track associator – associates peaks with existing azimuth tracks.

(3) Azimuth track initiator – initiates new azimuth track hypotheses.

(4) Locator – hypothesizes possible location from direct and communicated azimuth tracks.

(5) Spatial tracker – associates locations with existing spatial tracks and modifies and strengthens track hypotheses based on these updates.

(6) Spatial track initiator – initiates new spatial tracks based on residual locations not used by tracker.

(7) Spectral correlator – weakens or strengthens location and track hypotheses based on spectral content correlations.

(8) Motion correlator – weakens or strengthens hypotheses based upon target motion being close to that expected for targets of interest.

(9) Peak to Spatial Track correlator – strengthens or weakens spatial tracks based on recently detected peaks for which no corresponding information has yet been received from other nodes.

The Blackboard is the data structure on which each of the knowledge sources operate. In DSN, there will be a Blackboard in each node and this will contain that node's hypotheses as to the state of the environment. These hypotheses will be different for each node. Each Knowledge Source is a task that runs in a computer and communicates only with the Blackboard. Hence new KS tasks can be added, or old ones modified, without affecting any other KSs. This is very important in DSN where much work needs to be done in the development of the KSs, and where different KSs will be developed by different people.

Some Knowledge Sources will take local data and generate local hypotheses or modify them. Other KSs will take incoming communicated data and use them to modify the Blackboard state, by creating new hypotheses or adding to or weakening existing hypotheses. By this mechanism, both local and communicated data can be used in formulating hypotheses. However, by using separate KSs for communicated data, we can introduce the element of believability based on the source of the data. This prevents one node from overwhelming another, even in the presence of a malfunction.

The Pruner is a special KS whose function is to eliminate weak hypotheses. This regulates the size of the Blackboard to be within the amount of available memory. There are two ways of eliminating hypotheses, first by simply deleting them and second by combining them with other hypotheses thereby strengthening the remaining hypotheses.

The function of the Scheduler is to select which KS to run. There is only a limited amount of computing power available, and it is important that the Scheduler run those KSs that will produce optimum results. By its choice of KSs to run, the Scheduler can force attention to be devoted to tracking existing targets or to the generation of new target hypotheses. In addition, we may need some focusing mechanism that will cause the Knowledge Sources to focus on some spatial sector or some designated target. Also different Knowledge Sources can have different abilities to resolve hypothesis conflicts and the Scheduler will ultimately have to take this into account.

In addition to the hypotheses, KSs, Scheduler, and Pruner, we need yet another function for DSN, that of the Communicator. The Communicator is a specialized KS that looks at the current state of the Blackboard and decides what data it should communicate. The available communications bandwidth is limited so that the Communicator must be selective as to what is communicated. It must work at all levels of hypothesis, not just at the top level which is normally of interest to system users.

The hypotheses themselves are built upon hypotheses or on measured data as shown in Fig. II-11. Each hypothesis has a certain strength. This is determined by the strength of its

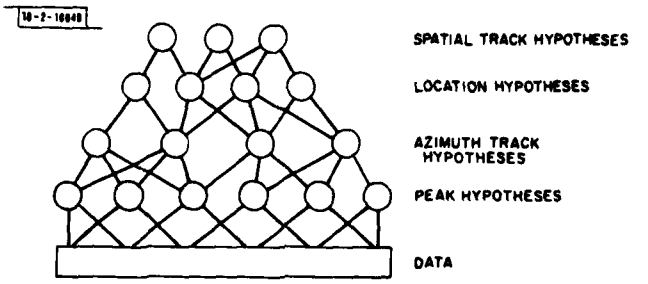

Fig. II-11. Hypothesis structure.

supporting hypotheses and the strength of the supporting link. These strengths may be determined statistically or purely heuristically dependent on the KS doing the evaluation. It is important that the strengths are on a comparable basis, such that the best of the competing hypotheses can be selected.

In selecting the final results to present to a user, we cannot say that one spatial track exists and another does not. We can only state that one is more likely to exist than another, and provide a measure of how much more likely. This measure is a combination of all the clues incorporated into the result, through the hypothesis structure.

In applying this type of methodology to DSN, one has to be aware of the real-time nature of the problem. New data are continuously being generated. These data are more valuable in that they are more up to date than old data. Hence, we must continuously degrade the strength of our hypotheses as their supporting data grow old. In this way, hypotheses that are not continuously being reinforced with new data will die out and be eliminated by the Pruner.

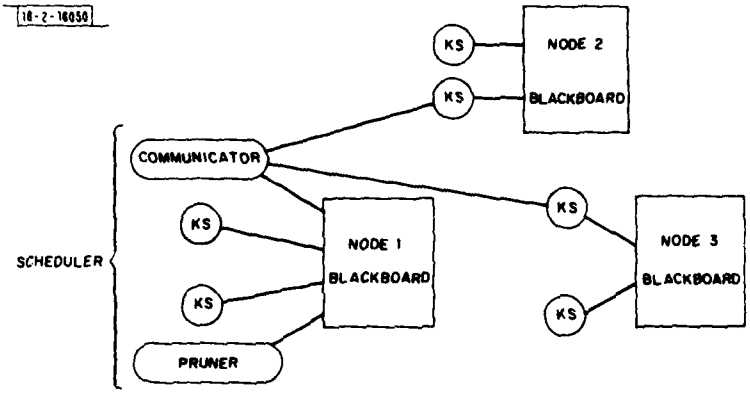

Fig. II-12. Top-level node software structure.

The resultant structure for tracking software is shown in Fig. II-12 for multiple nodes. Each node has a common data structure (the Blackboard), and multiple tasks (KSs) to be run under the control of a Scheduler. The tasks are independent and only communicate internally through the common data structure. External communications is handled through a single task (the Communicator) which talks to tasks (KSs) in other nodes. A detailed examination of this structure reveals that it is in line with modern software engineering concepts. In reality, most of the Hearsay methodology is based on common sense reasoning. However, by placing it into a formal framework we can consider the constituent parts without having to consider the whole as one huge problem in heuristic programming.

C. PROGRAMMING LANGUAGE SELECTION

When developing software, the selection of a programming language is very important. In a real-time environment with some highly mathematical determinators of hypothesis support levels, the appropriate selection is extremely important. With the Hearsay structure, the Blackboard consists of lists of hypotheses that have lists of attributes and lists of supports. These may themselves be elements of lists. Hence, it is highly desirable to use a language that has good list processing capabilities.

The traditional AI language, which very effectively supports list processing, is LISP. To evaluate its suitability in the DSN application, we installed ULISP (Ref. 14) on our PDP-11/70 computer and then wrote some simple Knowledge Sources in it. It was easy to implement the list-processing parts of the KSs in LISP. However, the mathematical algorithms required a lot of work to achieve what could have been achieved far more simply in a scientific language.

The interpretive implementation of LISP made programs very easy to debug. However, the LISP programs ran very slowly and also were very core intensive. We rapidly ran out of space on our 11/70 when trying to do anything significant, even though all 64K bytes of data space was available to the LISP interpreter. Hence, we decided to seek an alternative.

We needed a language that had good mathematical, logical, and list-processing attributes, and that compiled and ran efficiently on a PDP-11. This latter requirement stems from the fact that the experimental nodes will have PDP-11/34 computers in them. A study of languages such as Sail (Ref. 15) and Mainsail (Ref. 16) revealed that there were no suitable versions for PDP-11 computers. The "C" language that we have been using for our processing under Unix meets all the requirements except for the list processing. Hence, we decided to use C and extend it by writing a list-processing subroutine package that we have called CLMS for C List Management System.

An initial version of CLMS has been written, debugged, and used to write Knowledge Sources. It appears to meet the requirements in that it is only slightly harder to write the list-processing parts in CLMS than in LISP, whereas the math and logic is much easier to program. The implementation is efficient in its use of space and appears to run fast.

In designing CLMS two major choices were made. The first was to make the basic data element of CLMS a list of C structures, which has proven extremely powerful. The second decision was not to use an implicit garbage collector. Like most list-processing packages, CLMS allocates and returns space to and from a free storage area. In CLMS, the programmer has to specifically call for the removal of each list or element of a list. He cannot imply removal by reassignment, nor can he specify a list of lists for removal. Removal occurs when requested. This is different from a garbage collector which will run when new space is needed and all the free space has been used up. We made this decision based on the need for speed and control in a real-time environment. An implicit garbage collector is slower (has to do more work to be smarter) and also can choose to run at an inappropriate time in a real-time environment.

It is expected that we will add additional routines and functions to CLMS as the DSN project progresses. It is currently planned to use CLMS for all the tracking algorithm development. Because CLMS is written in C, it will be transported along with the C language onto different processors that we may incorporate into DSN nodes.

D. DISCUSSION

We are now pursuing a detailed investigation of Knowledge Sources, the Blackboard structure, and the Scheduler for a real DSN system. These elements will be tested against live data acquired using the data acquisition system, as well as simulated data for specific environments of interest. The following paragraphs discuss some of the ideas to be pursued.

