Image Cover Sheet

CLASSIFICATION
UNCLASSIFIED

SYSTEM NUMBER
148368

TITLE
DEMONSTRATION OF ADVANCED TECHNIQUES FOR MULTI-RADAR TARGET TRACKING

System Number:
Patron Number:
Requester:

Notes:

DSIS Use only:
Deliver to:
DEMONSTRATION OF ADVANCED TECHNIQUES
FOR MULTI-RADAR TARGET TRACKING

File # 041ST.W7714-2-9637

Prepared by
T.M. Sibbald,
E.H. Buller
and
J.K.E. Tunaley

For the Defence Research Establishment Ottawa,
Shirley Bay, Ottawa, Ontario
Final Report under Contract File # 041ST.W7714-2-9637
between the Department of Supply and Services, Canada
and London Research and Development Corporation,
1495 Geary Avenue,
London, Ontario, Canada
N5X 1G6
Telephone (519)-438-7583

© DND Canada, March, 1993
CONTENTS

ABSTRACT.........................................................3
EXECUTIVE SUMMARY........................................4
INTRODUCTION...............................................6

TRACKING BY A SINGLE RADAR
  Introduction................................................8
  Summary of Simulation Programs............................8
  Detailed Description of Simulation Programs.............13
    SIMTRAKC..................................................13
    SIMTR1....................................................13
    SIMTRC2..................................................14
    SIMTRC3..................................................14
  Applications of the Transputer..........................17

COMBINATION OF TRACKS FROM MULTIPLE RADARS
  Introduction...............................................19
  Dempster-Shafer Association..............................19
  Kalman-Filter Updating....................................24
  The Effect of Biases......................................25
    Conversion to Geocentric Coordinates.................28
  System Track Display....................................29
  Applications of the Transputer..........................30
  Conclusions...............................................31

HOPFIELD NEURAL NETWORKS FOR DATA ASSOCIATION
  Introduction...............................................33
  Review of the Hopfield Network..........................33
  Results of Simulations..................................41
  Applications of the Transputer..........................45

CONCLUSIONS...............................................48

APPENDICES
  2A: Program listing for the simulator control
      program, SIMTRAKC.C........................................50
  2B: Program listing for the DREO radar simulator,
      SIMTR1.FOR................................................53
  2C: Program listing for the file conversion
      program, SIMTRC2.C........................................84
  2D: Program listing for the site tracker, SIMTRC3.C....93
  3A: Program listing for the system tracker, SIMTRC4.C..106
  3B: Program listing for the coordinate
      transformations program, GEOCONV.C.....................135
  4A: Program listing for the Aiyer neural network.....139
ABSTRACT

This report describes a demonstration system for combining track information from a number of radar sites. The site track information is used to form system tracks for the entire area surveyed by the radars. The system tracker is currently limited to accepting data from up to 10 radars and can track up to 20 targets. The target positions and velocities are displayed graphically using earth centred coordinates: latitude and longitude. Targets with and without identification information can be displayed using different colours.

The site tracking program is a track-while-scan (TWS) system that uses a relatively simple algorithm to associate observations with tracks. This association algorithm can only associate a target with one track and each track can only have one target associated with it. The track updating is performed using an α-β algorithm.

The system tracker uses the Dempster-Shafer method of data fusion to assign probabilities of association between site tracks and system tracks. Each system track is updated using information from site tracks with a sufficiently probable association. The track updating is performed chronologically using a Kalman filter.

The system tracker has been tested using the Dempster-Shafer method of data fusion with a non-zero uncertainty and also with the uncertainty set to zero (a Bayesian approach). No significant difference in the approaches was observed, however, reasons for preferring the Dempster-Shafer method are discussed in the report.

The problem of correcting biases in the site radars has been examined and an initial attempt at bias correction has been included in the system tracker. There are currently several unresolved issues concerning bias correction which are discussed in detail in the report.

The application of a neural network to the travelling salesman problem has been studied because it is related to the problem of associating data to tracks. A neural network that works well on the Travelling Salesman Problem (TSP) has been simulated and tested with several sets of ten cities. This neural network can be related to the data association problem and could be used by site and system trackers. The software simulation of the neural network is slow and does not offer a realistic alternative to conventional techniques. However, it should be possible to implement a neural network in hardware thus obtaining increased speed over traditional methods.
EXECUTIVE SUMMARY

Systems such as the North Warning System use many radars in order to survey a large region. Typically the area scanned by each radar overlaps with the regions scanned by neighbouring radar stations. This gives rise to the possibility of more than one radar observing a target. Because of this a system which combines the information from the various radar sites may be able to derive more accurate information. It also suggests that the errors in individual radars might be estimated and corrected.

In order to examine what is possible in a multiple radar system, a simulation of a system has been developed. The simulation, which runs on a PC, begins with a DREO radar simulator which produces simulated radar returns for the various sites and targets. The simulated returns are examined for each radar independently and tracks appropriate to the site observations are produced. The resulting tracks from each radar are then combined in a system tracker which checks for multiple observations of a target. When multiple observations of a target occur, the system track is updated using every observation. Thus the system tracks are the most comprehensive source of information in the entire system.

The results of the system have been inspected visually and matched what was expected. It was noted that, because of poor associations of some observations with system tracks, multiple system tracks could arise for a given target. This can be corrected by implementing a track merging algorithm which would use retrospective updating. This will improve the performance of the system.

The results from operating the simulation system are displayed on the screen with the user having full control of the display parameters. Typically a grid and radar sites will be displayed and the targets will appear as squares with a tail indicating the target velocity. The system can be set to colour targets which have identification numbers a different colour than those which do not. The system currently operates only in a low resolution graphics mode, however high resolution is possible. Currently the display is a relatively slow part of the system so various approaches to improving its speed are discussed.

Several key aspects of the multiple radar tracking problem can be studied with the system. The possibility of feeding system tracker information to the individual radar sites can be assessed. One instance in which this has a clear benefit was discovered during the simulator development. The discovered benefit arises because of "end of scan losses" that can occur in some tracking systems and are explained in detail in chapter 2.3.4. The use of system track information could be used to resolve the irregularities. Other benefits such as improved site track estimation would also be expected.
The radar error corrections that multiple observations of a target suggest are evidently very difficult to implement. This is because the errors can arise from errors in only one radar or a combination of errors in more than one radar. Further difficulties arise if some of the radars do not provide elevation or altitude information. This is because the coordinate transformations used by any multiple radar system require elevation information. In both cases further study is required to determine to what extent error correction is feasible.

The tracking of targets by an individual radar or a multi-radar system requires that an individual observation be associated with existing tracks. The speed of the association is quite important to the speed of the tracking system therefore a fast method of associating data would improve the system. In view of this, neural networks have been examined and their performance assessed for the association problem. The analysis was done by applying the Hopfield or mean field neural network to the travelling salesman problem which is related to the association problem. The travelling salesman problem is simply to find the shortest route which visits a set of cities.

A neural network which provides good solutions to the travelling salesman problem was simulated and this suggests that the association problem could be resolved using a neural network. However, the simulated neural networks are slower than traditional methods of solving the problem. This means that the neural network would have to be built as a special piece of hardware. The development of a PC driven neural network board would be very useful for further examination of this and other neural network problems.

The use of transputers for the purposes of increasing the speed of the system tracker and simulated neural network has been considered and no advantages are currently seen. For the system tracker this is due to the problem not being particularly well suited to parallel processing and because of the amount of time that would be required for communications between the system tracker and the transputer. The application of the transputer to the simulated neural network would be no faster than traditional methods. Hardware implementations that would be advantageous are a special display for the system tracker (so that the tracking and displaying are performed simultaneously) and a PC driven neural network board.
CHAPTER 1

INTRODUCTION

Surveillance of a wide area can be achieved by using many radars but, in order to obtain a complete picture, the data from the radars must be combined in a central system. This report describes a demonstration system, designed by London Research and Development Corporation, which combines track information from a number of radar sites into a set of tracks for the entire area surveyed by the radars. The system can be used to simulate the North Warning System. This system has been tested with simulated land based radars but its application could be modified to include other sensors.

At each radar site, observations are collected and used to form track information. This process of tracking targets with a single radar is referred to as "site tracking" in this report. Typically, site tracking includes associating observations with existing tracks, updating the existing tracks, deleting tracks and forming new tracks. Blackman, 1986, has written a comprehensive review of site tracking.

When site track information is combined it is natural to form a set of "system tracks". These tracks reflect all the information gathered from radar sites about a particular target. In this report, the tracking of targets using information from all sources is referred to as "system tracking". System tracking can be broken down into the same general tasks as site tracking. Site tracks from the individual radars are associated with the system tracks which are then updated. The system tracker must be able to delete poor system tracks and create new ones.

In the simulation system, the site trackers and the system trackers operate in slightly different fashions. The site trackers operate as track-while-scan systems. This means that the time between observations of a target is approximately the scan period of the radar observing the target. Every target is updated in every scan even if no observation is associated with the track and updating is naturally chronological. The system tracker updates a system track only when it receives data that is associated with the track. Therefore the time between updates is variable. It is likely that updates in the present simulator will occur at least once every scan period, as long as the site track is not deleted, because the site tracker will produce information every scan period. Also because the system tracker receives information from multiple sources extra care is required to ensure that the updating is chronological.

Another difference between site tracking and system tracking is the amount of clutter that is present. Site trackers will observe clutter and must separate the target observations from the clutter. The system tracker obtains the track information from the site trackers and little, if any, clutter is present.
The system tracker is therefore less likely to obtain a false track. The presence of high levels of clutter at a site radar tends to disrupt the site tracks which can lead to poor track information being passed to the system tracker.

Chapter 2 contains information on how site tracks are simulated. This includes descriptions of the programs used to simulate radar data and perform the site tracking. The site tracker uses the $\alpha$-$\beta$ algorithm in geocentric coordinates for track updating and a simple data association algorithm. The data association assigns at most one track to each observation and one observation to each track.

Chapter 3 is a detailed description of the system tracker which accepts track information from the individual radars. The tracks from different radars are associated using the Dempster-Shafer method of data fusion. This may be thought of as a type of probabilistic association. The system tracks are updated using a Kalman filter but with variable time steps. Each system track is updated using the information from site tracks if the tracks have a probability of association above a threshold. The update is performed after all associated information has been organised chronologically.

In Chapter 4, the possible application of neural networks to the data association part of the tracking problem is discussed. Literature concerning the application of neural networks to optimisation problems is reviewed with emphasis given to the application of the Hopfield or mean field network. The most frequently studied optimisation problem is the Travelling Salesman Problem (TSP) but some study of the data association problem in multiple target tracking has been done. The results of simulations for the TSP are presented.

Chapter 5 contains conclusions from the completed work. Several key areas for future work are mentioned and general improvements are outlined.

REFERENCES

CHAPTER 2
TRACKING BY A SINGLE RADAR

2.1 Introduction

This chapter describes the programs used to simulate tracks from individual radars. Detections at a radar are simulated using a simulator supplied by DREO which has been modified to run on IBM compatible PC's. The output from the simulator is converted to a format suitable for the London R&D tracking program. This primarily involves a transformation from range, azimuth and elevation to geocentric coordinates (latitude, longitude and altitude). The site tracking program is run independently with the data from each radar site. Each run of the tracking program produces the predicted positions of the targets that are tracked at the site analysed. Once all the site tracks have been simulated the system tracker is applied. The details of the system tracker will be discussed in the next chapter.

2.2 Summary of Programs

A suite of programs which create simulated track data from individual radars has been packaged, along with the system tracker, to work as a single unit. The files used and their purposes are summarised below. Note that an extension of "FOR" means the program is written in Microsoft FORTRAN and an extension of "C" means that the program is written in Borland Turbo C++. It should be noted that where possible ANSI C conventions have been used and we are not aware of any C++ specific commands. Some of the C code is specific to DOS; in particular the control program will only work in a DOS environment.

SIMTRAKC.C - This is the control program which executes the other programs in the appropriate order. This is the only program that needs to be run; the other programs will be called by this one.

SIMTR1.FOR - This program is the DREO radar simulator modified to run on the PC. This program is limited to 10 radars and 16 manoeuvres per target.

SIMTR2.C - This program converts the output of the DREO simulator to a format suitable for use by the LRD tracking program. This program can process up to 20 targets observed by 10 radars and 5000 detections per radar.

SIMTR3.C - The London Research and Development Corporation site tracking program. This program predicts the location of targets during the next scan. The program limitations are 100 radars, 100 detections per scan and a maximum of 100 tracks at any one
time.

SIMTRC4.C - The multi-radar system tracker. This program combines the site tracks, performs system tracking and displays the system tracks. A Video Graphics Adapter (VGA) is required. A maximum of 10 radars and 20 system tracks are allowed. The latter constraint must take into account that tracks are not merged and multiple tracks can arise for a target. This will be discussed in detail in the next chapter.

SIMTRAKD.DAT - A parameter file used by the SIMTRAKC system. The parameters direct the input and output of each program as well as the system tracking, cross-referencing and display parameters. An example is shown in Table 2.1 below. Note that the filenames and prefixes must begin in the first column.

LRDFPLAN.2 - This is an example of a modified flight plan file for the DREO radar simulator. The modification is the addition of a likelihood of receiving an identification number from the target. A simple example of one target travelling in a straight line is given in Table 2.2.