In target acquisition and tracking problems, there are many more possible hypotheses than can ever be evaluated in detail. Consider, for example, the very simple case of position readings from two aircraft flying parallel. Figure II-13 shows their positions, labeled A through H,

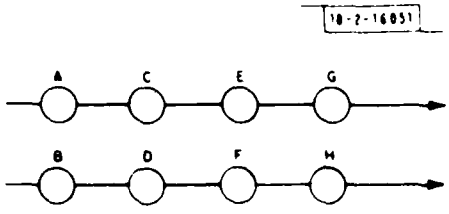

Fig. II-13. Location hypotheses for two parallel aircraft tracks.

at a sequence of four times. In forming tracks, we could hypothesize a track ACEG or ADEH and so forth. Simply trying to form every possible hypothesis and eliminating them later is not a good idea. Instead, the alternative is to fit the new data to existing tracks, and to modify the tracks to fit the data. This eliminates the problem of hypothesis explosion, but leaves unanswered the problem of acquisition of new targets. The most promising solution seems to be the following. First analyze the data in terms of existing hypotheses and strengthen the hypotheses where appropriate. Then evaluate residual data to see if there are any potential new tracks and initiate new tracks where appropriate. The notion is to treat targets as one if you cannot differentiate between them clearly. This is a common approach to acquisition and tracking and is the initial approach to be investigated for the DSN.

There will be a number of spatially stationary sources of noise. It should be possible, once having determined that they are stationary, to eliminate these directly from the lowest level of hypothesis formulation. That is, a highly confirmed hypothesis can be used to eliminate data on input. It may also be possible to use measurements of the spectrum of the stationary source to choose frequencies with low noise for the purpose of searching for other targets.

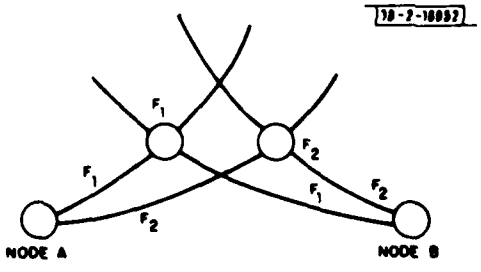

Fig. II-14. Location ambiguity resolution based on spectral discrimination.

Two major questions to be investigated are how to use spectra and motion as target discriminators. It is apparent that if we have two azimuth tracks observed from two sites we can potentially use spectral discrimination as shown in Fig. II-14. Here, if one target has a strong spectral response at F1 and the other at F2, then we can determine the true locations from the ghosts. This is complicated by the fact that the spectrum emitted from an aircraft varies with attitude and that the observed spectra vary with speed and distance. Nevertheless, we can probably strengthen or weaken hypotheses based on spectral discrimination. The question of target discrimination based on the reasonableness of observed target motions is currently being investigated.

REFERENCES

1. Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (30 September 1977), DDC AD-A050160/1.
2. Ibid. (30 September 1978), DDC AD-A070455.
3. V. R. Lesser and L. D. Erman, "An Experiment in Distributed Interpretation," University of Southern California Information Sciences Institute Report ISI/RR-79-76 (May 1979).
4. L. D. Erman, "An Environment and System for Machine Understanding of Connected Speech," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, May 1975.
5. L. D. Erman, "Overview of the Hearsay Speech Understanding Research," Computer Science Research Review 1974-1975, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.
6. L. D. Erman and V. R. Lesser, "A Multi-level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge," Proc. IJCAI., Vol. 2, 483 (1975).
7. V. R. Lesser, R. D. Fennel, L. D. Erman, and D. R. Reddy, "Organization of the Hearsay II Speech Understanding System," IEEE Trans. Acoust., Speech, and Signal Processing ASSP 23, 11 (1975).
8. R. D. Fennel and V. R. Lesser, "Parallelism in AI Problem Solving: A Case Study of Hearsay II," Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania (1975).
9. F. Hayes-Roth, G. Gill, L. D. Erman, V. R. Lesser, and D. J. Mostow, "Hypothesis Validity Ratings in the Hearsay II Speech Understanding System," Working Papers in Speech Recognition IV, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania (1976).
10. F. Hayes-Roth, G. Gill, L. D. Erman, V. R. Lesser, and D. J. Mostow, "Discourse Analysis and Task Performance in the Hearsay II Speech Understanding System," Working Papers in Speech Recognition IV, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania (1976).
11. A. R. Smith, "Word Hypothesization in Hearsay II Speech System," in Proc. IEEE Int'l. Conf. on ASSP, Philadelphia, Pennsylvania, 12-14 April 1976, p. 549.
12. R. Cronk and L. D. Erman, "Word Verification in the Hearsay II Speech Understanding System," Working Papers in Speech Recognition IV, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania (1976).
13. L. Shokey and C. Adam, "The Phonetic Component of the Hearsay II Speech Understanding System," Working Papers in Speech Recognition IV, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania (1976).
14. R. L. Kirby, "ULISP for PDP-11s with Memory Management," Report MCS-76-23763, University of Maryland Computer Science Center (June 1977) (revised September 1978).
15. J. A. Feldman and P. D. Provner, "An Algol Based Associative Language," Commun. ACM 12, 439 (1969).
16. C. R. Wilcox, M. L. Dageforde, and G. A. Jirak, "Mainsail Language Manual," Stanford University, California (July 1979).

III. SIGNAL PROCESSING

A. ACOUSTIC ARRAY PROCESSING

The experimental DSN system that we are planning to deploy will use acoustic sensors to detect and track low-flying aircraft. Each node will use an array of microphones to determine the direction of arrival of the sound waves at that node. The azimuths measured by different nodes will then be combined to determine the locations of the aircraft.

This section reviews the algorithms that will be used to process the microphone signals to estimate the directions of arrival of sound waves. Some of this material has appeared in other forms in previous SATS and references (1, 2, 3), but interactions with other DSN contractors and internal discussions have led to the view that it would be of value to the overall program to summarize our basic approach for acoustic array processing.

1. General Problem Description

Sound waves emanate from many sources in our environment. Some of these, such as aircraft, trucks, and air conditioners act as point sources. Others, such as groups of trees being rustled by the wind, appear as distributed sources of random noise. In general, we may consider the DSN environment as a number of point sources of sound amidst a background of distributed random noise.

All sources of sound emit waves which, in a homogeneous atmosphere, spread spherically and are therefore attenuated as the inverse of the square of the distance from the source, in addition to any atmospheric absorption. These waves are pressure waves, which are converted by the type of microphone we are using in DSN to an electrical signal whose voltage is proportional to the pressure.

The output from a microphone is proportional to the sum of the pressure waves impinging on it. With a single microphone, all we can determine is the loudness of the sound at that point. However, with an array of microphones, such as that shown in Fig. III-1, we can potentially determine the direction of arrival of the waves, as a wave appearing at one microphone will appear at another at a slightly different time. The problem we are addressing is how to determine the angles of arrival of multiple sound waves at a microphone array, in a DSN environment.

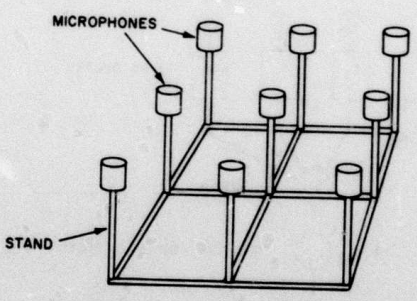

Fig. III-1. 3 × 3 microphone array.

2. Methods of Angle of Arrival Determination

To determine the angle of arrival of a wave, we use the fact that a wave from a single point source will produce coherent signals at different microphones. Consider a wave impinging on two microphones as shown in Fig. III-2. The wave will arrive at microphone 2, at time

$$td_{12} = \frac{d_{12} \cos \phi}{C}$$

later than it arrived at microphone 1, where d_{12} is the distance between microphones 1 and 2, C is the speed of sound in air (approximately 330 m/sec), and ϕ is the azimuth. By determining the time delay, it is possible to determine the direction of arrival of the sound.

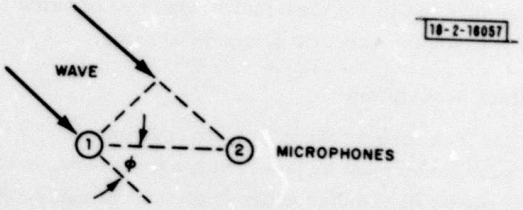

Fig. III-2. Wave impinging on two microphones.

If at any time, there is only one impulsive source, we can determine the difference in arrival times of the pulses at the microphones directly. However, in DSN, this situation does not prevail. Hence, we must use more indirect techniques. These can be divided into two basic methods: time-domain beamforming and frequency-domain beamforming.

In simple time-domain beamforming, we delay the signals received at each microphone, sum them, and then determine the average output power. Consider a linear time-domain beamforming array with a single wave incident at angle ϕ as shown in Fig. III-3. This array has time delays T_1 through T_4 applied to the outputs of the microphones, which are then summed and the average power output detected.

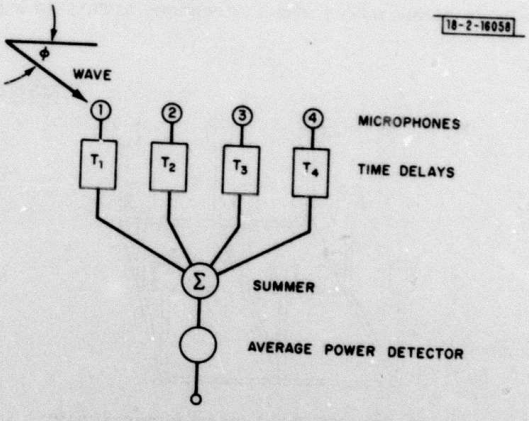

Fig. III-3. Time-domain beamforming.

If we make

$$T_1 = td_{12} + td_{23} + td_{34} = \frac{d_{12}\cos\phi + d_{23}\cos\phi + d_{34}\cos\phi}{C}$$

$$T_2 = td_{23} + td_{34} = \frac{d_{23}\cos\phi + d_{34}\cos\phi}{C}$$

$$T_3 = td_{34} = \frac{d_{34}\cos\phi}{C}$$

$$T_4 = 0$$

where the td_{jk} are selected based on the assumed angle ϕ, then all the components from the wave will add in phase and the output will be maximized. At angles other than that for which the delays are set, components of the wave will be out of phase and the average output power will be lessened. For a single incident wave, we can determine the angle of arrival by adjusting the values of T_i to look at each angle of possible arrival and then select the angle with maximum power amplitude. When there is more than one incident wave, there is the possibility of confusion as the response due to wave 1 when looking in the direction of wave 2 will, in general, be non-zero.

The output contains components for all the frequencies in the incident waves. To assist in discriminating between the incident waves, it is advantageous to separate out the frequencies in the waves. We could do this by Fourier analyzing the time series out of the summer for each selected azimuth before calculating the power components.

The analysis procedure can be considered in a different way in which time delays are replaced by phase shifts. This modified procedure is referred to as frequency-domain beamforming analysis. For frequency-domain beamforming, we would prefilter the acoustic signals to a narrow frequency band of interest, sum the components at a particular frequency, after phase shifting an appropriate amount, and determine the average power, as shown in Fig. III-4.

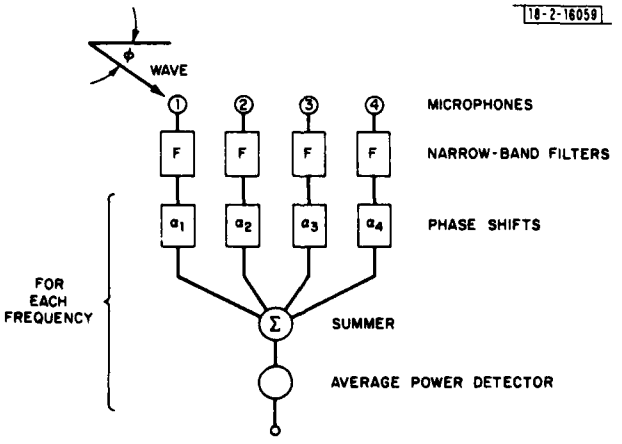

Fig. III-4. Frequency domain beamforming.

If we make

$$\alpha_1 = \frac{2\pi}{\lambda} \cos\phi(d_{12} + d_{23} + d_{34}) = 2\pi f T_1$$

$$\alpha_2 = \frac{2\pi}{\lambda} \cos\phi(d_{23} + d_{34}) = 2\pi f T_2$$

$$\alpha_3 = \frac{2\pi}{\lambda} \cos\phi(d_{34}) = 2\pi f T_3$$

$$\alpha_4 = 0 = 2\pi f T_4$$

where $\lambda = C/f$, C is the velocity of sound, and f is the frequency, then the signals will add in phase.

For waves incident at angles other than the look angle for which the alphas have been set, the power output is less, due to the signals being out of phase. Hence, as in the time-domain case, we could find the angle of incidence by evaluating the power at different look angles to find the maximum.

Simple frequency-domain beamforming works for a single incident wave. However, when there are multiple waves impinging on the array there is the possibility of confusion. The array response, when the phase shifts are set for a particular look angle, vs true angle of arrival, is a function such as is shown in Fig. III-5. The response as a function of azimuth is called the beam pattern. The look angle can be moved in azimuth by changing the phase shifts.

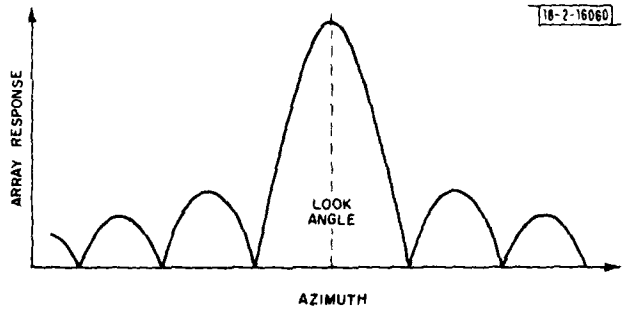

Fig. III-5. Beam pattern for array with simple beamforming.

The first sidelobes are normally not very far down from the peak response. Hence, a strong signal in the direction of a sidelobe would give an apparent output at the look angle, even if nothing were there. In a DSN, we are trying to acquire aircraft when they are far away in the presence of closer and therefore louder sounds. In this environment, the confusion caused by the sidelobe effects can be great. Hence, it is necessary to use more sophisticated beamforming techniques.

The more sophisticated technique selected for the initial DSN experiments is the Maximum Likelihood Method. This method is a frequency-domain beamforming technique wherein phase and amplitude weights are applied to the frequency components of the microphone signals in such a way as to steer the nulls of the beam pattern into the directions of the interfering sound sources as shown in Fig. III-6.

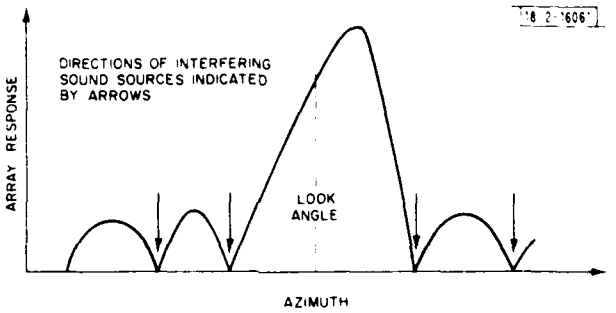

Fig. III-6. Beam pattern for array with adaptive beamforming.

This is done for each look angle and frequency and results in a computationally feasible technique that has good discrimination between adjacent sources of sound.

3. Frequency Domain Beamforming and Maximum Likelihood Method Algorithms

The preceding discussion gave a physical interpretation to the ideas of directional measurements using time-domain beamforming and frequency-domain beamforming, including the Maximum Likelihood Method of frequency-domain beamforming analysis. Here we quickly review the basic ideas of the actual algorithms which are being used.

Consider an array of N microphones located in the horizontal plane. For that array of microphones and a frequency f, we can define a steering vector $\bar{E}$ with elements

$$E_j = e^{(-i2\pi/\lambda)\,\bar{\nu}\cdot\bar{z}_j}$$

where $\bar{\nu}$ is a two-dimensional vector in the horizontal plane and $\bar{z}_j$ is the two-dimensional vector location of the jth microphone in the horizontal plane relative to the center of the array. The vector $\bar{\nu}$ is the projection into the horizontal plane of a three-dimensional unit vector pointing in the direction from the source to the center of the array. This is all a simple generalization of the linear-array phase shifts for planar arrays in three-dimensional space. We refer to $\bar{\nu}$, which has a maximum length of unity, as a normalized wavenumber. We also define the spectral covariance matrix K. This matrix is a function of frequency, and the i, j element at frequency f is the cross-power spectral density between channels i and j. As a frequency function, the i, j element is the Fourier transform of the cross-correlation between channels i and j.

Consider the procedure indicated in Fig. III-4. If the averaging time of the power detector goes to infinity and the bandwidths of the filters go to zero, the output of the detector goes to

$$P_{ave} = \bar{E}^H K \bar{E}.$$

If the value of K were known and the signals of interest did not change their statistical properties as a function of time, then the above equation could be used to scan frequency and direction space to determine power as a function of those two parameters. However, K is not known and the statistics of the sound field change in time. The solution is to consider a small time interval of observations which can be used to estimate K and also to make the interval small enough so that the processes can be considered stationary over the time interval. The above equation is then used to search for power sources but now using an estimate $\hat{K}$ of K rather than the true value.

The estimate $\hat{K}$ which we use is obtained by the block averaging method. The data during the time period T, during which the processes are believed to be stationary, are broken into M blocks of data. The Fast Fourier Transform (FFT) of each block is calculated. For each block, and for frequencies of interest, the products of the transform of the ith and jth channel are formed. These products are then averaged over the M blocks. This will converge to K if T and M both go to infinity in an appropriate way. For finite T, the estimate of K obtained in this way is on the average a smear of the true values over a small band of frequencies. The width of the band of frequencies is on the order of M/T Hz, and the smearing function is called a window function. The value of M determines the statistical stability of the estimate. It should be noted that K is an $N \times N$ matrix, and the estimate described here will be singular if M is less than N.