LRDRAD.1 - This is the file containing the radar parameters for the DREO radar simulator. An example containing parameters for two radars is shown in Table 2.3.
TABLE 2.1

Example of the Simulation Parameter File, SIMTRAKD.DAT.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRDFPLAN.3</td>
<td>FLIGHT PLAN FILE</td>
</tr>
<tr>
<td>LRDRAD.1</td>
<td>RADAR PARAMETER FILE</td>
</tr>
<tr>
<td>TEMP0001.DAT</td>
<td>RADAR SIMULATOR REPORT</td>
</tr>
<tr>
<td>TEMP0002.DAT</td>
<td>RADAR SIMULATOR DATA</td>
</tr>
<tr>
<td>TEMP0003</td>
<td>DATA CONVERSION OUTPUT FILE PREFIX</td>
</tr>
<tr>
<td>TEMP0004</td>
<td>SITE TRACKER OUTPUT FILE PREFIX</td>
</tr>
<tr>
<td>TEMP0005.DAT</td>
<td>SYSTEM TRACKER DATA</td>
</tr>
<tr>
<td>200.0</td>
<td>MID-SCREEN TO TOP (KM)</td>
</tr>
<tr>
<td>100.0</td>
<td>GRID SIZE (KM)</td>
</tr>
<tr>
<td>43.0</td>
<td>LATITUDE OF SCREEN CENTRE</td>
</tr>
<tr>
<td>-80.2</td>
<td>LONGITUDE OF SCREEN CENTRE</td>
</tr>
<tr>
<td>30</td>
<td>GRID COLOUR</td>
</tr>
<tr>
<td>254</td>
<td>RADAR SITE COLOUR</td>
</tr>
<tr>
<td>127</td>
<td>TARGET COLOUR (WITH ID'S)</td>
</tr>
<tr>
<td>191</td>
<td>TARGET COLOUR (WITHOUT ID'S)</td>
</tr>
<tr>
<td>200.0</td>
<td>TARGET TAIL LENGTH FACTOR</td>
</tr>
<tr>
<td>3</td>
<td>TARGET HEADSIZE</td>
</tr>
<tr>
<td>0.0</td>
<td>TIME INTERVAL BETWEEN PLOTS (S)</td>
</tr>
<tr>
<td>6.0</td>
<td>TIME INTERVAL BETWEEN SYSTEM UPDATES</td>
</tr>
<tr>
<td>0.8</td>
<td>PROBABILITY THRESHOLD FOR ASSOCIATION</td>
</tr>
<tr>
<td>0.7</td>
<td>MAX UNCERTAINTY TO START NEW TRACK</td>
</tr>
<tr>
<td>15</td>
<td>SYSTEM TRACK DELETION TIME (S)</td>
</tr>
<tr>
<td>0.0</td>
<td>INITIAL PROBABILITY OF ASSOCIATION</td>
</tr>
<tr>
<td>0.0</td>
<td>INITIAL PROBABILITY OF NON-ASSOCIATION</td>
</tr>
<tr>
<td>1.0</td>
<td>INITIAL UNCERTAINTY</td>
</tr>
<tr>
<td>0</td>
<td>HISTORY (0) OR VELOCITY (1)</td>
</tr>
<tr>
<td>1</td>
<td>NO GRAPHICS (0) OR GRAPHICS (1)</td>
</tr>
</tbody>
</table>

The following parameters are listed as follows:
Range bias (m), azimuth bias (deg), elevation bias (deg), standard deviation of position (m), standard deviation of velocity (deg)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 0</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 1</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 2</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 3</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 4</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 5</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 6</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 7</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 8</td>
</tr>
<tr>
<td>0.0 0.0 0.0</td>
<td>1000.0 10.0</td>
<td>RADAR 9</td>
</tr>
</tbody>
</table>
### TABLE 2.2
Sample Flight Plan for the Radar Simulator.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678901234567890123456789012345678901234567890123456789012345678901234</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FLIGHT PLAN**

SAMPLE MG SCENARIO

**NUMBER OF TARGETS** 1.0

**NUMBER OF PATCHES** 0

<table>
<thead>
<tr>
<th>GMT OF EV EVENT ID</th>
<th>ACCEL FWD&gt;0</th>
<th>ACCEL LFT&gt;0</th>
<th>RATE UP&gt;0</th>
<th>LONGITD EAST&gt;0</th>
<th>LATATUD NORTH&gt;0</th>
<th>INITL ALTD</th>
<th>INITL SPEED</th>
<th>EON+</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEGMMSS.SS</td>
<td>M/S/S</td>
<td>M/S/S</td>
<td>M/S</td>
<td>DEGMMSS.S</td>
<td>DEGMMSS.S</td>
<td>METRE</td>
<td>KM/HR</td>
<td>DEG</td>
</tr>
</tbody>
</table>

**SIGMA B ID LIKELIHOOD**

| 01AP00000020.000 | 0. 0. 0. -814200.0 | 413000.0 | 5000. 400 80. |
|------------------|---------------------|----------|------------|------|
| VA0010530.000    | 1.0                 |          |            |      |

**NO MORE TARGETS**

<table>
<thead>
<tr>
<th>GMT OF EV EVENT ID</th>
<th>CLUTTER PATCH DIMENSIONS EAST&gt;0</th>
<th>LONGITD</th>
<th>LATATUD</th>
<th>INITL</th>
<th>INITL</th>
<th>HDG</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEGMMSS.SS LENGTH DEPTH WIDTH</td>
<td>DEGMMSS.S</td>
<td>DEGMMSS.S</td>
<td>METRE</td>
<td>KM/HR</td>
<td>DEG</td>
<td></td>
</tr>
</tbody>
</table>

**DENSITY SIGMA**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

11
### TABLE 2.3

Example of Radar Parameter File.

```
#RADARS
RADAR LONGITUDE (DEG:MIN:SEC)  
RADAR LATITUDE (DEG:MIN:SEC)  
RADAR HEIGHT (M)               
RADAR ANTENNA ELEVATION POINTING
ANGLE(DEG)                     
ANTENNA SPEED OF ROTATION(Revs/Min) 
RADAR PEAK POWER(KW)           
INITIAL ANGULAR POSITION OF     
RADAR ANT.(DEG)                
AZIMUTHAL BEAMWIDTH(DEG)       
ELEVATION BEAMWIDTH(DEG)       
RADAR WAVELENGTH(M)            
PROBABILITY OF DETECTION AT     
RADAR RANGE LIMIT              
PROBABILITY OF FALSE ALARM     
RANGE BIAS (M)                 
PULSE WIDTH(SEC)               
AZIMUTH BIAS (DEG)             
PULSE REPEITION FREQUENCY(HZ)   
ELEVATION BIAS (DEG)           
RADAR RECEIVER NOISE FIGURE(DB) 
RADAR SYSTEM LOSSES(DB)        
MAXIMUM TIME FOR FALSE ALARMS(SEC)
NUMBER OF PULSES FOR COHERENT
PROCESSING                     
RADAR LONGITUDE (DEG:MIN:SEC)  
RADAR LATITUDE (DEG:MIN:SEC)   
RADAR HEIGHT (M)               
RADAR ANTENNA ELEVATION        
POINTING ANGLE(DEG)            
ANTENNA SPEED OF ROTATION(Revs/Min) 
RADAR PEAK POWER(KW)           
INITIAL ANGULAR POSITION OF     
RADAR ANT.(DEG)                
AZIMUTHAL BEAMWIDTH(DEG)       
ELEVATION BEAMWIDTH(DEG)       
RADAR WAVELENGTH(M)            
PROBABILITY OF DETECTION AT     
RADAR RANGE LIMIT              
PROBABILITY OF FALSE ALARM     
RANGE BIAS (M)                 
PULSE WIDTH(SEC)               
AZIMUTH BIAS (DEG)             
PULSE REPEITION FREQUENCY(HZ)   
ELEVATION BIAS (DEG)           
RADAR RECEIVER NOISE FIGURE(DB) 
RADAR SYSTEM LOSSES(DB)        
MAXIMUM TIME FOR FALSE ALARMS(SEC)
NUMBER OF PULSES FOR COHERENT
PROCESSING                     
```
2.3 Detailed Description of Programs

2.3.1 SIMTRAKC

The system control program, SIMTRAKC, checks that the necessary files exist and then proceeds to execute the simulation programs in the correct order. The source code is listed in Appendix 2A. The files that must exist before running the simulation are: SIMTRAKD.DAT as well as the flight plan and radar parameter files named in SIMTRAKD.DAT. Note that the flight plan and radar file names (as well as the other file names in SIMTRAKD.DAT) are read in one character at a time until a space occurs. This means that the file names must begin in the first column.

If any file is not found, the control program halts execution and prints a message stating which file was not found. If all the files are found, the simulation proceeds with the programs called in the same order as they are described below. Any errors produced by the programs will be detected and cause the control program to halt the system.

Currently, the simulation programs can handle up 20 targets in the flight plan and 10 radars. However, it should be noted that targets can produce more than one system track. Therefore the limitation of 15 system tracks must also be considered.

2.3.2 SIMTR1

The DREO radar simulator has been modified to run on a PC and renamed to SIMTR1. A complete listing of the source code is supplied in Appendix 2B. Only the modifications to the simulator are discussed here. Full details on the operation of the simulator can be obtained in the documents listed in the references.

The first modification applies to the target identification numbers (ID) in the flight plan file. The target ID's should begin at zero and include all integer values up to and including one less than the number of targets. For example, if three targets are listed in the flight plan, the ID values should range from 0 to 2.

The targets do not have to be in numerical order according to the ID value. Consider three targets with ID's of 0, 1 and 2. The target information in the flight plan file can be in any order. The order can be 0,1,2 or 1,0,2, etc. The only requirement is that when three targets are in the flight plan, the ID values are 0, 1 and 2.

The likelihood of a site observing an ID from a target has been added to the flight plan. The value is given on the flight plan line following the simulator's target ID number.
Two output files are created by SIMTR1. The first contains the detections and the second contains a few of the radar parameters. The names of these files are specified in SIMTRAKD.DAT.

The radar simulator supplies range, azimuth, elevation, Doppler and relative velocity information for each target. It has been assumed that only range and azimuth will always be available and elevation may be available. The Doppler and relative velocity values have not been used in the simulation system.

2.3.3 SIMTRC2

The conversion program, SIMTRC2, reads in the data from SIMTR1 and produces output files, one for each radar, that can be read in by the site tracking program, SIMTRC3. The output file name prefix is specified in SIMTRAKD.DAT. The file extensions are 0, 1, ... up to one less than the number of radars. The source code for SIMTRC2.C is listed in Appendix 2C.

The detections at each radar site are sorted chronologically and converted to geocentric coordinates (latitude, longitude, altitude). Each detection is written on a separate line in the output file. The output data for each detection consists of the time, the latitude, longitude, altitude and, when applicable, an ID. The Doppler and relative velocity information supplied by SIMTR1 is excluded.

The likelihood of the target ID being received at a site radar is specified in the flight plan on the line following the simulator target identification. The occurrences of ID observations are determined by this program using the random number generator. The occurrences are independent for each observation and for each site. If the target ID should never appear for a target, set the likelihood to zero, conversely if it should always appear set it to one.

2.3.4 SIMTRC3

The site tracking program, SIMTRC3, operates as a Track-While-Scan (TWS) system. Thus tracks are updated during each scan regardless of whether the target was observed. New data is associated with existing tracks using ID numbers when they are available and a nearest neighbour approach when they are not. The site tracking is performed using geocentric coordinates. The assumption is made that the effects of the earth's curvature are negligible for the distances travelled by the targets during one scan time. The source code for SIMTRC3.C is listed in Appendix 2D.

The tracking program begins by reading SIMTRAKD.DAT for the
prefixes of the input and output files. The radar rotation rates are read in from the radar parameter file named in SIMTRAKD.DAT. SIMTRC3 generates a separate file of track data for each radar with the file name prefix specified in SIMTRAKD.DAT. The filename extensions are 0, 1, ... to one less than the number of radars.

Once the radar observation data is obtained, the program is prepared to begin its main function which is to track the targets detected by each radar. The program calculates the tracks for each radar independent of the tracks and observations made at the other radars. For each radar, a scan is read in and, following association with tracks (where possible), the tracks are updated. The data for which association is not feasible are stored for 3 further scans after which time they have either formed a track or they are discarded. This is repeated until all the data for the radar has been processed. The program ends when all the data from all of the radars has been processed.

The data association tries to match data from a scan with track predictions from existing tracks and unmatched detections from the last scan. The unmatched detections may be used to initiate new tracks; this is done as part of the track updating function. The algorithm begins by associating data with ID numbers to tracks with the same ID number. This association is performed regardless of the implied velocity of the target.

When ID numbers are not available the association algorithm is the one described by Blackman, 1986, on page 95 as "Suboptimal Solution One". The algorithm is suboptimal in that it does not always produce the associations that minimise the total error. Associations are assigned on a one-to-one basis where a detection can only be associated with one track and a track can only be associated with one detection. Some tracks may not be associated with any detections and, similarly, some detections may not be associated with any tracks.

When ID's are not available, the algorithm uses the distances between the measured target positions and the predicted positions from the tracks. If a target is to be associated with a specific track, the measured position of the target must be close to the position predicted at the last scan. This is done by checking that the distance between the two positions is less than a gate value. The gate value eliminates unlikely pairings and restricts associations to targets with an average acceleration of less than 12g over the scan time.

When a detection is to be matched to a previously unmatched detection, two gates are set according to the acceptable range of target velocities. The gate values represent a minimum velocity of 90 m/s or 180 knots and a maximum velocity of Mach 3.

The site tracker is a track-while-scan system which can suffer from an "end of scan loss" due to a fault in the association.
This loss is an important example of how a system tracker may be able to assist a site tracker. The loss occurs because a target which moves appropriately during a scan (one period of rotation) of the radar may be unobservable for one scan (even with perfect detection). This can cause the target track to decay and a new track to be initiated. Because a new track is initiated the information about the target contained in the decaying track is lost.

The end of scan loss occurs when a target passes through the end-of-scan-line. Consider the case of the antenna which points north at the end of each scan and is rotating clockwise. If a target is located slightly west of north, it will be observed at the end of a scan. Let this scan be called scan 1. If the target has a westward component of velocity, and passes through north before the end of the next scan, it will not be observed at the end of scan 1+1. The target will be observed at the beginning of scan 1+2. This represents a time difference between observations of slightly more than a scan time: the additional time being due to the movement of the target. The tracker predicts that the target will be observed exactly one scan time after the last observation. Therefore the predicted location is at the end of scan 1+1, but the observation is at the beginning of scan 1+2. This means that the prediction cannot be associated with the correct observation and the observation at the beginning of scan 1+2 will not be associated with the correct track.

The unassociated observation is used to create a new track which is updated on subsequent scans because observations are present that can be associated with this track. The track that predicts observations at the end of the scan will be deleted when no new observations have been associated for a number of scans. The criteria for deleting old tracks and recording new tracks, which is discussed later, has been set so that the new track with the target at the beginning of the scan is not recorded in the output until the old track with the target at the end of the scan is deleted. This insures that only one track appears during occurrences of this loss.

The end of scan loss occurs in the site trackers because they are track-while-scan systems. The problem does not arise in the system tracker because the system track updating only occurs when an observation is received. One method of avoiding the loss is to use information from the system tracker to correct the site track when this loss occurs.

Following the association, those observations which have been associated with tracks are used to update the tracks. Track updating is based on the α-β filter, described by Blackman, 1986, which can be regarded as a simplified version of the Kalman filter. This filter is applicable to a target moving in a straight line at constant speed though application of the filter to Raid Tracking Trials (RATT) data illustrated that manoeuvres can be followed reasonably well. Following the update, the
predictions for the next scan are returned for use by the association function once the observations for that scan are obtained.

The values of $\alpha$ and $\beta$ are set using a technique based on the tracking index described by Kalata, 1984. The tracking index is proportional to the ratio of the manoeuvre noise to the measurement noise. The measurement noise is obtained from the mean squared difference between the track prediction and the associated observation. The manoeuvre noise is obtained from the mean squared difference between the old and new estimates of the target velocity divided by the scan time. Thus the tracker is adaptive since the value of the tracking index depends on the observations.

Any track that has not been associated with a detection for 5 scans is deleted. During the time when no detections are matched to a track, the track is updated using the predictions based on the previous detections.

The data from the tracks are written to the output file in chronological order. The data from a track is not written to the file until that track has been updated at least 5 times. This allows reasonable estimates of the velocities to be formed. Also, as mentioned above, when an end of scan loss occurs and a new track needs to be created, the new track is recorded when the old track is deleted.

2.4 Applications of the Transputer

All of the programs mentioned in this section are appropriate for parallel processing in the current application. This is because the simulation of tracks and site tracking is specific to each radar. The system could, for example, use a transputer for each radar to perform these operations. However, the programs are used to simulate site tracks which does not need to be performed fast. Therefore the application of the transputer to the simulation of site tracks has not been pursued.

REFERENCES


"Scenario Plot Generator (SPCDML.221H)". Prepared by Hitech
Canada Limited and modified by A.W. Bridgewater, April, 1980. This document describes the flight plan file and radar file required by the DREO radar simulator program. It does not include documentation of the likelihood modification.
CHAPTER 3
COMBINATION OF TRACKS FROM MULTIPLE RADARS

3.1 Introduction

The performance of a tracking system can be improved by utilising information from multiple radars. This is the case with overlapping radars where more than one radar may detect a specific target. If the target is being evasive or the probability of detection is low, then the combination of information from multiple radars can greatly increase the tracking performance.

To demonstrate the combination of radar information from multiple sites a system tracker has been developed. The system receives predicted state vectors for each track generated by the site trackers. Thus some smoothing of the data and noise removal has already been performed by the site trackers. The state vectors are associated with system tracks using the Dempster-Shafer method of data fusion; this can be considered to be probabilistic data association. Following the association new system tracks may be established and old tracks may be updated.

The system tracks are updated using a Kalman filter. Unlike the site trackers the time scale for updating is variable. This is necessary because of the differences in antenna pointing angles and radar rotation rates of the different sites. Thus the updating is done using a variable time step which is the difference between the time of the last update and the time of the observation. Track deletions occur when the track has not been updated for a user specified amount of time.

The system tracks are displayed on the screen using several parameters which control colours and the grid size. The grid is composed of squares and is calculated according to the latitude and longitude of the centre of the screen. Thus along the horizontal and vertical lines through the centre of the screen the distance corresponds to the change in longitude and latitude respectively. It should be noted that since the screen, and hence the grid, is flat some errors due to the curvature of the earth can be expected in the corners of the screen. These errors can be kept small by limiting the area displayed.

3.2 Dempster-Shafer Association

The locations and velocities of targets predicted by the site trackers will usually have less noise than the true radar observations. Thus, it is generally preferable to perform the association of site tracks with system tracks using the information predicted by a site tracker. The predicted information and the system tracks are associated using the
Dempster-Shafer method of data fusion.

The Dempster-Shafer method of data fusion is a practical means of combining evidence from multiple sensors or pieces of evidence from one sensor. The theory, described by Shafer, 1976, utilises probabilities based on belief functions rather than statistics. Unlike statistics, the estimates of belief can apply to the truth of hypotheses and allow ignorance to be treated in a quantitative fashion. Like the Bayesian approach, in which probabilities must be established, some means of arriving at the extent of belief in single pieces of evidence must be found.

The Dempster-Shafer (DS) method of data fusion resembles the Bayesian method. While the Bayesian method is based on accepted and well-founded concepts of probability theory, the D-S rules of combination are ad hoc. The DS method was introduced in order to overcome the problem of assigning prior probabilities in the Bayesian approach; sometimes these are difficult to estimate. This is achieved by invoking an "uncertainty", which, like a probability is given a value between 0 and 1. When the uncertainty is zero, the DS and Bayesian methods are identical. The uncertainty can be regarded as the possible "error" in a probability assignment. Because the DS rules are not fully consistent with those of probability theory, the theory is often justified by saying that it manipulates "degrees of belief" in a hypothesis. However, it is best regarded as just a practical method of data fusion.

When observations are made, they are usually consistent with at least two distinct hypotheses but with different probabilities or levels of belief. As more observations are made, there is often a tendency for the belief in one of the hypotheses to become firmer at the expense of the others. In the Bayesian approach, the several possible hypotheses are often given equal prior probabilities and, as more data is received, these probabilities are updated according to the Bayesian rules. After some time, memory of the prior probabilities decays and the results become independent of the priors. When the priors and the subsequent probabilities used in the Bayesian method are known, the method is optimal. However, when the priors and other probabilities are not reliable, the results can be seriously in error. This will typically occur when some unforeseen or unlikely event happens and this has not been considered explicitly.

The DS method may be sub-optimal (non-zero uncertainty) but is more robust. The greater robustness is introduced by the uncertainty. At the beginning of the data collection, the uncertainty can be set to 1 which avoids the problem of assigning priors. During the data collection process, there is typically a reduction in the uncertainty and a tendency for the belief in one hypothesis to grow. If the uncertainty does not decay to very small values, the DS method provides some protection against unforeseen events.
In its application to target tracking and under ideal conditions, when the probabilities are known, the DS method is identical to the Bayesian method and results in a performance equal it. If the uncertainty is given a finite value, the performance will be slightly inferior to the Bayesian method. In other situations, where conditions are not ideal, the DS method is likely to be superior. In situations where targets are manoeuvring and tracks are crossing, the system must arrive at robust conclusions rapidly and be able to handle unexpected target acceleration. Similarly, the DS method should be appropriate in the initial stages of track acquisition. In such scenarios, the DS method should outperform the Bayesian approach.

In this application, the Dempster-Shafer method is used to determine the probability that a target, A, which has a site track is the same as a target, B, which has a system track. This is done using two contradictory hypotheses along with uncertainty. The hypotheses are:

1) A and B are the same target.
2) A and B are different targets.

The uncertainty may be regarded (though, strictly speaking, it is not true) as the probability that either A or B may be true. The sum of a probability of an hypothesis and the uncertainty is called the plausibility. The plausibility indicates the maximum probability that may be associated with the hypothesis being considered. The plausibilities contain the same information as the probabilities and uncertainty but reduce the number of values to examine.

The initial probabilities are defined by the user. There are primarily two ways to specify the initial probabilities, these are: the Bayes approach where the uncertainty is zero and the Dempster-Shafer approach where the uncertainty is one. In the former approach the uncertainty will always be zero which means that every piece of evidence has to provide a conclusive contribution to the fusion. The latter approach allows the uncertainty to acquire any value in the 0 to 1 range and accepts that some data may be inconclusive. The interpretation of the results will depend on the initial values, in particular the thresholds for association and track displaying will depend on the values. Trials which have been performed show little if any difference between the two approaches. The Dempster-Shafer approach of having an initial uncertainty of one will be assumed for the rest of the discussion.

With the Dempster-Shafer approach complete uncertainty is initially assumed. This means that in the absence of any information one doesn't know whether or not A and B are the same target. As time progresses each piece of information is used to update the uncertainty and the probability of each hypothesis. The update requires that the position and velocity information for A and B is used to determine probabilities for the two
hypotheses and the uncertainty. In order to do this the state vector for target B is extrapolated to the time of the state vector of A. This will be extrapolation forward in time provided the data is in chronological order. The extrapolation is performed by adding the product of the velocity of target B and the time difference between the observation of A and the last update of B to the position.

The probabilities for the hypotheses and uncertainty are then determined using two intermediate probabilities. The probability that A and B (after extrapolation) have the same position is:

$$P_x = e^{-\left(\frac{d}{\sigma_x}\right)^2}$$

where d is measured in metres and defined by

$$d = \sqrt{\left(R_e(A_{lat} - B_{lat})\right)^2 + \left(R_e \cos(A_{lat}) \left(A_{lng} - B_{lng}\right)\right)^2 + \left(A_{alt} - B_{alt}\right)^2}$$

The value of $\sigma_x$ is set for each radar site by the operator and should reflect the expected errors due to the radar.

The probability that the targets have the same heading is:

$$P_\theta = e^{-\left(\frac{\theta}{\sigma_\theta}\right)^2}$$

where theta is determined using

$$\theta = \cos^{-1}\left(\frac{v_A \cdot v_B}{|v_A| \cdot |v_B|}\right)$$

The value of $\sigma_\theta$ is set for each radar site by the operator and should reflect the expected errors due to the radar. If the target is not performing manoeuvres then the heading should not depend on the antenna positions of the radars.

The values of $P_x$ and $P_\theta$ are combined to form probabilities for the two hypotheses and the uncertainty, denoted as $P(1)$, $P(2)$ and $P(UNC)$ respectively. These two values may provide consistent information, which will give a small uncertainty, or inconsistent information, which will give a large uncertainty. Since contradictory hypotheses are being used, the combinations of the two values must give independent estimates of each hypothesis. This is done as follows:
\[ P(1) = \sqrt{P_0 P_x} \]
\[ P(2) = \sqrt{(1-P_x)(1-P_0)} \]

and

\[ P(UNC) = 1 - P(1) - P(2) \]

These probabilities are used to update the old values, denoted by OLD(). Table 3.1 shows the first stage in the application of the Dempster-Shafer rule of combination.

**TABLE 3.1**

The Dempster-Shafer Rule of Combination.

<table>
<thead>
<tr>
<th></th>
<th>P(1)</th>
<th>P(1)OLD(1)</th>
<th>K</th>
<th>P(1)OLD(UNC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(2)</td>
<td>K</td>
<td>P(2)OLD(2)</td>
<td></td>
<td>P(2)OLD(UNC)</td>
</tr>
<tr>
<td>P(UNC)</td>
<td>P(UNC)OLD(1)</td>
<td>P(UNC)OLD(2)</td>
<td></td>
<td>P(UNC)OLD(UNC)</td>
</tr>
</tbody>
</table>

Here "K" denotes inconsistent hypotheses which give no contribution. A set of masses is constructed so that MASS(1) is the sum of all components in Table 3.1 which contain 1 as an argument. The value of MASS(2) is determined in the same way as MASS(1). MASS(UNC) is the sum of entries which have only UNC for both arguments. The rules of combination are outlined by Blackman, 1986. In this instance the rules are:

\[
\text{MASS}(1) = P(1)\text{OLD}(1) + P(1)\text{OLD}(UNC) + P(UNC)\text{OLD}(1) \\
\text{MASS}(2) = P(2)\text{OLD}(2) + P(2)\text{OLD}(UNC) + P(UNC)\text{OLD}(2) \\
\text{MASS}(UNC) = P(UNC)\text{OLD}(UNC)
\]

The masses are then normalised to arrive at updated probabilities for the hypotheses and uncertainty.

In a few cases the Dempster-Shafer rule of combination is not used, these correspond to when a velocity of zero occurs and when a probability close to one is observed. Firstly, if the velocity is very small, the probabilities are arbitrarily set to:

\[
P(1) = 0.5 P_1 \\
P(2) = 0.8 - P(1) \\
P(UNC) = 0.2
\]

This has the effect of reducing the uncertainty in favour of one of the hypotheses, the particular hypothesis being determined by
the position information. This will only arise when a very small velocity is reported by a site tracker or the system tracker.

In the second case, a probability of 1 for hypothesis 1 or 2 does not allow the probabilities to change. However, the probabilities for the hypotheses must be allowed to decay. This is because the quality of association reflected by the probability values can decline such as might be observed when a track splits into two.

Therefore, when a probability greater than 0.98 occurs the values of the probabilities are arbitrarily set to:

\[
P(1) = 0.95 \quad \text{or} \quad 0.
\]
\[
P(2) = 0. \quad 0.95
\]
\[
P(\text{UNC}) = 0.05 \quad 0.05
\]

before updating. These values allow the Dempster-Shafer rule of combination to reexamine the probabilities. The values may then decay if poor associations are observed. This rule does not prevent the two targets from being matched again quickly, which might occur if the transmission of track information was interrupted. This rule causes a cyclic behaviour when a very high probability is observed over a period of time. Often the values will behave as 1.0, 0.95, 0.98, 1.0, 0.95, ...