The above frequency-domain beamforming algorithm is deficient in that it does not adapt to changes in the noise field or to the presence of multiple targets. In the terms of Fig. III-4, we want to introduce phase shifts and gains which will give unity output in the direction of interest and which are selected to minimize power contributions from sources located in other directions. To do this we multiply each element of $\overline{E}$ by the corresponding element of a complex filter vector $\overline{W}$. The elements of $\overline{W}$ are constrained to sum to unity to assure that the signal in the steer direction is processed with unity gain. Subject to that constraint, the elements of $\overline{W}$ are selected to minimize the power output coming from noise and from sources which are not in the look direction. This filter is formed for each look angle and frequency and results in the adaptive beamforming formula:

$$P_{ave} = \frac{1}{\overline{E}^H K^{-1} \overline{E}}$$

where $\overline{E}$ is our original beamsteering vector. This result, which does not contain $\overline{W}$ and is not obvious, is derived in Reference 1 and, for a slightly different situation, in Reference 2.

As with the P_{ave} formula for ordinary frequency-domain beamforming, the problem is that the noise and signals are not time stationary and the true value of K is not known. Once more the solution is to use the above expression for P_{ave}, but to use the estimate $\hat{K}$, obtained over a limited time interval, in place of K.

B. DETAILED DESIGN OF PROCESSING ALGORITHMS

As discussed in Section III-A above, it is necessary to form estimates of the spectral covariance matrix K as part of the array processing task. In forming the estimate we must address the following questions:

(1) Over what time interval is the received signal approximately stationary? That is, what is the optimum time over which to sample the data to achieve a good estimate of the covariance matrix?

(2) Given a section of data, what is the optimum method of utilization? This is a trade-off between bias and statistical stability.

(3) How does one choose the frequencies at which to do spatial processing?

The duration of the period of quasi-stationarity is related to the speed and track of the source, and the resolution of the processing system in space and time. The resolution is a function of both array aperture and signal-to-noise ratio (SNR), since for adaptive processing the spatial

resolution depends on the SNR. For successful frequency analysis, the track and speed must be such that the frequency shifts encountered over the interval are not larger than the frequency resolution of the frequency analysis processor and the amplitude contributions of the various sources should not change substantially over the interval, thus changing the received power spectrum. Spatial processing assumes that the change in source location over the interval be no larger than the resolution of the spatial processor. Beyond this, some smearing in the spatial processor output will be evident. The spatial resolution of the MLM process is given in Fig. III-7 (adapted from Ref. 9.) as a function of signal-to-noise ratio. However, it is expected that the optimum processing interval will be determined experimentally or even, eventually, dynamically adapted to the scenario and environment.

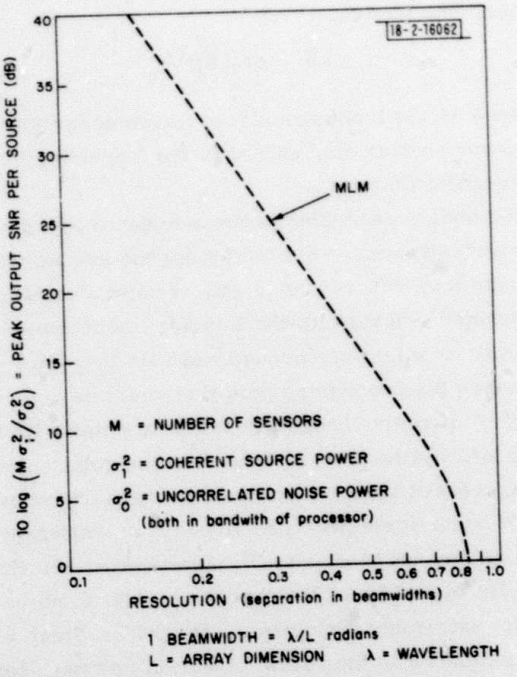

Fig. III-7. MLM resolution as a function of SNR.

In determining how the data in the assumed stationary time interval should be Fourier analyzed for input to the frequency domain spatial processor, the items to be considered are:

(1) The statistical stability of the spatial processor output,

(2) The bias of the spatial processor output,

(3) The signal-to-noise ratio and resolution of the spatial processor output.

As is commonly done for spectral analysis, the interval of data determined above, T, is divided into M subintervals of length $\tau = T/M$ for each channel. Each is windowed for spectral leakage control and then transformed to the frequency domain via the FFT algorithm. The spectral covariance at a given frequency is then formed by averaging outer products of the frequency samples across channels, as indicated in the previous section. The exact statistics of the spectral covariance matrix, and the final power vs azimuth estimates are available[4,5] for the case where M is equal to or larger than the number of sensors, and in Reference 6 for the covariance element in the general case. What is important here is that the greater the value of M, the more

stable the estimator. It is very important that enough averaging be done to eliminate spurious contributions to covariance matrix cross terms which result from two or more independent sources. This is just one aspect of stability of estimates but it is an important one for the maximum likelihood processor. The extremely poor results which can be obtained when this is not done are described in Reference 7.

It is not possible to arbitrarily increase M for a given piece of data of length T. The fundamental frequency resolution involved in the estimation of K as described above is roughly $\Delta f = 1/\tau$ Hz, which is further increased by the spectral windowing operation.[8] The detrimental effects of excessive processor bandwidth are threefold.

First, a bias or smearing in the spatial processing output occurs for broadband sources. This results from the relationship between frequency, velocity, and wavenumber for plane waves. The bias is:

$$|| \Delta\bar{\nu} || = \Delta f \, || \bar{\nu} || / f_o$$

where $\bar{\nu}$ is the true normalized wavenumber for the signal at frequency f_o, f_o is the nominal frequency of analysis, and Δf is the frequency increment by which the off-center signal component differs from f_o.

Secondly, excessive processor bandwidth also puts an additional burden on the spectral covariance estimator. Not only must the averaging eliminate cross terms between sour at the assumed (center) frequency, but it must also eliminate interference effects between adjacent frequencies leaking into the Fourier coefficients. Finally, in the case of very narrowband sources, as might be encountered with propellor-driven aircraft, excessive processor bandwidth decreases the processing gain that could be achieved at the frequency analysis processor by allowing excessive broadband noise to enter with the narrowband signal. Since the resolution of the MLM processor is related to the coherent signal to (uncorrelated in space) noise levels, this decreases the resolution of the spatial processing output.

To state some practical figures, a 9-element, 2-m-aperture uniformly sampled array was found to have an 11-deg 3-dB beamwidth at 100 Hz with an 8-dB input signal-to-noise ratio in the 4-Hz bandwidth of the processor. (Both noise and signal were broadband, thus the bandwidth was not extremely important to the SNR.) Since we are primarily interested in azimuthal resolution, smearing in the radial direction was not prohibitive until the bandwidth was increased to 16 Hz, leading to a $|| \Delta\nu || = 0.14$ spreading in radius. Since the time series were sampled at 2 kHz, this corresponds to a length 128 discrete Fourier transform. The beamwidth of such a system is large enough so that target motion is not a strong restriction (a 300-m/sec source on the horizon at 5 km with no radial component moves only 7 deg in 2 sec.) Also, the "integration" interval over which the covariance estimates are formed is limited by the desired wavenumber map output rate, tentatively set at 1 to 2 sec. Thus, to achieve a satisfactorily stable covariance estimate, 8 to 16 transforms of length 256 samples leading to 8 Hz bandwidth are recommended. In general, simulations (see Sec. III-C) have shown that the number of transformed blocks that may be successfully averaged is as low as four, especially in single source situations, but a minimum of eight is recommended. Also, increases in transform length over 512 points (4 Hz bandwidth at 2 kHz sampling rate) are diminishingly effective.

If the number of data blocks used to form the covariance matrix is too small, then the matrix will be singular. To overcome these numerical instabilities, a small amount of noise is added to each diagonal element of K, typically 0.001 or less of the smallest diagonal. This

makes the estimated covariance matrix non-singular and invertible. It also reduces its signal-to-noise ratio. It is best to use as large an M as possible, and stabilize only when necessary.

Ideally, the diagonal elements of the covariance matrix should all be the same. To achieve this, the estimated matrix can be normalized to equalize the diagonal (and presumably compensate for variation in calibration between the different channels of the signal collection system). However, differences in the diagonal values may reflect mostly random noise fluctuations. In this case, normalization may cause difficulties in the processing. This also mitigates in favor of sufficient averaging (large M) if normalization is to be used.

Finally, a computationally non-intensive method is required to determine the frequencies at which to perform spatial processing. Since most practical processing will be done when the log signal-to-spatially-uncorrelated-noise ratio at the input of the processor (at a given frequency) is a positive (below this the resolution is very poor), a reasonable method of choosing likely frequencies is to determine the peaks in the power spectrum of the received data. Thus, as the time series for each channel and block are transformed, they can be square magnitude summed into a power spectrum estimate. This operation is followed by peak detection, and the N largest peaks are used to form covariance matrices, where N is the number of frequencies for which there is adequate computer processing time. This procedure is one which will be further investigated, but it is clear that frequency selection is a difficult problem and may require a more sophisticated approach, possibly involving the results of the preceding analyses.