The probability for hypothesis one has to be greater than a user defined threshold for the information to be used to update a system track. If an observation has not been used to update any system track and its uncertainty is less than another user defined threshold then a new system track is created. The probability values are not used for deleting tracks. Deletions are determined by the amount of time which has passed since the last update.

Presently there is no attempt made to merge system tracks. This is an important step which should be included in future development. The system may develop two tracks very close to one another because of some poorly associated points or because of targets which are travelling close to one another. In either case it is worth merging the two tracks for several reasons. Firstly, if multiple tracks arise because of poor association the result can be that too many system tracks develop and the system halts. This limits the number of targets which can be tracked. In one test scenario with 5 targets there were more than 9 system tracks because of duplicate tracks. The duplicate tracks slow the system tracker as well.

Secondly, when there are two or more targets in close proximity they will be essentially indistinguishable on the display. In this case the best for which one could hope is a count of how many targets are in the group. The count could reflect the number of identification numbers as well as an estimate of the number of unidentified targets. The latter could
be accomplished using a rule base which would consider the number of observations in the region during one scan at the various radar sites.

The merging of tracks will require either a loss of information or retrospective updating. The latter method is obviously to be preferred.

3.3 Kalman Filter Updating

The system tracker uses a Kalman filter for updating the system state vectors. The update is always performed in chronological order. The system cannot presently perform retrospective updating as may be required if chronological updating cannot be guaranteed. Throughout this section the state vectors are composed of the target position in latitude and longitude, the target velocity in latitude and longitude and the biases in latitude and longitude.

The Kalman filter describes a linear system using a model of the system evolution and measurements of the state of the system. The state of the system is represented by a vector, \( x \), which has \( N \) elements and the measurements are contained in the vector, \( y \), which has \( M \) elements. A matrix, \( A \), models the evolution of the system and noise associated with the model is added as a vector, \( w \). The covariance matrix of the model noise is \( Q \). An offset vector, \( b \), models time independent changes to the state vector and is required to reproduce an example in Bozic, 1979. In other formulations of the Kalman filter, including the system tracker, this vector is not present and can be set to zero. The measurements are linear combinations of the elements of the state vector, \( x \) and this is contained in the matrix, \( C \). Measurement error is added as a vector, \( v \) with covariance matrix, \( R \). The state of the system at time step, \( k \), is described by the following two equations.

\[
\begin{align*}
x(k) &= Ax(k-1) + b + w(k) \\
y(k) &= Cx(k) + v(k)
\end{align*}
\]

The Kalman filter equations combine the system state vectors with the observation vectors to give the best, least square error, estimate of the system. The process of filtering has two stages. In the first stage, the best estimate at time \( k-1 \) is extrapolated to time \( k \). This extrapolation is then combined with the measurement at time \( k \) to give the best estimate. The filter equations, in the order of calculation are
\[ P_1(k) = AP(k-1)A^T + Q(k) \]
\[ K(k) = P_1(k)C^T [CP_1(k)C^T + R(k)]^{-1} \]
\[ P(k) = P_1(k) - K(k)CP_1(k) \]
\[ \hat{x}(k) = A\hat{x}(k-1) + b + K(k) [y(k) - C(A\hat{x}(k-1) + b)] \]

The matrix, \( P_1 \), is the covariance of the predicted state vector, \( A\hat{x}(k-1) + b \). Note that the state of the system is represented by the vector, \( x(k) \), and the best estimate of this vector is, \( \hat{x}(k) \). The Kalman gain, \( K(k) \), is used to weight the error, \( y(k) - C(A\hat{x}(k-1) + b) \), obtain a correction to add to the predicted state vector and produce the new estimate of the state vector, \( \hat{x}(k) \). The covariance of the estimated state vector is \( P(k) \).

The Kalman filter requires initial values of the covariance, \( P(0) \), and estimated state vector, \( \hat{x}(0) \), which are defined in the main section of the system tracker. The system model is specified by the matrices, \( A \) and \( Q \), and the vector, \( b \). The measurement process is described by the matrices, \( C \) and \( R \).

The Kalman filter routines have been tested by comparing the output with a worked example for a falling body found on pages 130 to 133 in Bozic, 1979. The output of the program agrees with the results tabulated by Bozic.

3.4 The Effect of Biases

For a given radar there are liable to be errors in azimuth, range and, where applicable, elevation. These errors will be more or less constant or slowly varying over time. When two or more radars observe the same target or many targets and a sequence of apparent target positions are obtained, there exists the possibility of determining the bias errors and correcting them.

The problem is not trivial because the bias errors are embedded in the data in a complicated way. If a single target is observed over a short track, its position during the observation period will be uncertain because of the inherent inaccuracy of the radar associated with thermal noise and clutter. These inaccuracies, which may appear quite small, may nevertheless give rise to very large uncertainties in the bias corrections. In particular it may be impossible to determine effectively which of two radars needs the correction or whether both require correction.

If many radars observe a short track, the situation may be eased in principle because there is the possibility of cross checking the accuracies of the radars in a pairwise fashion so that errors in one radar can be identified with that radar.

One approach to the problem is based on Gaussian statistics.
Thermal noise and clutter can be expected to result in an elliptical contour of constant probability around each predicted target location from each radar. In addition to this, independent bias errors in the target azimuth and range can be assumed. Once again these can be assumed to be Gaussian. While the noise errors can be assumed to be independent and to vary from scan to scan, the bias errors can be assumed to be constant.

In general, the trajectory of the target is not known. However, most targets will be moving in a straight line for at least portions of their trajectories. Manoeuvring can be modelled as the introduction of periods of acceleration. This reflects the fact that the position of a target cannot change instantaneously from one point to another since the target must possess some inertia. It provides a built-in constraint to the target motion.

Because of the uncertainty in the true target track, it is probably not practically feasible to correct the bias in a single radar based on the observation of one or more tracks by that radar alone. If there are two radars, the two biases on each of the radars must be deduced from the data corresponding to one or more tracks. As the number of radars and common tracks increase, there is more data available and the bias estimates are likely to be more precise.

It should be possible to employ a Kalman filter to provide not only estimates of the state vector of a target but also the estimates of the radar biases. The latter can be regarded simply as components of the state vector. Once these are estimated, the radar data can be corrected or adapted so that subsequent biases are small. If there are many radars observing a target, this implies that the dimensionality of the state vector will be large and the calculations rather cumbersome. Similarly, if there are many uncorrelated tracks (but with the same biases), these will have to be processed simultaneously and again the dimensionality of both the data and measurement vectors will be large.

When the radars are located in a chain, a target will be seen by at most two radars. As mentioned above, if the target is seen by only one radar, there is no practical possibility of radar bias correction. It would appear that some information could be gleaned about biases if the target model were highly constrained, for example if the trajectory were a straight line at constant velocity. However, the target can change its velocity and execute manoeuvres at will.

The question arises as to how effective the filter will be in a situation where the bias errors are difficult to estimate because the system is inherently under-determined. As an example of this situation, the case when the target travels perpendicular to the line joining the radars can be considered. If there is an inconsistency in the range of the target, the range error can be ascribed in any proportion to either of the radars. Similarly
azimuthal biases cannot be uniquely identified with either radar. Thus the problem is under-determined.

The general under-determined problem is discussed by Menke, 1989, for the case where a model vector is to estimated from data and it is shown that an additional constraint involving the minimum model parameters can resolve ambiguities. This corresponds in some sense to the simplest solution. Thus the mixed problem of both under and over determination is solved by minimising a linear combination of the mean squared error and the norm of the model vector. If the model column vector, \( m \), and the data column vector, \( d \), are linearly related by the matrix \( G \) by:

\[
Gm = d
\]

the damped least squares solution is obtained by minimising

\[
\Phi(m) = e^T e + \epsilon^2 m^T m
\]

where \( \epsilon \) is a weighting parameter to be chosen empirically and the vector \( e \) is the error or difference between the observed and predicted data, i.e.

\[
e = d - Gm
\]

The estimated model vector can be shown to be:

\[
m = [G^T G + \epsilon^2 I]^{-1} G^T d
\]

In the Kalman filter, described by Bozic, 1979, the model and data vectors are also treated as column vectors. The Kalman gain matrix has a number of rows, \( q \), equal to that of the model vector and a number of columns, \( r \), equal to that of the data vector, where \( q \) is typically greater than \( r \). It is noteworthy that the Kalman gain has a structure which is similar to that of the last equation.

A further consideration for attempting to determine the biases is the difficulty incurred by the transformation to a common coordinate system. Typically a radar determines a range, \( r \), azimuth, \( \theta \), and elevation, \( \phi \), for each target that it detects. In order to use information from more than one radar it is necessary to transform the radars measurements to a common (earth) coordinate system. The current system tracker uses geocentric coordinates i.e. latitude, longitude and altitude. However, the difficulties will arise regardless of the common
coordinate system.

If every radar provides range, azimuth and elevation then the corresponding geocentric coordinates can be accurately determined. This allows for comparisons to be made and no special consideration of the coordinate transformation is required. It does, however, point to the need to perform tracking in altitude as well as in latitude and longitude.

If some of the radars do not provide elevation information, as in the northern watch system, then the transformation to geocentric coordinates requires an estimate of the altitude of the target. Since no measured value is available the estimate that is used will likely give rise to errors. The error will vary from one target to the next and will be nonlinear. Therefore these errors will likely affect the estimates of the biases for the radars.

In regions of overlap between two radars the opportunity to estimate the target altitude arises. This can be done because the two radar ranges determine two circles which will usually intersect in two places. One point of intersection will likely be unobservable by the radars and the other will be the target location. This method assumes that two observations arose from the same target. This allows an estimate of the altitude to be made, however, the reliability of assuming two observations arose from the same target is unclear. If the assumption was incorrect all system tracks involving the two site radars would be affected.

3.4.1 Conversion to Geocentric Coordinates

The natural coordinates for comparison of information from two radar sites are latitude, longitude and altitude. Thus a conversion from one to the other is required. A program which performs the conversions between geocentric and radar coordinates has been written and is included in Appendix 3B.

To do the conversion we imagine the earth to be a unit sphere so that only unit vectors need to be considered. Initially the vector position of the radar site, R, and the location of north for the site, N, are known. A vector pointing east is constructed using E=NXR. The vectors R, N, and E are orthonormal and the apparent position of the target, P, can be expressed as:

\[ P = \cos(\theta)N + \sin(\theta)E \]

where \( \theta \) is the azimuth angle of the target.

The angular distance between the radar site and the target denoted by \( \omega \) is the inverse tangent of the ground range divided by the radius of the earth. The target position is a linear combination of R and P. Denoting a vector to the target as T we
have:

\[ T = \cos(\omega)R + \sin(\omega)P \]

The latitude and longitude are derived from the unit vector T. In particular two of the components of T, T_1 and T_2, are in the equatorial plane and the angle between them is the longitude, namely:

\[ \text{longitude} = \tan^{-1}\left( \frac{T_2}{T_1} \right) \]

Similarly the latitude is:

\[ \text{latitude} = \tan^{-1}\left( \frac{T_3}{\sqrt{T_1^2 + T_2^2}} \right) \]

The altitude is determined using the cosine law, since the radius of the earth, the range of the target and the elevation of the target are known.

The effect of range and azimuth biases is to alter the measured range by an amount \( \delta r \) and the azimuth by \( \delta \theta \). Only terms up to second order in the increments are used. In this case:

\[ P_{\text{biased}} = \cos(\theta+\delta \theta)N + \sin(\theta+\delta \theta)E \]

The range bias causes a bias in the angle between the radar site and the target. If this bias is denoted by \( \delta \omega \) then

\[ \cos(\omega+\delta \omega) = \frac{(R+\delta r) \cos(\psi+\delta \psi)}{R+A} \]

where \( R \) is the radius of the earth, \( A \) is the altitude of the target and \( \psi \) is the target elevation with elevation bias \( \delta \psi \). The target position when the biases are included is:

\[ T_{\text{biased}} = \cos(\omega+\delta \omega)R + \sin(\omega+\delta \omega)P_{\text{biased}} \]

This can be converted into (position dependent) longitude and latitude biases.
3.5 System Track Display

It is difficult to obtain a high resolution display in VGA colour compared with monochrome, especially if many colours are required. In the present application different colours are used for the latitude and longitude lines, the radar sites and the target tracks. The tracks use two colours depending on whether the target has an associated ID number or not. The operator also has the ability to change the colour scheme. Therefore it was felt that a 256 colour option, as described below, is desirable and this has the effect of reducing the resolution.

The standard IBM resolution in the VGA graphics mode ranges from 640x480 pixels per screen with a choice of 16 colours to 320x200 pixels with 256 colours. Because of the particular requirements of the tracker display, the lower resolution is appropriate unless special hardware is employed.

Super VGA (SVGA) cards are produced by several manufacturers. Examples are ATI, Chips, Genoa, Paradise and Trident. Though there are similar trade-offs as with the standard IBM system, a much better performance is possible. One of the cards with the best performance is the Trident card which can produce a display of 1024x768 pixels with 256 colours. This uses the Intel 8900 video controller chip. However, to obtain this performance, a high quality monitor is required. The card has an auto-detect monitor feature which implies that, if the monitor is not of the correct type, it will be impossible to obtain the excellent resolution from this board. A useful reference text is Ferraro, 1990.

The normal moderately-priced colour monitor, such as the NEC MultiSync-3D can be employed with the Trident card to achieve a resolution of 640x480 pixels with 256 colours. The results with the higher resolution are more visually satisfying but there may be problems with the standard IBM screen dump (PrintScreen key) which seems to be configured for the lower 320x200 screen.