In implementing the operations outlined in Section III-A, many symmetries in the operands can be exploited to save computation time. These will be examined in detail in the course of forthcoming real-time implementation; the ideas are briefly outlined here. First, since the input data are real, Fast Fourier Transform algorithms specifically coded for real data should be used. If these are not available, a single complex algorithm may be used to transform two blocks of data simultaneously, and the results separated using the symmetry properties of the real and imaginary parts of the output. This results in a savings of a factor of 2 at this level. Next, when forming and dealing with the covariance matrices, the fact that they are Hermitian may be used to reduce computation by more than a factor of 2 in almost all instances, and reduce storage by exactly a factor of 2. Additionally, the inverse covariance matrices may be formed using an algorithm of order M^2, where M is the number of sensors, avoiding the order M^3 general inversion process. However, due to the relative coefficients, for small M the M^3 algorithm may be faster. Finally, for arrays with sensors uniformly spaced on a rectangular grid, a very large time savings in the evaluation of the quadratic forms may be achieved due to properties of the steering vectors. For the problem at hand, this yields of computational savings of a factor of 5 on the most time consuming part of the algorithm.

C. ACOUSTIC TIME-SERIES SIMULATION

To further evaluate the performance and determine the optimum processing parameters in the single-node bearing estimation algorithm, we have designed and implemented a simulator for the time series received at a node. The desirability of this capacity stemmed from two points: the lack of availability of long sections of digitized data and the need to know the underlying ensemble characteristics of the data to be processed. For instance, to check the resolution capability of the algorithm under operating conditions, it is necessary to be able to simulate arbitrarily located acoustic sources at various signal-to-noise ratios.

To this end, a first-order time series simulator adequate for stationary sources with arbitrary autoregressive (AR) spectra in uncorrelated white Gaussian noise was developed. The AR models can be arbitrarily specified by the user or can be developed from real data using the "Yule-Walker" or "Autocorrelation" method of all-pole model fitting. This procedure was chosen as a result of the guaranteed stability of the model it generated. Time series are simulated by driving the AR model with white Gaussian noise and delaying the sequence stored for each channel consistent with the array geometry and the source location in space with respect to the array. Because of the sampled nature of the time series, this "steering" entails an integral number of sample delays and no effort for precise delays via allpass linear phase filters has been implemented. The effect of this inattention to detail is to introduce into the simulation the errors resulting from sensor location uncertainties.

After several source sequences are generated, these can be combined with arbitrary amplitude and time relationships. Thus nonstationary data sequences can be simulated. Finally, the appropriate level of spatially uncorrelated white Gaussian noise can be added to simulate the sensor noise. The remaining gaps in the simulation software regard source movement and its attendant Doppler shifts, and the amplitude fading characteristics of the acoustic propagation channel. These lead to increasingly nonstationary received sequences.

This set of simulation and analysis software has been used to help obtain a number of practical results:

(1) Bandwidth considerations in the front-end frequency analysis processor were quantified, and the results of windowing the time-series data were determined. The decrease in spatial processing resolution as a function of spectral leakage on broadband sources was found to be in agreement with theoretical predictions.

(2) The decreases in performance and biases introduced in the output of the spatial processor due to sensor location uncertainty were observed.

(3) The ability to arbitrarily locate broadband sources allowed the determination of resolution characteristics for a particular array as a function of SNR and frequency.

(4) Theoretical results regarding the statistical characteristics of the spectral coherence matrix as a function of the amount of averaging and the underlying ensemble statistics of the data were verified.

(5) Confidence in the acoustic node as a viable detection and bearing estimation method was increased.

It should be noted that the simulation software discussed here adheres to time-series storage standards independently developed and used for the DARPA Vela project at Lincoln Laboratory. By doing this, we have been able to make a great deal of use of existing display and data manipulation tools which have been developed for other than DSN purposes.

REFERENCES

1. J. Capon, "High-Resolution Frequency-Wavenumber Spectrum Analysis," Proc. IEEE 57, 1408 (1969) DDC AD-696880.
2. R.T. Lacoss, "Data Adaptive Spectral Analysis Methods," Geophysics 36, 661 (1971) DDC AD-734104.
3. R.T. Lacoss, E.J. Kelley, M.N. Toksoz, "Estimation of Seismic Noise Structures Using Arrays," Geophysics 34, 21 (1969), DDC AD-688564.
4. J. Capon, R.J. Greenfield and R.J. Kolker, "Multidimensional Maximum Likelihood Processing of a Large Aperture Seismic Array," Proc. IEEE 55, 192 (1967), DDC AD-651722.
5. J. Capon, and N.R. Goodman, "Probability Distributions for Estimators of the Frequency-Wavenumber Spectrum," Proc. IEEE 58, 1785 (1970), DDC AD-723788.
6. D.E. Amos, and L.M. Koopmans, "Tables of the Distribution of the Coefficient of Coherence for Stationary Bivariate Gaussian Processes," Sandia Corporation Monograph SCR-483 (March 1963).
7. G. Duckworth, "Adaptive Array Processing for High Resolution Acoustic Imaging," Masters Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology (January 1980).
8. F.J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," Proc. IEEE 66, 51 (1978).
9. H. Cox, "Resolving Power and Sensitivity to Mismatch of Optimum Array Processors," J. Acoust. Soc. Am. 54, 771 (1973).

IV. SOFTWARE

The following three sections describe software development efforts related to the implementation of a data acquisition system and to the development of software which will be ultimately required for multiple-node real-time experiments during FY 80. The three sections describe the user-level data acquisition software, progress on system-level software to support data acquisition and future DSN experiments, and simple communication software for downline loading, dumping, and debugging remote nodes.

A. DATA ACQUISITION SYSTEM

DAS (Data Acquisition System) is the user-level data acquisition software for the first DSN data acquisition node. It is being written in C and is being developed on a PDP-11/70 running under UNIX. DAS itself will run on a PDP-11/34 under the Data Acquisition Kernel (DAK) described in Section IV-B of this SATS. Design of DAS has been completed and coding is under way. Descriptions of the planned user interface and the internal structure of DAS follow. The system will record digitized acoustic waveforms on magnetic tapes which are to be used for the development of detection and tracking algorithms and evaluation of acoustic capabilities.

1. DAS Operation

A DAS experiment will consist of digitally recording acoustic energy incident upon a microphone array over a period of time (typically 5 to 30 min.). To conduct a typical experiment, the operator must perform a well-defined sequence of actions. First, he must check and power up the DAS hardware, mount magnetic tapes, and downline load DAS from the PDP-11/70 (or bootstrap from magnetic tape). Next, a sequence of console commands must be issued. The commands can be keyed in at the operator's console or may be read from a UNIX file.

The operator controls DAS using a concise UNIX-compatible command language. Commands naturally fall into four classes: (a) setup commands, (b) record commands, (c) display commands, and (d) tape positioning commands. Syntax and use of the different kinds of commands are indicated in the following examples.

a. Setup Commands

Setup commands define the hardware environment to DAS and allow the operator to specify system parameters for an experiment. The parameters input by the setup commands will be recorded at the beginning and end of the magnetic tape file corresponding to the run. The commands that follow comprise a complete setup of the A/D converter, tape, date, and tape file header text.

`setad chan=0-9 rate = 2000`	The A/D converter is defined to have 10 active channels numbered 0 to 9. Sampling rate is 2000 scans per second at 10 channels per scan.
`date 01NOV79:10:35:00.am`	The date is set to Nov. 1, 1979. Time is 10:35 am.
`text Experiment #371. Site: ...`	Input up to 1024 bytes of text describing the experiment.

b. Record Commands

After the operator has set up DAS, recording may begin. The command "start" causes DAS to enter record mode and to record data (with sample rates, etc., as specified in the setup commands). Recording will continue until the command "stop" is given.

c. Display Commands.

At any time after setup, display commands may be issued to visually check system activity. Again, a typical sequence after recording has begun might be as follows:

status	Prints status of all hardware devices.
snapad	Prints a buffer of data from the A/D converter.
power	Prints power levels on each channel.
disp 1	Displays waveform data on channel 1.
^c	Stops display.

d. Tape Positioning Commands

This class of commands allow the tape to be positioned and examined. They may only be issued to a drive which is not under direct control of the recording process. Examples of these commands are:

drive 0	Assign tape drive 0 as the current tape drive.
file 3	Go to beginning of file 3.
rewind	Rewind tape drive.

2. DAS Design

The internal structure of DAS is simple. DAS consists of two processes running under DAK: (a) A user interface process which monitors the operator's console and interprets operator commands and (b) a recording process to supervise data flow from the A/D server to the magnetic tape server. The two processes communicate and cooperate via command queues and queueable command objects. More details of the interprocess communication mechanisms and queueable objects are given in Section IV-B of this SATS.

The following is a simplified description of the interaction between the two processes. When the user console process receives the start command, several things happen. First, the console process uses DAK to write a file header on the magnetic tape. Second, a start command object is constructed and sent to the recording process' command queue. Receipt is acknowledged. The console process can then return to interpreting further operator commands (e.g., "display" and "stop"). The recording process will record data until forced to halt by a stop command.