Two versions of the program have been supplied. The first is appropriate to the standard IBM resolution and the second to the 640x480 pixel screen. In the latter case, it is assumed that a card capable of the higher resolution and a compatible monitor are available for the computer.

In the future it should be possible to auto-detect the video card and the monitor and set the appropriate video mode by software. It would also be useful to try a very high quality monitor to obtain the 1024x768 display. It is noteworthy that the Trident card retails for less than $100.00.

Standard graphics drivers such as Borland's C++ and Turbo Pascal drivers allow the use of some functions of video cards. Unfortunately the supplied drivers do not take advantage of all the features of the video graphics adapter (VGA). To access the
256 colour mode requires that the mode be set using specific interrupts because C++ cannot access this feature. For this reason a new graphics driver has been written.

The 256 colour mode of the VGA is of particular interest because it allows 64 shades of each of grey, blue, red and green. Combinations of these shades give a wide variety of possible colours, however, only 256 colours may be chosen at any given time. The colours which parameters may be chosen from are:

- 0 - 63 Grey scale.
- 64 - 127 Blue scale.
- 128 - 191 Red scale.
- 192 - 255 Green scale.

For each colour the different values correspond to different intensities with higher values being brighter. Thus the values 0, 64, 128 and 192 all correspond to black.

The display of strings on a graphics screen is obtained using BIOS interrupts. This means that the character size is determined by the graphics mode. In the low resolution graphics mode the characters are larger than normal, however, in the high resolution mode the normal letter size is obtained. It should be noted that the size of characters written to the screen in any graphics or text mode can be controlled. This is clearly the case when using a standard driver (see SetUserCharSize in Turbo C++ and Turbo Pascal). The SetUserCharSize routine alters the character size by changing the character table used for video output. Altering the character size can be added to the current graphics driver if this is deemed to be important.

The present implementation of the system tracker requires a significant amount of time for displaying the tracks. Several strategies can be used to reduce the display time. As mentioned earlier, merging system tracks would significantly reduce the number of tracks to be displayed. The number of grid lines that targets penetrate also affects the display speed. Thus a small tail size and a large grid size are beneficial. The speed might also be improved by writing directly to video memory.

3.6 Applications of the Transputer

The use of parallel processing has been proposed for the system tracker. Such a system would have as much of the system tracking being performed simultaneously as possible. The improvement in processing speed is obtained at the cost of breaking the tracking problem into suitable tasks.

Currently the main aspect of the system tracker which can be performed in parallel is the association. However, this will only improve the speed significantly if there are a significant number of tracks from each radar. If there were a transputer for
each radar dedicated to performing the association of that radar's site tracks to the system tracks then the association time would be reduced. The reduction would only be significant if the association time for each radar was large. Currently this does not appear to be the case.

The use of transputers may be important if the expected scenario is going to involve a high number of targets. It may also be of value if the retrospective updating is incorporated into the system tracker. With retrospective updating the association and the updating could be performed by radar specific transputers. This has the limitation, however, that the system tracks can only be updated by one radar at one time, i.e. the various transputers may end up having to wait their turn for updating.

The speed of the system could be improved by performing the display operations at the same time as the system tracking. This could be done using specialised video hardware or by using two computers - one performing the tracking and the other displaying the results. This will be particularly important if large areas are to be displayed. For example, if the northern watch system is to be, displayed then a spherical display which accounts for the earth's curvature will be important.

3.7 Conclusions

The system tracker incorporates a Kalman filter and can track targets from up to ten radars with a maximum number of tracks of 15. The radars can be in any configuration and do not have to be chained. The performance is reasonably rapid though several improvements are desirable.

The system does not presently combine multiple system tracks which correspond to the same target. This along with the merging of tracks is desirable because it would reduce the number of system tracks. The reduction in the number of system tracks will improve the display speed.

Prior to updating system tracks the relevant pieces of data are sorted chronologically. Presently a simple "bubble" sort is used. The "bubble" sort is known to be inefficient so it would be worthwhile to improve this if long queues are anticipated. This will occur if a target can simultaneously be observed by numerous radars or the update time is significantly longer than the site scan times.

The display is relatively slow particularly when grid lines have to be refreshed. It is unclear whether or not the speed of the display can be significantly improved (without performing the display in parallel). Writing data directly to video memory, rather than calling an interrupt, may improve the speed. Storing the grid with the radar sites and then restoring them from memory
each time an update occurs may be faster than the current display method. It may be possible to store the grid and radar sites on the VGA board and use an on-board memory transfer.

The display of text and numbers is presently constrained to a relatively large character size in the low resolution graphics mode. In high resolution graphics a standard letter size is obtained. Control of the character size is possible and should be examined if text and numerical data is to be displayed regularly.

REFERENCES


CHAPTER 4

HOPFIELD NEURAL NETWORKS FOR DATA ASSOCIATION

4.1 Introduction

In this chapter, the applicability of the Hopfield neural network for optimisation problems is examined and its usefulness in the data association problem is assessed.

In general, any technique for solving optimisation problems has to be considered in terms of the quality of the solution (i.e. how close it is to the optimal one) and the speed with which the solution is obtained. Hardware neural networks could provide solutions in very short times.

After a review of the literature on the Hopfield network, the simulations performed at London Research and Development will be presented. The implication of the simulations for the data association problem in multiple radar tracking applications will be discussed.

Typically, the Hopfield network (also called Mean Field Network) does not provide solutions quickly and they are not always close to optimal. No advantage over conventional methods is apparent, especially when the problem size is small. Hardware implementations of the network may produce solutions quickly but the implications of non-optimal solutions needs to be considered.

4.2 Review of the Hopfield Network

This is a brief review of the attempts by researchers to reproduce the work of Hopfield and Tank, 1985. Some researchers tried to reproduce the work directly and others tried to apply the algorithm to other problems. Many authors were unable to reproduce the promising results quoted by Hopfield and Tank, 1985, and tried to adjust the network to obtain improved performance. The changes were always justified on an intuitive basis. The work of Aiyer et al, 1990, provided an analysis of the network and explained the intuitive corrections that other authors found improved the performance of the network. The review is organised in chronological order.

Hopfield and Tank, 1985, showed that an analog form of the discrete Hopfield neural network (Hopfield, 1982, 1984) could be used to solve optimisation problems such as the Travelling Salesman Problem (TSP). This analog network can be constructed out of simple electronic components such as operational amplifiers, resistors and capacitors. The dynamics of the network are described by a set of coupled differential equations. The activity of a neuron is determined by its input signal which depends on the activity of the other neurons, the connection
weights and the amount of current applied from external sources. The response function of a neuron is a monotonically increasing function with values between 0 and 1. The response function is also called a sigmoid function in reference to its shape when plotted.

The neurons are totally interconnected and the connection weights indicate the level of influence between the neurons. The other type of signal present in the network is the input current to a neuron. This represents one way to input external signals into the network. Each neuron also has its own bias current which creates an offset for the response function. The effect of the bias current is to maintain a minimum level of activity in the network.

The TSP for a set of N cities is described using an N x N matrix where the rows represent the different cities and the columns represent positions in the tour. A network of N^2 neurons is used to solve the N city TSP. Each neuron represents whether or not a city occupies a certain position in the tour. The connection weights between the neurons are determined by comparing the energy function of the network with an energy function that describes the optimisation problem. The energy function for the optimisation problem is designed to contain local minima at valid solutions. The optimum solution is the global minimum of the energy function.

Each city in the TSP is to be visited once and only once. This implies the constraints on the network which are that there must be one and only one fully active (i.e. output of 1) neuron in each row and each column. The final state of the network is a permutation matrix.

In the TSP, the input currents are constant and the connection weights contain all of the information (i.e. intercity distances) about any specific instance of the TSP. This means that before the network can operate, the intercity distances must be incorporated into the connection matrix. Non-zero bias currents were required to have the network settle into states with N active neurons.

For one instance of the 10-city TSP, the Hopfield and Tank network obtained valid tours in 80% of the runs and half of these were the shortest tour. Only limited results were available for a 30-city instance of the TSP. No comparison with standard algorithms was done.

These authors reported a full set of parameters for the energy function they used but failed to state some important details of the operation of the network. The network is represented by a set of differential equations which must be integrated and the integration method can be important. Typically either the Euler method or the more sophisticated Runge-Kutta method is selected. Usually, the Euler method is not appropriate for integrating
differential equations and the Runge-Kutta method is preferred. Press et al, 1988, explain the serious limitations of Euler’s method. Moreover, Hopfield and Tank do not indicate the size of the time steps used in the integration.

They also did not state whether the neurons were updated synchronously or asynchronously. In synchronous updating, every neuron is updated once at each epoch but in asynchronous updating, only one, randomly selected, neuron is updated at each epoch. Ideally, each neuron should be updated at each epoch. However, in the discrete Hopfield network, this approach can cause the network to cycle between two states rather than settle into one state and remain there. When asynchronous updating is used, the network settles into stable states.

One important feature of the Hopfield and Tank network is that the details of the problem are coded into the connection weights. For example, a network to solve the 10-city TSP has the intercity distances coded into the connection weights. If another 10-city problem, with different intercity distances were to be solved, the connection weights would need to be altered. A hardware implementation must have the ability to alter the connection weights. This is not necessarily true for other optimisation problems.

When the Hopfield and Tank network is simulated in software, the connection weights can be altered easily but the performance must be compared to standard optimisation methods. The comparison must involve the speed with which a solution is obtained. A second comparison is equally important and that is the quality of the solution. In the TSP, the quality of a solution (i.e. a valid tour) is the total length of the tour compared to the length of the shortest tour.

Wilson and Pawley, 1988, attempted to reproduce the results of Hopfield and Tank, 1985. The coordinates of the 10 cities used by Hopfield and Tank were obtained from an illustration in the original paper. They found that only 15 out of 100 runs converged to valid tours. The valid tours found were not much shorter than randomly selected tours.

They proceeded to run the network on 10 other sets of 10 randomly placed cities. On average, the number of valid tours obtained was only 8%. No indication of the tour lengths in the few valid solutions is given. These results are significantly different from those of Hopfield and Tank, 1985, so they made a few modifications to try to improve the performance of the network. These modifications are described in the following paragraphs.

Hopfield and Tank, 1985, stated that the set of parameters they used represented a good but not necessarily optimal operating point of the network. Wilson and Pawley, 1988, found that if they altered the parameters by more than 10%, the ability
of the network to converge to valid solutions decreased.

They tried altering the input bias currents to the neurons during the simulation but this only made the convergence properties worse.

Wilson and Pawley, 1988, also observed that the network had a tendency to place the same city at adjacent positions in the tour. This would incur no distance penalty but would incur the penalty associated with violating the constraint that each city should only occur once in the tour. When they included a distance penalty for the same city being at adjacent positions in the tour, the convergence to valid tours was increased only slightly.

The last important modification they tried was to include global information in the network. Each neuron was given an extra input bias depending on the coordinates of the city it represented and the position in the tour. This bias is designed to stop cities with the largest separation from being at adjacent positions in the tour. When this added bias was included in the simulations and the 10 sets of 10 randomly placed cities were tested again, the convergence to valid tours increased to 18%. The mean tour length was less than the mean tour length of randomly selected tours.

One important point to make about the work of Wilson and Pawley, 1988, is that they used the Euler method to integrate the differential equations and the time step was $10^{-5}$ times the time constant. Since they only allowed the simulation to run for 1000 iterations, this represents only 0.01 of a time constant. Even 10,000 iterations is still 0.1 of a time constant. This means that the network may not have the opportunity to overcome the initial random settings of the neurons and find a good solution.

Van den Bout and Miller, 1989, modified the Hopfield and Tank network and were able to obtain nearly optimal tours reliably. Their energy function contains only the distance term and a single term to discourage two cities from occupying the same position in the tour. The constraint which requires each city to occur only once in the tour is handled by normalising the neurons representing one city in different positions (i.e. along a row) so that the sum of their activations is one. These activations are interpreted as probabilities that the city occupies different positions in the tour.

They examined the performance of the network using 10 and 30 cities. At the optimum parameters, the mean tour length was only slightly larger than the shortest tour so the network does converge to nearly optimal tours.

In the approach outlined by Van den Bout and Miller, 1989, only two explicit parameters are used instead of the six appearing in the Hopfield and Tank, 1985, approach. One of these
parameters is the neuron temperature, $T$, which is proportional to the parameter, $u_0$, in the Hopfield and Tank network. Van den Bout and Miller, 1989, show that the two parameter values giving the optimum performance of their network lie along a line that depends on the number of cities. The performance of the network was improved slightly when the temperature was annealed through a small range near the line of optimum performance.

Cuykendall and Reese, 1989, examined the behaviour of the Hopfield and Tank, 1985, network for tours ranging from 10 cities to 165 cities. The case of 10 cities was analyzed for 10 different sets of randomly placed cities. They provide the arguments that relate the different parameters of the network and allow the network to be described by only two parameters. The arguments are based on the principle that the different constraints should have equal strength during the operation of the network. When this is applied, the network parameters can be reduced to the bias current of each neuron and its gain. The parameters for the constraints can be related to the bias current.

These authors were able to find bounds on the bias current according to either barely having $N$ neurons on or barely having $N(N-1)$ neurons off. These bias current bounds approach a maximum range as the number of cities, $N$, approaches infinity.

When the bias current bounds are used, the network performs well for the 10 city tour. The rate of converging to invalid tours is low and the minimum tour length is found frequently.

The main achievement of Cuykendall and Reese, 1989, is that they are able to show how the network parameters can be bounded for a problem involving any number of cities.

One of the first works to compare the performance of the analog Hopfield network to other methods is that of Sengupta and Ilitis, 1989, who applied the Hopfield network to the association problem in multiple target tracking (MTT). The assignment problem is set up as a matrix in a fashion similar to the TSP. The different columns represent different measurements from the same scan and the rows represent the different tracks. They compared the performance of the Hopfield network to that of the Joint Probabilistic Data Association (JPDA) technique.

The network constructed for data association uses the association likelihoods in the input currents only. The association likelihood for a detected target compared to a track prediction is defined as the probability of observing the error between the measurement and the track prediction. The connection weights in the neural network depend only on the constraints. This is an important difference from the TSP network where the intercity distances are included in the connection weights. Thus, data association can be achieved using a network with fixed weights and with the information specific to a set of
measurements conveyed in the input currents.

An important aspect of the JPDA technique, is that each track is associated with a weighted average of the measurements. This is incorrect since only one measurement can arise from one target. The work of Silven, 1992, (see below) addresses this difficulty.

According to Sengupta and Ittis, 1989, the Hopfield network works well for problems involving up to 6 targets and 6 tracks. This is represented by a 6x6 matrix and would be comparable to the 6-city TSP. They report no difficulties in implementing the Hopfield network with the Euler method of integrating the differential equations. They conclude that the performance of the Hopfield network for 6 targets and 6 tracks is comparable to the performance of the JPDA technique.

A thorough analysis of both discrete and analog Hopfield networks has been performed by Aiyer, et al., 1990. They have analyzed the networks in terms of the eigenvalues of the connection matrix. This analysis is very helpful in understanding the behaviour of Hopfield networks.

They show that the connection matrix as obtained by Hopfield and Tank, 1985, has three distinct eigenvalues. The eigenvectors associated with each eigenvalue represent a space of states of the network.

The first eigenvalue has as an eigenvector the state where all the neurons are equally active. This means that the activity of each neuron is 1/N. This is close to the usual initial state where the neurons are equally active with a small amount of noise. The second eigenvalue is associated with the states that represent valid tours in the TSP plus a component in the direction of the first eigenvector. The third eigenvalue has as eigenvectors all the remaining states and they are all invalid solutions to the TSP.

The performance of the Hopfield and Tank network will be optimised if the space of valid solutions can be associated with one eigenvalue. Aiyer et al., 1990, show how to alter the connection weights to have the space of valid solutions associated with one eigenvalue. The changes to the connection matrix include: self inhibition which tends to force a neuron to turn off and a global stimulus to help neurons turn on. Their work also explained why non-zero bias currents were required to stabilise the network. When they applied these findings to the connection matrix for the TSP, they always obtained valid solutions with tour lengths comparable to those produced by the nearest neighbour algorithm.

The work of Aiyer et al., 1990, seems to be the best analysis of the Hopfield and Tank network to date and shows that the parameters used by Hopfield and Tank, 1985, were close to the
best ones.

Kamgar-Parsi and Kamgar-Parsi, 1990, examined the relative effects of the parameters in the Hopfield and Tank network and then tested the network on the TSP and a clustering problem. They state that Hopfield and Tank, 1985, used an adaptive step size method to integrate the differential equations. Kamgar-Parsi and Kamgar-Parsi, 1990, feel that such a method can step over local minima in the solution space and this would not accurately simulate the performance of a hardware implementation.

These authors obtained valid tours at the same poor rate as Wilson and Pawley, 1988, although the valid solutions found are much shorter than those found by Wilson and Pawley, 1988. By changing the parameters and the termination conditions, Kamgar-Parsi and Kamgar-Parsi, 1990, increased the rate of valid solutions to 33%. A further increase to 50% was obtained when the constraint for the total number of active neurons was applied to the rows and columns separately. With these modifications, the quality of the solutions is decreased slightly since the weight of the distance penalty is decreased.

The clustering problem that they examined is the problem of dividing N patterns into K clusters. Patterns are represented as points in a multidimensional feature space. The patterns in a cluster must be close to each other and not close to the other patterns. The comparison is often made using the size of a cluster and its distance from the other clusters. Bow, 1984, describes a conventional method of separating data into clusters where the number of clusters is not known ahead of time.

The Hopfield and Tank network was applied to a sample where there were 128 points divided into 5 well defined clusters in the plane. The optimum clustering was found in all of 128 trials compared to a conventional technique which only found the optimum clustering 46 times. As the clusters were smeared in the plane, the neural network solutions approached the solutions found using the conventional technique.

Lee and Sheu, 1991, analyzed the Hopfield network applied to an analogue-to-digital converter (ADC). When the formulation is done according to Hopfield and Tank, the network has many local minima that represent poor solutions. These solutions make certain digital codes stable even though they are not the correct codes for the applied input voltage. The local minima were smoothed away by adding an extra self stimulation to the neurons. This is similar to the self inhibition Aiyer et al, 1990, added to the TSP network.

Kunz, 1991, applied the Hopfield and Tank network to the channel selection problem in planning cellular mobile radio networks. This problem is described as assigning a number of channels to individual communication base stations with the constraint that two channels cannot be assigned at different base
stations if their frequencies are too close together. The performance of the network was disappointing in that sub-optimal solutions occurred frequently. In a simple numerical example, he shows that even a strong constraint encoded in the network can be violated. The network avoids the optimum solution and this is attributed to higher order (i.e. non-linear) coupling of the connections.

These higher order couplings might explain why the Hopfield and Tank network does not perform well. In the neural network solution to the TSP, the constraints are encoded separately. When they are combined into the final energy function, higher order coupling could be affecting the dynamics of the network.

Bizzarri, 1991, is another author who has attempted to reproduce the results of Hopfield and Tank, 1985. She did not get very good performance of the network and attempted to improve the performance by introducing an extra stimulus which depends only on the activation of the neuron being updated. The self forcing was done instead of adding a constant bias current to each neuron.

This self forcing can be compared to the work of Aiyer et al, 1990 who used non-zero bias currents and self inhibition to obtain good performance of the network. Bizzarri, 1991, had no bias currents but required self stimulation in order to obtain states with N neurons active. Since she had no offset current to stimulate the network, self stimulation helped avoid having too many neurons turn off.

She obtained valid tours with frequencies between 72% and 88%. Nearly optimal tours were found regularly.

Silven, 1992, has applied the Hopfield network to the problem of associating tracks and targets in multiple target tracking. This work follows on the work of Senguta and Ilitis, 1989. In his work, only one measurement can be associated with one track. The possibility of unmatched measurements and tracks are included in his work as an extra row and an extra column in the association matrix. This is different from the JPDA approach where all the measurements are used for each track but with varying weights.

He gives results for situations involving three targets and three tracks. The association matrix is 4x4 because of the extra row and column for unmatched targets and tracks. This small problem is easily solved by the Hopfield network.

In the work of Silven, 1991, the connection weights do change depending on the data used. The connection weights depend on the constraints but also have a term that depends on the number of measurements and the number of tracks. The information on the association likelihoods is contained in the input currents.

Shrivastava et al, 1992, have considered the use of the
discrete Hopfield network for solving the vertex cover problem in graph theory. They were mainly concerned with studying the difficulty of the synchronous update method to converge to stable rather than cyclic states. In their work, a resetting feature into the synchronous update method is introduced. When the network enters a 2-cycle (i.e. the network oscillates between two states), the elements of the state vector can be divided into two classes. The first class is the set of elements that do not change at each iteration. The second set is made up of the elements that do oscillate. Only elements from the second set are altered in the reset scheme. The result is that the network will converge to a stable state in at most \(3N/2\) iterations, where \(N\) is the number of neurons. The implications of this work to the analog Hopfield network are not yet clear. The difficulties of the analog Hopfield network are not restricted to the existence of cyclic states. Even when the analog Hopfield network converges to a stable state, this final state is not necessarily a good quality (i.e. nearly optimal) solution.

Zhou and Bose, 1993, have commented on the work of Sengupta and Ittis, 1991. They conclude that the energy function of the previous authors is inappropriate since it can produce solutions that are not acceptable in JPDA. Also, they show that the time complexity of the software version of the network is greater than that of JPDA.

4.3 Results of Simulations

In this section, we summarize the results obtained in our attempts to reproduce the work of Hopfield and Tank, 1985. Initially, the differential equations were integrated using the Euler method and asynchronous updating. The performance of the network was not satisfactory even when the 5-city TSP was studied. Parameter sets that often resulted in valid tours did not strongly favour the nearly optimal tours. The performance on the 10-city TSP was poor since it was difficult to find a set of parameters that frequently gave valid tours.

The Runge-Kutta method of integrating the differential equations was adopted along with synchronous updating. This gave good performance on the 5-city TSP. Valid tours were frequently obtained and the optimum tour was found most often. Again, the performance on the 10-city TSP was poor because valid tours were difficult to obtain.

In order to obtain the gradual settling into a final state as described by Hopfield and Tank, 1985, their parameters could not be used. When their parameters were used, the network did not gradually shift between states, instead it tended to have significant jumps in state space. For example, if at one time step, a neuron had an activity of 0.8, the activity at the next time step could drop to 0.5 or rise to 1.0. This is not the type of behaviour described by Hopfield and Tank, 1985. In order to
get changes of only a few percent of the neuron activity, we used the following parameters defined by Hopfield and Tank:

\[ A=50.0 \quad B=50.0 \quad C=20.0 \quad D=50.0 \quad u_0=0.2 \quad \tau=1.0 \quad N=15 \]

The time step was 0.00001 for the 5 city TSP and 0.00003 for the 10 city TSP. The larger time step for the 10 city TSP was required to give the network time to settle into a state close to a valid solution. The maximum number of time steps allowed was 1000 which implies a total time of less than the neuron time constant, \( \tau \).

The Hopfield network was run 70 times using 5 cities from Hopfield and Tank, 1985. The coordinates given by Wilson and Pawley, 1988, were used to determine the distances. In the 5 city TSP, there are 12 unique tours. In the 70 runs, valid tours were not found only 3 times. This means that 67 valid tours were found. The tour lengths and the frequency each occurred are given in Table 4.1 below. Note that even the longest tour (i.e. worst solution) has a significant probability of occurrence. This is not very good performance.

### Table 4.1

Performance of Hopfield Network on 5 City TSP.

<table>
<thead>
<tr>
<th>Tour Length</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2406</td>
<td>13</td>
</tr>
<tr>
<td>2.3848</td>
<td>9</td>
</tr>
<tr>
<td>2.6178</td>
<td>7</td>
</tr>
<tr>
<td>2.6220</td>
<td>4</td>
</tr>
<tr>
<td>2.6567</td>
<td>6</td>
</tr>
<tr>
<td>2.7273</td>
<td>5</td>
</tr>
<tr>
<td>2.8049</td>
<td>3</td>
</tr>
<tr>
<td>2.8755</td>
<td>5</td>
</tr>
<tr>
<td>2.9102</td>
<td>2</td>
</tr>
<tr>
<td>2.9144</td>
<td>5</td>
</tr>
<tr>
<td>3.1474</td>
<td>4</td>
</tr>
<tr>
<td>3.2916</td>
<td>4</td>
</tr>
</tbody>
</table>

When the 10 city TSP was tested, the performance was even worse than for the 5 city TSP. A total of 30 runs were done for the 10 city TSP. The city coordinates were taken from Wilson and Pawley, 1988. Valid tours were only found in 2 of the runs and the lengths were not very good. The tour lengths were 3.3639 and 3.2760. The shortest tour has a length of 2.6908.

When the work of Aiyer et al, 1990, is incorporated into the neural network, much better performance has been achieved. The Turbo Pascal program that uses the Aiyer form of the connection matrix is given in Appendix 4A. The program reads in the
intercity distances from the file "CITY.DAT" and the parameter values from the file "AIYER.DAT". The program runs until the user presses a key on the keyboard at which time the program finishes the current run and then halts. The results of each run are written to the file "JUNK.DAT". Each line in "JUNK.DAT" shows the tour length and the number of iterations required to reach convergence. If the network did not converge, the message "Not a valid tour" is written to the file.

The performance on the 5 city TSP has been tested using a total of 170 runs. Only 17 runs did not converge to valid solutions. The performance for the remaining 153 runs is shown in Table 4.2. Note that the parameters used were the same as those of Aiyer et al, 1990, except that the upper limit on the number of time steps was 1000 instead of 3000. This would explain why, in our runs, the network did not always converge to valid solutions.

TABLE 4.2
Performance of the Aiyer Network on the 5 City TSP.

<table>
<thead>
<tr>
<th>Tour Length</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2406</td>
<td>111</td>
</tr>
<tr>
<td>2.3848</td>
<td>29</td>
</tr>
<tr>
<td>2.6178</td>
<td>4</td>
</tr>
<tr>
<td>2.6220</td>
<td>8</td>
</tr>
<tr>
<td>2.6567</td>
<td>0</td>
</tr>
<tr>
<td>2.7273</td>
<td>1</td>
</tr>
<tr>
<td>2.8049</td>
<td>0</td>
</tr>
<tr>
<td>2.8755</td>
<td>0</td>
</tr>
<tr>
<td>2.9102</td>
<td>0</td>
</tr>
<tr>
<td>2.9144</td>
<td>0</td>
</tr>
<tr>
<td>3.1474</td>
<td>0</td>
</tr>
<tr>
<td>3.2916</td>
<td>0</td>
</tr>
</tbody>
</table>

The two shortest tours were selected in 91% of the runs. The 6 longest tours were never selected. This is good performance for the 5 city TSP.

Very good performance was obtained on the 10 city TSP. A total of 88 runs were done for the 10 city TSP using the city coordinates given by Wilson and Pawley, 1988. The network failed to converge to valid solutions in only 6 runs and selected the shortest tour 81 times. A tour of length 2.7784 was selected once. This is still quite close to the shortest tour.

The results of running the neural networks can be compared to the performance of a simple nearest neighbour approach. The algorithm can be described by three steps. Begin by selecting the first city of the tour at random. Select the next city out of the ones not yet selected, as the closest one to the previous
city. Finally, continue to select the cities according to the second step until the tour is completed. If all 10 cities are used as the first city in the tour, a set of 10 tours is obtained. These tours can be compared to the tours selected by the neural networks.