The recording process is responsible for receiving A/D buffers sent by the A/D server, copying data from full A/D buffers to empty magnetic tape buffers, sending full magnetic tape buffers to the magnetic tape server, and requeueing empty A/D buffers on the A/D server. The recording process also monitors tape-drive switching and will (when commanded by the user

interface), format and buffer A/D data for teletype or graphics output. The recording process must also monitor its own command queue and acknowledge commands sent to it by the user interface process. A more detailed description of the recording server is contained in Section IV-B where it is used to illustrate the interprocess communication services in DAK.

B. THE DATA ACQUISITION KERNEL

The Data Acqusition Kernel (DAK) is designed to support the Data Acquisition System (DAS) which will be used for acquiring single-node data early in FY 80. An enhanced version of DAK will support all DSN node software to be developed in FY 80.

As of 30 September 1979, DAK consists of 8,000 lines of code and 2,000 lines of short-form documentation. The code has been written, desk-checked, rewritten, and re-desk-checked, and testing and debugging are under way. The load and dump facilities which are described in Section IV-C of this SATS and which are needed for debugging are now becoming available for use.

The philosophy of DAK has been to build a real-time system that makes it easy to build small but sophisticated server processes. The reason for doing this is that a large portion of DSN software is being organized as servers which respond directly to requests. For example, the Data Acquisition System contains a recording server which does the actual work of recording data from the A/D converter to magnetic tape. Later this server will be expanded to copy data from the A/D to the array processor in a DSN node. We expect array processing software to be organized as a server that receives buffers full of data and computation requests and returns buffers full of processed results. Servers are also needed to load and dump memory, and to monitor the system.

Another part of the DAK philosophy has been to use more "core" memory in order to substantially decrease the cost of writing complex code. The economic basis for this philosophy is the decrease in memory prices by a factor of 4 in the last three years, the projected continuation of this trend, and the potential for increased programmer productivity when less constrained by memory limitations.

DAK is also a first step towards the development of a real-time network kernel. By a network kernel we mean a multicomputer distributed operating system kernel that includes processor schedulers, "core" memory managers, and communications servers, but not I/O servers or file system managers, and that allows real-time operating systems and standard operating systems to be built on top of it. Principal characteristics of such a kernel are that network communications are built into the system at a lower level than the I/O servers, that I/O servers are on a par with user processes and use the network communications to communicate with user processes, and all interprocess communication is through a network that behaves something like a FIFO of nontrivial length and delay.

DAK currently is designed to operate on a single computer in a single virtual address space. We are examining conceptual problems of interprocess communication without the need to confront problems of throughput, lost messages, and communication errors which will complicate the issue in distributed systems. The interprocess communication scheme in DAK is based on queueable objects which combine the attributes of messages, asynchronous procedure calls, and shared data structures. This communication scheme is easier to use than one based on simple byte stream FIFOs or message passing, and is better adapted to overcoming the reliability and buffer management problems which will need to be faced in a distributed environment with less than perfect communications. DAK, somewhat enhanced, will be sufficient for our initial DSN

feasibility experiments and demonstrations. We are undertaking the design of a second-generation system which would be a true multicomputer distributed system. This second-generation system will both support advanced signal processing and multisite tracking research and represent a serious investigation of basic issues of interprocess communication in a distributed environment.

1. Coding Facilities

We have attempted to make the coding of complex servers easier in DAK than in most operating systems. First, all code can be written in C, which is a proven productive higher level programming language. In addition, we have augmented C with an object-structured programming discipline supported by a few macros and subroutines. With this system, we have been able to bring to C many of the advantages of languages which more explicitly support object-structured programming, such as SIMULA 67 (Ref. 1), CLU (Ref. 2), and ECL (Ref. 3). Last, we have implemented a process scheduler that provides nice features for constructing servers.

a. Object-Structured Discipline

Our object-structured discipline is largely a notation for writing programs and short-form documentation. The short-form documentation consists mostly of formalized descriptions of the objects used or defined by a software module and services it will provide. Each module generally defines one, and occasionally two or three, new kinds of objects and the operations which can be performed on those objects. In our short-form documentation, statements, subroutine calls, and expression are all written as users would write them, subject to parameter replacement. Detailed descriptions and discussions of the notation, documentation format, and suggested programming techniques are available within our computer system.

The notation, documentation, and programming techniques largely deal with typed structured objects, which are structures whose first word is a type constant that specifies at run time the type of the structure. For example, there are queueable objects and queues, both of which are typed structured objects. We also make extensive use of typed structured object variables which are not structures themselves but are pointers to structures.

Allocation of objects to memory at run time is done by functions such as ob_local and ob_global for the local-procedure-call-stack and global-shared-among-everyone stack, respectively. For load time allocation, and allocation as elements of other structures, macros such as qu_alloc (to allocate a queue in this case) are used and, after allocation, the ob_init statement is used to zero the object. Qu_alloc and ob_init are both examples of generic functions of the form xx_alloc and xx_init which take on slightly different forms for different kinds of objects. Another example is pc_alloc which allocates a process at load time.

b. Processes in DAK

The following describes DAK processes with emphasis on how they differ from those of other operating systems. DAK processes are typed structured objects allocated and initialized in the manner usual for such objects by macros and functions such as pc_alloc and pc_init.

The process scheduler is an eight-priority-level scheduler based on the PDP-11 hardware. Priority levels are specialized numbers, called plevels, storable in pl_level variables. The priority levels are numbered from 0 through 7, with 7 being the highest priority, and the respective plevel values being pl_value(0) through pl_value(7). The macros pl_raise and pl_lower raise and lower priority. These macros include careful checks to verify priority level changes.

In DAK, device server processes are of the same kind as user processes. In particular, they each have their own stack. We have observed that complex device server processes, such as network servers, are very difficult to write as a set of interrupt routines, because after each interrupt the process state must be stored in a control block, and upon each interrupt the state must be recovered from the control block. We feel that using a proper, permanent process stack to save context allows a programmer to write a complex server in much less time than would be needed in an environment with no permanent server stack.

A process may stop itself by calling pc_stop. A stopped process may be restarted by an interrupt, or by a call made by another process to pc_start the stopped process. The hardware interrupt vectors are called ivectors, and may be connected to processes by functions such as iv_connect defined by the iv module. When the appropriate device flags are on, an interrupt occurs, and merely restarts the process connected to the ivector. Sixty-four software interrupt vectors, called svectors, are implemented. These function conceptually like the ivectors. Each has an associated start flag that may be turned on by the sv_start function to trigger an interrupt. The 64 flags are grouped 8 to each priority level. When a process is initialized, it is assigned a user-specified priority level, and automatically assigned an svector on that plevel. Pc_start takes a process as its one argument, and sv_start's the process's svector.

Assembly language interrupt routines are also supported and are used to make more intelligent device controllers where extra efficiency is required. Small assembly language interrupt routines are used by the A/D converter, terminal printer, and clocks.

This process scheduling system has several desirable features. First, it is efficient. For example, interrupts are never totally disabled for longer that 33 μsec on a PDP-11/34 with MOS memory. Second, it is possible to run user processes at priorities above device servers. This is most important for the terminal printer server, which, when handling 1000 character per second video terminals, can absorb most of the available CPU time. Therefore, this server is assigned priority level 2, while user foreground processes are typically assigned priority level 3. Another interesting feature is that processes do not have to raise their priority level in order to check for the completion of some event.

2. Interprocess Communication

In most operating systems, different mechanisms are employed by a user process for communicating with device servers than are used for communicating with other user processes. A distinguishing feature of our notion of a network kernel is that the mechanism used in both cases is the same. In other words, general interprocess communication is supported in the kernel system layer that is underneath the device server layer, and the device servers use this communication to communicate with users.

In DAK, the primary interprocess communication mechanism is the queueable object. A queueable object is an object with a special header that allows it to be placed on a queue. Processes have queues which behave like input ports that receive queueable objects sent by other processes. Queueable objects also have extra structure that can be used to make them behave like asynchronous procedure calls: specifically, a queueable object can be sent to a server by a procedure call defined by the server module; and can be marked by the sending user process so that after being processed the object will be returned to a queue specified by the user process.

Queueable objects combine the attributes of messages, shared data structures, and asynchronous procedure calls. By way of example, we will review below several uses for queueable objects in the Data Acquisition System.

a. Queueable Objects

The DAK queueable object communication system is designed to support device servers such as those found in operating systems, plus more complex servers likely to be found in a DSN. A queueable object is like a message which is typically (but not necessarily) sent to another process, modified by that process, and returned to the originating process. Following is a description of some of the major elements of queueable objects and of the major functions supplied to operate with them for inter-process communication.

Each queueable object has a qu_status element for common status flags, a qu_option element for common options, a qu_name element for giving the object a name (primarily for error diagnostic purposes), and a qu_server element for associating the object with a server's queue. The qu_retq and qu_done functions set the object so that when it is returned by the server it will either be put on a designated "done queue" or merely pc_start the user process. The qu_send function sends an object to its qu_server queue.