The nearest neighbour algorithm described above was applied to the 10 cities used in the tests. Only 7 unique tours were found. The tour lengths and frequency of occurrence are given in Table 4.3. Also, it should be noted that, the total number of possible tours is 181,440 and the overall mean tour length is approximately 4.5. Thus, the mean length of randomly selected tours is 4.5.

**TABLE 4.3**

Performance of Nearest Neighbour Algorithm on the 10 city TSP.

<table>
<thead>
<tr>
<th>Tour Length</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6908</td>
<td>1</td>
</tr>
<tr>
<td>2.7744</td>
<td>1</td>
</tr>
<tr>
<td>2.7784</td>
<td>2</td>
</tr>
<tr>
<td>2.8364</td>
<td>2</td>
</tr>
<tr>
<td>2.8974</td>
<td>2</td>
</tr>
<tr>
<td>2.9789</td>
<td>1</td>
</tr>
<tr>
<td>3.4098</td>
<td>1</td>
</tr>
</tbody>
</table>

The performance of Aiyer's network on the 10 city TSP was confirmed by running the network using three other sets of 10 randomly placed cities. The frequency of selecting tours in certain intervals is shown for the second set of 10 cities in Table 4.4. In a total of 40 runs, the Aiyer network did not find valid solutions 7 times. The shortest tour found by the Aiyer network had a length of 2.3545 and the shortest tour found by the nearest neighbour algorithm had a length of 2.4425.

**TABLE 4.4**

Performance Aiyer Network Compared to Nearest Neighbour Algorithm Second Set of 10 Cities.

<table>
<thead>
<tr>
<th>Tour Length Interval</th>
<th>Aiyer Network</th>
<th>Nearest Neighbour</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.35 - 2.45</td>
<td>27</td>
<td>3</td>
</tr>
<tr>
<td>2.45 - 2.55</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2.55 - 2.65</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2.65 - 2.75</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2.75 - 2.85</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2.85 - 2.95</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>33</strong></td>
<td><strong>10</strong></td>
</tr>
</tbody>
</table>
Table 4.5 shows the results for the third set of 10 cities. The Aiyer network was run a total of 34 times and did not converge to valid solutions 4 times. The shortest tour found by both algorithms had a length of 2.4697.

<table>
<thead>
<tr>
<th>Tour Length Interval</th>
<th>Aiyer Network</th>
<th>Nearest Neighbour</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.45 - 2.55</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2.55 - 2.65</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>2.65 - 2.75</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2.75 - 2.85</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2.85 - 2.95</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2.95 - 3.05</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>30</strong></td>
<td><strong>10</strong></td>
</tr>
</tbody>
</table>

Finally, the performance of the two algorithms on a fourth set of 10 cities is shown in Table 4.6. The Aiyer network was run 33 times and did not find valid solutions 3 times. The shortest tour found by both algorithms had a length of 2.0348. The longest tour found by the Aiyer network had a length of 2.0427.

<table>
<thead>
<tr>
<th>Tour Length Interval</th>
<th>Aiyer Network</th>
<th>Nearest Neighbour</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00 - 2.05</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>2.05 - 2.15</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2.15 - 2.25</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2.25 - 2.35</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>30</strong></td>
<td><strong>10</strong></td>
</tr>
</tbody>
</table>

4.4 Applications of the Transputer

Transputers allow inherently parallel algorithms to be divided among a set of processors. The most effective way to utilise transputers is to identify the components of an algorithm that can be processed independently. In the neural network for optimisation, the neuron updates can be performed independently
but the new states must be passed to all the processors before the next update.

In general, the speed advantage of neural networks will only be fully realised when dedicated neural hardware is used. Any software implementation is only an approximation to the real capabilities of the network. While a set of transputers running neural network software will be faster than a single processor, similar advantages could be possible for standard algorithms.

REFERENCES


D. Sengupta and R.A. Iltis. "Neural Solution to the Multitarget


CHAPTER 5
CONCLUSIONS

A system for studying multi-radar tracking problems has been developed. The system uses the DREO simulator to produce radar data, track-while-scan site trackers and a probabilistic data association system tracker. The system can presently examine 10 radars and 10 tracks. The results from operating the system are displayed on the screen with the user having full control of the display parameters. Typically a grid and radar sites will be displayed and the targets will appear as squares with a tail indicating the target velocity. The system can be set to colour targets which have identification numbers a different colour than those which do not.

The site trackers use an $\alpha$-$\beta$ algorithm to estimate target positions. They also use identification numbers for association of targets when this information is available. When identification numbers are not available the association is performed using a pair of gates. Because the site trackers use the track while scan method, they can suffer a loss of information at the end of a scan because of target motion during the scan.

Several improvements are possible for the site trackers. The $\alpha$-$\beta$ algorithm could be replaced with a Kalman filter which might improve performance when targets perform manœuvreurs. The end of scan losses could be eliminated and the overall tracking performance improved by using information from the system tracker. It is feasible to apply the current system tracker to the problem of site tracking. This would require that the problem of merging be addressed.

The system tracker has been tested with several scenarios and performed well. Comparisons of Bayesian and Dempster-Shafer methods of association have shown no visible differences. It is recognised that, due to the probabilistic data association, more than one system track can arise for each target. This requires that track merging using retrospective updating be incorporated into the tracker. The use of system track information by the site trackers may lead to improvements of the site trackers in noisy environments and this may lead to improvements in the system tracker.

An initial attempt at incorporating bias correction into the system tracker has been made. Further work is required because tracking in altitude has not been incorporated and is clearly necessary for bias correction. Since some of the radars that might be used in the system tracker do not supply height information, special consideration of the errors they will introduce is required. Also methods for deducing the altitude of targets observed by more than one radar should be examined because they will be required to improve bias estimations.
The slowest section of the system tracker is the display section. Some improvements in the display speed are obtainable by software improvements but depending on the expected scenarios these improvements may not be sufficient. If numerous targets are anticipated at any time then a separate "display computer" or specialised hardware will be appropriate. The display also suffers from errors because the earth is curved and the display is not. If large regions, with significant curvature, are to be displayed then a specialised display (say a curved display) may be an important consideration.

A neural network which finds short solutions to the travelling salesman problem has been found. This suggests that a neural network solution to the association component of the multi-radar tracking problem is feasible. The neural network application to the assignment problem requires that the implications of sub-optimal solutions to the assignment problem be examined. A further drawback to the neural network approach may be the amount of time required for communications with the tracker.

The neural network simulation is slower than traditional methods which means that dedicated neural hardware would have to be used, therefore examination of hardware will also be important. The latter can be achieved using a standard computer interface board (such as London Research and Development's I024 board) and neural network integrated circuits which are available. The development of a PC driven neural network board would allow a thorough examination of all aspects of the current and future applications.

Several possible applications of transputers are recognised but the benefits are unclear. The simulation and site trackers could be executed in parallel but this has no bearing on the multi-radar tracking performance. The system tracker association (and possibly retrospective updating) could be performed in parallel but this only has a clear benefit if many targets are anticipated at each radar. The neural network simulation software probably would not be fast enough on a transputer and specialised hardware would be required.
APPENDIX 2A

Program Listing for the simulator
control program, SIMTRAKC.C.

/* Control program for multi-radar system tracker simulation.
   Developed at London Research and Development Corporation.
   Last updated March, 1993.*/
#include <stdio.h>
#include <process.h>
#include <dos.h>
#include <dir.h>
#include <string.h>
#include <stdlib.h>

struct DOSErrOr info;

int main()
{
    char *qq, ans, fname[81], q[81], a;
    int i;
    FILE *fparm;

    qq=searchpath("simtr1.exe");
    if (qq==NULL)
    {
        printf("SIMTR1.EXE not found.\n");
        exit(1);
    }
    qq=searchpath("simtrc2.exe");
    if (qq==NULL)
    {
        printf("SIMTRC2.EXE not found.\n");
        exit(1);
    }
    qq=searchpath("simtrc3.exe");
    if (qq==NULL)
    {
        printf("SIMTRC3.EXE not found.\n");
        exit(1);
    }
    qq=searchpath("simtrc4.exe");
    if (qq==NULL)
    {
        printf("SIMTRC4.EXE not found.\n");
        exit(1);
    }
    qq=searchpath("simtrakd.dat");
    if (qq==NULL)
    {
        printf("SIMTRAKD.DAT not found.\n");
        exit(1);
    }
fparrm=fopen("simtrakd.dat","rt");
fgets(q,81,fparm);                        /* skip first line */
i=0;
do
{
    *qq=fgetc(fparm);
    if (*qq != ' ')
    {
      fname[i]=*qq;
      i++;
    }
}
while (*qq != ' ');
fname[i]='/0';
printf("Flight plan file name: ");
puts(fname);
qq=searchpath(fname);
if (qq == NULL)
{
    printf("File not found.\n");
    exit(1);
    fclose(fparm);
}
fgets(q,81,fparm);                        /* skip to next line */
i=0;
do
{
    *qq=fgetc(fparm);
    if (*qq != ' ')
    {
      fname[i]=*qq;
      i++;
    }
}
while (*qq != ' ');
fname[i]='/0';
printf("Radar file name: ");
puts(fname);
qq=searchpath(fname);
if (qq == NULL)
{
    printf("File not found.\n");
    exit(1);
    fclose(fparm);
}
fclose(fparm);

printf("Executing SIMTR1.\n");
i=spawnl(P_WAIT,"SIMTR1.EXE",NULL);
if (i != 0)
{
    printf("Error in call to SIMTR1.\n");
    exit(1);
i=spawnl(P_WAIT,"SIMTRC2.EXE",NULL);
if (i != 0)
{
    printf("Error in call to SIMTRC2.\n");
    exit(1);
}

printf("Executing SIMTRC3.\n");
i=spawnl(P_WAIT,"SIMTRC3.EXE",NULL);
if (i != 0)
{
    printf("Error in call to SIMTRC3.\n");
    exit(1);
}

printf("\nPress any key to initiate display.\n");
a=getch();

printf("Executing SIMTRC4.\n");
i=spawnl(P_WAIT,"SIMTRC4.EXE",NULL);
if (i != 0)
{
    printf("Error in call to SIMTRC4.\n");
    exit(1);
}

return 0;
APPENDIX 2B

Program Listing for DREO radar simulator, SIMTR1.FOR.

C PROGRAM TO GENERATE TARGET/CLUTTER FLIGHT SCENARIOS AND RADAR
C FALSE ALARMS. THE DETECTION OF THESE TARGETS BY 'N' RADARS,
C IN RANGE, AZIMUTH, ELEVATION, DOPPLER, VELOCITY & TIME ARE THEN
C REPORTED TO A FILE.
C THIS FILE MUST THEN BE SORTED IN CHRONOLOGICAL ORDER.
C THE ANTENNA BEAM SHAPE FOR ALL RADARS IS ASSUMED TO BE GAUSIAN
C SET COMMANDS AS FOLLOWS:
C SET 2 TO RADAR DATA INPUT FILE
C SET 1 TO TARGET AND CLUTTER FLIGHT PLAN INPUT FILE
C SET 4 UC FOR RADAR AND FLIGHT PLAN HEADER OUTPUT
C SET 3 TO REPORT FILE
C SET 105 UC FOR INPUT IDENTIFIER
C FOR IDENTIFIER = 0, MEASUREMENT ERRORS ARE NOT ADDED
C IN ADDITION, CLUTTER DETECTIONS &
C FALSE ALARMS ARE NOT REPORTED
C FOR IDENTIFIER.NE.0, ALL OF THE ABOVE ARE INCLUDED
C IN THE REPORT FILE
C
C REAL LNGD,LNGM,LNGS, LATD, LATM, LATS, MAXT(10)
REAL MAXT(10)
INTEGER TARGET,RADARS,TARGETS, LABEL(20), PLAN/1/, HISTORY/3/
,, PATCHES,RADPOS/2/, HEADER/4/, PARM/6/
INTEGER*4 SEED
CHARACTER*20 PARMFN,PLANFN,RADF, HEADERFN,HISTORFN
REAL*8 RLNG(10), RLAT(10), RE, DR, RD, PI,
TWOPI, MAXTIM, MINTIM
DIMENSION RHGT(10), RLM(10), PFA(10), RBIAS(10),
,, EBIAS(10), ABIAS(10), ENOIS(10),
,, ELIM(10), THQ(10), THETA0(10), SNR1(10)
,, WVL(10), TOW(10), BWA(10), BWE(10), XNHT(10), PRF(10), NPROC(10)
COMMON / RANDOM/ SEED
COMMON PI,TWOPI, DR, RD
COMMON / DML/ DEVER(10), J, PERIOD(10), DELAY(10)
C DATA SEED/1220703125/ !*** CHANGED BY JT
C DATA SEED/1234/
C INITIALISE SOME PARAMETERS
C SET SEED FOR RANDOM NUMBER GENERATOR
CALL GETTIM( IHR, IMIN, ISEC, I100TH)
IBSEED=131*1100TH
SEED=INT4(IBSEED)
PI = 3.14159265358979DO
TWOPI=2.0D0*PI
DR=TWOPI/360.D0
RD = 360.D0 / TWOPI
C READ IN IDENTIFIER FOR PRODUCING PERFECT RESULTS
C OR FOR ADDING MEASUREMENT ERRORS
C IDENT=0 FOR PERFECT RESULTS
WRITE(*,110)
110 FORMAT(' RADAR SIMULATION PROGRAM')
WRITE(*,82) SEED