Server processes own server queues upon which user processes queue objects that are effectively requests. The server process pc_wait's for the qu_first element of one of its input queues to become non-zero. It then processes the element and qu_return's it to the user. If the user has preset the object via the qu_retq function before sending it to the server, the object will be returned on a user specified "done queue," and the user can merely pc_wait until the qu_first element of that queue becomes non-zero. Alternatively, the user can pc_wait till qu_done for the object returns non-zero.

Queues have associated priority levels, which are set by default to priority level 6. A queue cannot be accessed by a process whose current priority level is above that of the queue. Some of the qu module routines raise priority level internally to that of the queue, and disable interrupts at that level for about 100 μsec. Because the A/D converter hardware requires that priority level 7 not be locked out for such a long period, queues are not usable by priority level 7 processes.

Each queue is intended to have an owning process, qu_process. Other processes may place elements on the queue via qu_enq or qu_send, but only the owning process may do anything with elements already on the queue; except that the qu_done function may set an object element private to the qu module to cause a process to be pc_started when the object is qu_return'ed.

Each object in DAK is intended to be owned by only one process at a time. The owning process may pass the object to another process by qu_enq'ueueing the object on a queue owned by the other process. While it is possible to share variables between processes by other means in DAK, such sharing is discouraged because it would not be usable in a distributed system.

Servers are expected to provide modules defining the queueable objects accepted by the server, and provide functions to send these objects to the server. For example, the io module defines I/O operation, or ioop, objects, and function calls such as:

io_open (ioop)	Open device.
io_close (ioop)	Close device.
io_read (ioop, address, size)	Read into buffer.
io_write (ioop, address, size)	Write from buffer.
io_control (ioop, address, size)	Control using information in buffer.

The io servers are called devices. Each of the above function calls records the call arguments and type in the ioop and qu_send's the ioop to its qu_server device.

In effect, the queueable object system coupled with the object-structured discipline provides for delayed, queued, function calls, with optionally queued returns. This is a substantially easier system to use than one providing bare messages or simple byte streams. On the other hand, the user/server facilities of queueable objects need not be used when a bare message service is appropriate.

An important feature of DAK is that devices are themselves queueable objects and are passed around among processes. This is the equivalent of passing open files amongst processes in a distributed processing system.

b. Data Acquisition System Recording Server

The recording server in the Data Acquisition System is a real-time process that copies data from the A/D converter to a magnetic tape. This server illustrates two important points about the use of queueable objects.

The recording server uses two kinds of queueable objects internally: tape buffers and A/D buffers. A tape-buffer object begins with an I/O operation object, or ioop, which is the kind of object accepted by the tape server as a request to write a buffer as a tape record. The tape buffer further contains a 4K-byte data buffer and status parameters that specify how full the data buffer is. Similarly an A/D buffer object begins with an ioop for the A/D device server, and further contains a 2K-byte data buffer and status parameters specifying how empty the data buffer is.

The recording server owns a queue of empty tape buffer objects and another queue of full A/D buffer objects. It waits until both these queues are occupied, and then copies data from the partly full A/D buffer at the head of its queue to the partly empty tape buffer at the head of its queue. Whenever an A/D buffer becomes empty, it is sent to the A/D server to be filled, and whenever a tape buffer becomes full, it is sent to the tape server to be emptied. The A/D server passes full A/D buffers back to the recording server's queue of full A/D buffers, and the tape server passes empty tape buffers back to the recording server's queue of empty tape buffers. The important point is that the queueable objects are complex objects, and are more than just ioops. Conceptually, it is important in this application to associate an ioop, a data buffer, and extra status parameters as a single queueable object that is passed as a whole to various processes, even though some of these processes may not use the whole object.

The second point illustrated by the recording server is that server protocols are often not completely sequential. The recording server owns a third queue upon which it receives commands from its user. There are commands to start and stop the recording activity, and also a command to send the user a snapshot of current A/D data, possibly while recording is in progress. The start command object is returned to the user when the recording is initiated, in order to inform the user that recording has begun. In order for the user to find out when recording stops, the user must use a special wait-for-stop command, which is not returned until recording stops. If a wait-for-stop command is used before snapshot commands, the wait-for-stop command is often returned after the snapshot commands, and thus the recording server does not always return objects in the same order that it receives them.

3. DAK Device Servers

Many operating systems provide device servers which are either very basic, or which are not suitable for real-time applications. An example is the magnetic tape driver under DEC RT-11,

a small operating system for the PDP-11. The basic driver, which is intended for real-time applications, provides few functions not available to the user who directly controls the device hardware with no driver at all. A file management extension is provided for this driver, which provides tape labels and maintains a record of current tape location, but the tape label feature provides only part of the labelling required by the Data Acquisition System, and the tape location feature does not work properly if tape records are not 512 bytes long, a length too short for our application.

DAK device servers have the extra intelligence needed to make them suitable as general-purpose servers for real-time environments. We will briefly describe the main features of the DAK servers.

a. Magtape Server

The tape server includes a priority scheduler system, an automatic tape readying operation, an optional real-time retry mode, continuous position monitoring, and power-fail recovery. Each tape drive has a separate server process with its own queue of input/output operation requests, or ioops.

A user can associate a priority integer with every ioop. The tape controller is scheduled among tape drives using the priorities of the ioops at the head of each drive's queue. This is necessary so that labelling ioops on some tape drives do not interfere with real-time data writing ioops on other drives. Providing this feature requires that complex operations on one drive, such as a label writing operation involved in retries, be interruptable by higher priority operations on other drives. This feature also requires that several hardware inadequacies be overcome so that operations such as skipping off the end of tape are quickly timed out, and operations waiting for rewinding drives do not lock up the entire tape control.

There is a special ioop to check that a tape is ready for reading or writing, print a message if not, and wait for the tape to become ready. This ioop may be queued in front of label writing ioops, so that a user, such as the Data Acquisition System recording server process, does not have to instantiate a separate process to asynchronously ready a tape and write a label on it. On the other hand, normal ioops will consider it an error when the tape is not ready for them, thus warning the user process of any problems with the tape drive being accessed.

There is a special real-time option on write ioops. This option modifies the write retry strategy. Normally this strategy retries a write ten times, each time backspacing over the erroneously written record and then writing 3 in. of blank tape. With the real-time option in effect, the backspace and blank tape writing are suppressed and the user is expected to uniquely label each tape record so that missing and duplicated records can be detected when the tape is read.

b. A/D Server

The A/D server accepts control ioops that change the channels read and other parameters, plus ioops that read data. The server also keeps time using the system clock to establish an initial start time (to the nearest second) for the first read operation, and using the sample clock to maintain highly accurate relative time thereafter. When any read ioop is returned to the user, its io_time field contains the time the first sample was read by the ioop.

The A/D server goes to considerable software effort to switch buffers between consecutive read ioops without losing data, and to keep the sample clock time up to date even if the user does

not provide read ioops fast enough and data are lost, so that the loss of data can be detected by examining the io_time elements of returned read ioops. The queueable object mechanism of DAK allows many ioops to be queued in advance for the A/D server, and makes it easier for the user to use triple or quadruple buffering to avoid data loss.

c. Terminal Printer and Terminal Keyboard Servers

The terminal printer server is different from most DAK servers in that it does not deal with queueable objects, but rather with iobuffers which are queues of characters. Each iobuffer is similar to a virtual printer device to DAK user processes. However, several iobuffers may share a single real printer. Each line of output is prefixed by a "prompt" string of characters that tell which iobuffer the line came from. The stardard iobuffer is named io_console and has a null prompt string. Real-time processes use the io_attn iobuffer with the "ATTN:" prompt string. The io_attn iobuffer is intended for occasional messages by real-time processes. It has a special option which causes iobuffer full situations to be handled by throwing away characters until the printer has emptied the iobuffer, rather than by blocking the process using the iobuffer. The idea is that the only reason for overflowing io_attn is a long sequence of error reports issued by a real-time process, and that in this case only the first reports are likely to be of great interest. When characters are thrown away, the fact is carefully noted in the output listing at the point at which the characters were lost. Other iobuffers used by the printer server are io_keyboard, io_crash, and io_debug. For purposes of shared printer scheduling each iobuffer has an associated priority level.

The terminal keyboard server uses the io_keyboard iobuffer. Input characters from the terminal are placed in this iobuffer, from which they are echoed. The usual printer server policy of not printing a partially complete line is modified so that all characters in the iobuffer are printed immediately. The prompt string ">" is inserted automatically in the iobuffer as appropriate.

The keyboard server normally sends single complete input lines to user processes, but there is a facility for suppressing the printing action of the carriage return key that completes a line, so as to concatenate several logical input lines (each with its preceding prompt string) on one physical printer line. The inability of the keyboard server to deliver characters one at a time to the user process is intended to result in greater efficiency in a network environment.

In a real-time system, an input line is sometimes interrupted by output from an iobuffer such as io_attn. In DAK, the printer server handles this by printing #### at the end of the interrupted line to indicate it should be ignored, and then after printing the interruption from the io_attn iobuffer, reprints the input line so the user can continue.

C. COMMUNICATION SOFTWARE

CONP (CONnect to Peripheral computer) is a rudimentary communications system. It is a collection of software modules which run on a PDP-11/70 and, in part, on a peripheral computer. CONP is designed to link our central software development site with peripheral processors such as a data acquisition system or a DSN node. From the 11/70, an operator can downline load software, debug, update, and monitor remote program execution. This permits software staff to be located at only one control site. We plan to use CONP to develop the real-time network system for the FY 80 three-node experiment. CONP was developed on a PDP-11/70 running under UNIX. It is approximately 1000 lines of C code. The full CONP system for interprocessor

communication currently runs in two computers that are connected by two DL-11's in a serial link: the PDP-11/70 running UNIX, and the remote Data Acquisition System PDP-11/34.

CONP offers the user both character mode and packet mode communication. In the current DAS test facility, a 9600-baud line connects the 11/70 with the 11/34; character stream data currently runs at approximately 20 bytes/sec. Packet stream data runs at approximately 160 data bytes/sec. Both are useful and necessary for DSN applications. Character mode is used for examining 11/34 memory from the 11/70, for enabling 11/34 resident software to read and write 11/70 UNIX files, and for calling 11/70 programs from the 11/34. Packet mode is used for downline loading from the 11/70 and dumping 11/34 core images to 11/70 files. In FY 80, we will field deploy the data acquisition system and replace the 9600-baud line with a modem-coupled telephone line.

1. Character Stream Communication

Character stream communication is established by running two UNIX processes in the 11/70: a CONP input process receives characters from the peripheral computer, a CONP output process sends characters to the peripheral. These two processes communicate by sending signals through a shared UNIX file. For example, a character typed on the operator's console is buffered by a UNIX TTY server. The CONP output process reads the character from the TTY server and tests it. Depending upon the type of the character, the output process either writes it to the UNIX TTY server belonging to the 11/34 TTY port, or in the case of a command character, it takes appropriate action (e.g., stop transfer control to UNIX etc.). The TTY server sends the character to the 11/34. The CONP input process reads characters from the 11/34 TTY server and passes them to the operator's console TTY server.

Character stream communication has two principal DSN applications. First, to turn a remote user's teletype into a console that can communicate directly with the terminal interface in the PDP-11/34. Second, it enables a user of the 11/34 to call UNIX programs on the 11/70.

2. Packet Stream Communication

High-speed packet stream communication is provided by the runp module of CONP. It executes in both the PDP-11/70 and the PDP-11/34. Using runp, software on the 11/70 can be downloaded and run on the 11/34. For purposes of debugging software, the contents of 11/34 memory can be dumped to a file on the 11/70 where it can be examined, changed, and reloaded. For running software on the 11/34, CONP and runp are used together to enable the 11/34 to write data to and accept commands from the 11/70.

Runp is composed of two communicating processes: one process runs on the PDP-11/70 under UNIX, the other runs on the PDP-11/34. Communication between them is via checksummed packets. The 11/70 resident software consists of the following four modules: a command interpreter which accepts keyboard input, a packet transceiving module, a module which converts packets to files and vice versa, and last a module that enables runp to perform character mode communication. The 11/34 resident software consists of one module which interprets packets from the 11/70 as commands and executes them.

The runp command interpreter accepts a concise operator command set. Commands may be followed by at most one argument which must be a UNIX filename. The following are the runp operator commands.

load	filename	Checks if filename is in load format. If so, downline loads it to the 11/34.
start	filename	First loads filename to 11/34, and then transfers control to location 0.
dump	filename	Dumps 56K bytes of 11/34 memory to the UNIX file named.
return		Transfers control of the 11/34 to the DEC ROM console emulator.

3. Details of Downloading and Dumping

Following is a more detailed description of the tasks involved in downline loading the 11/34 from the 11/70 over a communication link. First, the operator starts executing runp on the 11/70. When the operator types "load filename," runp first examines "filename" for correct loader format. If no errors are found, character stream communication is established, via CONP, with the DEC ROM console emulator in the 11/34. The runp remote loader module (RLM) is then loaded and started in the 11/34. The file "filename" is then split into two parts: high- and low-core modules. The high-core part of "filename" is converted into checksummed packets. Each packet is 64 bytes long and consists of a header followed by data bytes to be loaded. The packet header is pictured in Fig. IV-1. A simple "handshaking" consists of STX and ETX characters sent by the 11/70 before and after the packet. The 11/34 responds to the correct

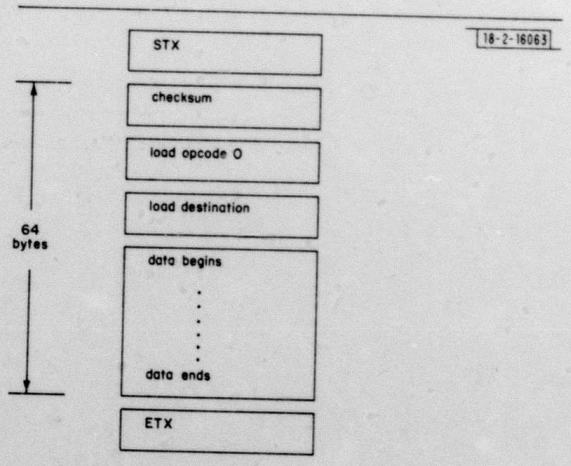

Fig. IV-1. Load-packet format.

receipt of a packet by an ACK, and negatively acknowledges with a NAK. The packet is sent to the remote loader module where it is unpacked, checksummed for errors and deposited in the correct 11/34 memory locations. When the high-core module has been loaded, control of the 11/34 is passed to the DEC ROM console emulator and the low-core part of "filename" is

downloaded in character stream mode. The low-core module is the same size as the 11/34 remote loader module and thus will overlay it.

A similar sequence of operations is performed when dumping 11/34 memory to an 11/70 file. *Rump* restores the low-core part of the dumped file, however. This is especially useful for diagnostic and debugging applications.

REFERENCES

1. J. D. Ichbiah and S. P. Morse, "General Concepts of the SIMULA 67 Programming Language," Annual Review of Automatic Programming 7, 65 (1972).
2. B. Liskov, A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanisms in CLU," Commun. ACM 20, 564 (1977).
3. B. Wegbreit, "The ECL Programming System," AFIPS Conference Proceedings 39, 253 (Fall 1971).

V. DATA ACQUISITION SYSTEM HARDWARE

The data acquisition system has been assembled and checked out, with the exception of the tape controller and tape drive which are currently attached to the 11/70 computer. The A/D Converter subsystem has been extensively tested and was found to have some wrongly located jumpers on the computer interface boards. Once these were corrected, the unit checked out without any further problems. We also experienced some DR-11B interface board failures which have been corrected.

The 11/34 is running under the control of the 11/70 computer, using a 9600-baud line connected to a teletype port on the 11/70 and the console port on the 11/34. We have also installed a simple remote capability to control the 11/34 power-down, power-up line by means of a DR-11C connected to the 11/70. This is used to put the 11/34 into console emulator mode and provides for complete restart after a crash without the need to physically move switches on the 11/34.

A 1200-baud telephone line has been ordered for installation between the 11/70 computer room and the flight facility. In addition, arrangements have been made for equipment space at the flight facility and a new power service is being installed. A 3 × 3 microphone array holder with a regular 1-m spacing has been constructed. This unit will be mounted on the sloping roof in front of the flight facility control room. An adjustable mounting platform is now being constructed to allow the array to be leveled on this roof. All the microphones have been tested, and the long cables required to connect between the roof-mounted array and the equipment rack have been delivered.

Currently the microphones are being suspended from the roof of an acoustic room, where they will be able to pick up sinusoidal sound waves being generated by a pair of speakers. This will enable us to test out the system in its final configuration before moving it to the flight facility.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ESD-TR-79-330	2. GOVT ACCESSION NO.: AD-A086800
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE *(and Subtitle)*: Distributed Sensor Networks	5. TYPE OF REPORT & PERIOD COVERED: Semiannual Technical Summary, 1 Apr – 30 Sept 1979
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Richard T. Lacoss	8. CONTRACT OR GRANT NUMBER(s): F19628-78-C-0002, ARPA Order-3345
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Lincoln Laboratory, M.I.T., P.O. Box 73, Lexington, MA 02173	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order 3345, Program Element No. 61101E, Project No. 9D30
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, VA 22209	12. REPORT DATE: 30 Sept 1979 13. NUMBER OF PAGES: 46
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Electronic Systems Division, Hanscom AFB, Bedford, MA 01731	15. SECURITY CLASS. *(of this report)*: Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

9D30

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

multiple-sensor surveillance system
multisite detection
target surveillance and tracking
2-dimensional search-space cell
acoustic, seismic, radar sensors
low-flying aircraft
acoustic array processing
high-resolution search-algorithms

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Research in the areas of tracking, signal processing, and system software is reported. A review of possible tracking techniques from the area of artificial intelligence was completed, and an approach selected for further development effort. Signal processing algorithms for measuring acoustic azimuths were further developed. Progress with software and hardware development of an acoustic data acquisition system, which will evolve into an experimental DSN node, is reported.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

207650 JM