82  FORMAT(' SEED FOR RANDOM NUMBER GENERATOR IS ',I6)
C  WRITE(*,100)
100  FORMAT(' ENTER IDENTIFIER [0 = NO CLUTTER / NON ZERO =', + ' CLUTTER] X : ',$,)
     PARMFN='SIMTRAKD.DAT' !PARAMETER FILE
C READ PARAMETERS OUT OF FILE
OPEN(PARM,FILE=PARMFN)
READ(PARM,* ) IDENT
READ(PARM,10001) PLANFN
10001  FORMAT(A20)
    READ(PARM,10001) RADFN
    READ(PARM,10001) HEADERFN
    READ(PARM,10001) HISTORFN
    CLOSE(PARM)
    OPEN(HISTORY,FILE=HISTORFN)
    OPEN(PLAN,FILE=PLANFN)
    READ(PLAN,1) LABEL,TEMPTAR,TEMPPAT
1  FORMAT(/// 20A4/ 30X,F10.0/ 30X,F10.0,/////////)
    TARGETS = INT(TEMPTAR)
    Patches = INT(TEMPPAT)
C C READ IN RADAR DATA AND COMPUTE RLIM(J),THQ(J),XNHT(J),SNR1(J)
C WHERE:
    RLIM(J)=MAX. RADAR RANGE OF DETECTION FOR A SPECIFIED
    PROB. OF DET. AND PROB. OF FALSE ALARM
    FOR THE JTH RADAR
    THQ(J)=1/E-POWER BEAMWIDTH(DEG) (TWO-WAY) FOR THE JTH
    RADAR
    XNHT(J)=NUMBER OF HITS ON TARGET FOR THE JTH RADAR
    COMPUTED FROM THE 1/E-BEAMWIDTH
    SNR1(J)=SIGNAL-TO-NOISE RATIO PER PULSE FOR THE SPECIFIED
    PROB. OF DET. AND PROB. OF FALSE ALARM FOR THE
    JTH RADAR
    CALL INIT(RADPOS,RADARS,RLNG,RLAT,
    , RHGT,RE,PERIOD,RLIM,ELIM,PF,ABIAS,ABIAS,
    , EBIAS,ENOIS,THETAO,THQ,SNR1,WVL,TOW,BWA,BWE,MAXT,
    , XNHT,PRF,NPROC,RADFN)
C
    OPEN(HEADER,FILE=HEADERFN)
    WRITE(HEADER,99) LABEL,RADARS,(INT(PERIOD(I)),I=1,RADARS)
99  FORMAT(///1X,20A4/' NUMBER OF RADARS',14X,I3,
    + ' PERIODS ',10I3)
    WRITE(HEADER,98) TARGETS,PATCHES
98  FORMAT(' NUMBER OF TARGETS',13X,I3/
    + ' NUMBER OF PATCHES',13X,I3//)
    DO 10 J=1,RADARS
C COMPUTE THE MINIMUM RADAR ELEVATION ANGLE/RADAR(ELMIN(J))
    ELMIN=-90.+ASIN(1./(1.+(RHGT(J)/RE)))*57.29578
    MAXTIM = -999999.
    MINTIM = 99999.
C COMPUTE THE RADAR ROTATION RATE(DEG/SEC)
    OMEGA=360.DO/PERIOD(J)
    IF( TARGETS .LE. 0 ) GO TO 4
C COMPUTE TIME HISTORY OF REPORTS BY THE RADAR FOR TARGETS
WRITE(*,120) J
120 FORMAT( 'WRITING TARGET DATA FOR RADAR',I3)
DO 2 TARGET=1, TARGETS
2 CALL PRODUCE(PLAN,HISTORY,RLNG(J),RLAT(J),RHGT(J),RE
, J,OMEGA,THETAO(J),RLIM(J),ELIM(J),
, RBIAS(J),ABIAS(J),EBIAS(J),
, ENOIS(J),MAXTIM,MINTIM,THQ(J),SNR1(J),
, WVL(J),TOW(J),BWA(J),BWE(J),PFA(J),ELMIN,
, XNHT(J),PRF(J),IDENT,NPROC(J))
READ(PLAN,9) IDUM
9 FORMAT(A2)
4 CONTINUE

IF(IDENT.EQ.0)GO TO 5
IF( PATCHES .LE. 0 ) GO TO 5
C COMPUTE TIME HISTORY OF REPORTS BY THE RADAR FOR CLUTTER
C PATCHES
C ADVANCE 10 LINES IN FLIGHT PLAN FILE TO SKIP OVER HEADINGS
C FOR CLUTTER DATA(SKIP 9 LINES AND READ DUMMY LINE)
C
WRITE(*,130) J
130 FORMAT('WRITING CLUTTER DATA FOR RADAR',I3)
READ(PLAN,7)
7 FORMAT(//////////) DO 3 I=1,PATCHES
3 CALL CLUTTER(PLAN,HISTORY,RLNG(J),RLAT(J),RHGT(J),
, RE,PERIOD(J),J,OMEGA,THETAO(J),
, SNR1(J),WVL(J),RLIM(J),PFA(J),TOW(J),
, THQ(J),BWA(J),BWE(J),ELMIN,ELIM(J),
, XNHT(J),PRF(J),NPROC(J))
5 CONTINUE
REWIND(PLAN)
READ(PLAN,1) LABEL,TEMPTAR,TEMPPAT
TARGETS = INT(TEMPTAR)
PATCHES = INT(TEMPPAT)
IF(IDENT.EQ.0)GO TO 10
C COMPUTE A TIME HISTORY OF REPORTS OF FALSE ALARMS OF RADAR
WRITE(*,140) J
140 FORMAT('WRITING FALSE ALARM DATA FOR RADAR',I3)
CALL FALSE(PFA(J),RLIM(J),PERIOD(J),THETAO(J),J,
, MAXT(J),HISTORY,TOW(J),PRF(J),WVL(J),NPROC(J),
, XNHT(J),BWA(J),BWE(J))
10 CONTINUE
END

SUBROUTINE CLOCK(TIME,TLAT,TLNG,TALT,TSPD,THED,TIME1,
X TLAT1,TLNG1,TALT1,TSPD1,THED1,RANGE,
X AZIMTH,BLVATN,CLIMP,ACCEL,TURNG,RLAT,
X RLAN,RHGT,RE,THETAO,OMEGA,V1,V2,V3,
* RR1,RR2,RR3)
COMMON PI, TWOPI, DR, RD
COMMON / DML / DEVPER(10), RADAR, PERIOD(10), DELAY(10)
REAL*8 PI, TWOPI, DR, RD, TIME, TLAT, TLNG, TIME1, TLAT1, TLLNG1, RLAT, RLNG, RE, TTIME, TTTLAT, TTTLNG, STSCAN, RGL1, RGL2, RGL3, TTIME1, STEP, TEMP, V1, V2, V3
*, RRL1, RRL2, RRL3
INTEGER RADAR

ON ENTRY, THE TARGET IS AT POSITION TLAT, TLNG, TALT, TSPD, THED
AT TIME "TIME". THIS ROUTINE ADVANCES THE RADAR BEAM TO THE NEXT
INTERSECTION OF THE BEAM WITH THE TARGET. THE TIME OF THIS C
INTERSECTION IS RETURNED IN "TIME1" AND THE TARGET IS THEN AT
TLAT1, TLLNG1, TALT1, TSPD1, THED1. THE RANGE, AZIMUTH, AND
ANTENNA ELEVATION ARE ALSO RETURNED.

NOTE: "TIME1" WILL ALWAYS BE GREATER THAN "TIME"

******************************************************************************

QUANTIZE INPUT TIME TO TIME TAKEN TO ROTATE 360/32768 DEGREES

TEMP = 360.D0 / ( 32768.D0 * 120.D0 )
TTIME = DINT( TIME / TEMP + 1.0 ) * TEMP

MOVE TARGET POSITION TO TEMPORARY VARIABLES

TTLAT = TLAT
TTLNG = TLNG
TTALT = TALT
TTSPD = TSPD
TTHED = THED

FIND TIME WHEN CURRENT SCAN STARTED

STSCAN = DINT( TTIME / PERIOD(RADAR) ) * PERIOD(RADAR)

CALCULATE THE BEAM POSITION AT "TTIME"

CALL TIMEAZ( BMAZ0, TTIME, OMEGA, THETA0 )

CALCULATE THE TARGET POSITION AT "TTIME"

10 STEP = TTIME - TIME
CALL ADVANCE(TTALT, TTLNG, TTTLAT, TALT1, TLLNG1, TALT1, TTSPD,
X TTHED, TSPD1, THED1, CLIMB, ACCEL, TURNG, STEP,
X RGL1, RGL2, RGL3, RE ,V1, V2, V3)
CALL OBSERVE( RLNG, RLAT, RHGT, RE, RGL1, RGL2, RGL3, RANGE,
X AZIMTH, ELVATN , RRL1, RRL2, RRL3)

SEE IF TARGET IN "CONOE OF SILENCE"

IF( ELVATN .GT. 89. ) GO TO 90

FIND "DELTA", THE CLOCKWISE DISTANCE FROM THE BEAM AZIMUTH AT
"TTIME" (BMAZO) TO THE TARGET AZIMUTH AT "TTIME" (AZIMTH).

DELTA = AZIMTH - BMAZO
IF ( DELTA .LT. 0. ) DELTA = DELTA + 360.

IF DELTA SHOWS THE BEAM AND TARGET ARE COINCIDENT, ADVANCE
THE BEAM ONE DEGREE AND PROCEED

IF ( DELTA .GE. 1.0986328E-2 ) GO TO 15
   BMAZO = BMAZO + 1.
   DELTA = 359.
15 CONTINUE
IF ( DELTA .LT. 0. ) DELTA = DELTA + 360.
IF ( DELTA .LE. 180. .AND. DELTA .NE. 0. ) GO TO 30

BEAM AZIMUTH IS COINCIDENT WITH, OR LESS THAN 180 DEGREES
AHEAD OF TARGET AZIMUTH. SLEW BEAM AHEAD 180 DEGREES, SO THAT
BEAM IS BEHIND TARGET.

   BMAZO = BMAZO + 180.
   IF ( BMAZO .GE. 360. ) BMAZO = BMAZO - 360.

FIND THE TIME WHEN BEAM AT "BMAZO". (THIS IS NOT A LINEAR
FUNCTION BECAUSE THE ANTENNA ROTATION RATE MAY VARY).

CALL AZTIME( BMAZO, TTIME1, STSCAN, OMEGA, THETA0 )

CHECK TO SEE IF WE ARE IN THE NEXT SCAN

IF ( TTIME1 .GT. TTIME ) GO TO 20
   TTIME1 = TTIME1 + PERIOD(RADAR)
   STSCAN = STSCAN + PERIOD(RADAR)
20 TTIME = TTIME1

REPEAT ABOVE WITH NEW BEAM AZIMUTH

GO TO 10

AT "TTIME": BEAM IS AT "BMAZO", AND IS BEHIND TARGET. THE
TASK NOW IS TO ADVANCE THE BEAM UNTIL IT IS AHEAD OF THE
TARGET.

FIRST, SAVE THE TARGET POSITION AT "TTIME"

   TTALT = TALT1
   TTLNG = TLNG1
   TTLAT = TLAT1
   TTSPD = TSPD1
   TTHED = THED1
   TAZO = AZIMTH

ADVANCE THE BEAM BY INCREMENTING ITS AZIMUTH IN SMALL (E.G. 10
DEGREE) STEPS. (THE DO-LOOP IS A SAFETY PRECAUTION).
AZINCR = 10.
K = 360.D0 / AZINCR + 1.

DO 50 I = 1, K
  BMAZ1 = BMAZ0 + AZINCR
  IF( BMAZ1 .GE. 360. ) BMAZ1 = BMAZ1 - 360.
  CALL AZTIME( BMAZ1, TTIME1, STSCAN, OMEGA, THETA0 )
  IF( TTIME1 .GT. TTIME ) GO TO 40
  TTIME1 = TTIME1 + PERIOD(RADAR)
  STSCAN = STSCAN + PERIOD(RADAR)

MOVE TARGET TO TIME "TTIME1"

STEP = TTIME1 - TTIME
CALL ADVANCE( TTALT, TTLNG, TTLAT, TALT1, TLNG1, TLAT1,
                TTSPD, TTHED, TSPD1, TTHED1, CLIMB, ACCEL,
                TURNG, STEP, RG1, RG2, RG3, RE ,V1,V2,V3)
CALL OBSERVE( RLNG, RLAT, RHGT, RE, RG1, RG2, RG3, RANGE,
                AZIMTH, ELVATN ,RR1,RR2,RR3)

ABANDON THIS IF TARGET IN "CONE OF SILENCE"

IF( ELVATN .LE. 89. ) GO TO 45
  TTIME = TTIME1
  GO TO 90

SEE IF THE BEAM IS NOW AHEAD OF THE TARGET

DELTA = AZIMTH - BMAZ1
IF( DELTA .LT. 0. ) DELTA = DELTA + 360.
IF( DELTA .GT. 180. ) GO TO 60

NOT YET, SAVE BEAM AND TARGET POSITIONS AND TRY AGAIN.

BMAZ0 = BMAZ1
TTIME = TTIME1
TTALT = TALT1
TTLNG = TLNG1
TTLAT = TLAT1
TTSPD = TSPD1
TTHED = TTHED1
TAZ0 = AZIMTH

CONTINUE

*** ERROR *** WE'VE STEPPED COMPLETELY AROUND THE RADAR AND
HAVEN'T CROSSED THE TARGET.

STOP 'BEAM/TARGET INTERSECTION NOT POSSIBLE'

AT THIS POINT, WE KNOW THE POSITION OF THE BEAM (BMAZ0) AND
THE POSITION OF THE TARGET (TAZ0) AT TIME "TTIME"; WE KNOW THE
POSITION OF THE BEAM (BMAZ1) AND THE POSITION OF THE TARGET
(AZIMTH) AT "TTIME1". WE ALSO KNOW THAT THE BEAM CROSSES THE
TARGET SOMEWHERE BETWEEN BMZ0 AND BMZ1.

BECAUSE BMZ0 AND BMZ1 ARE CLOSE (E.G. 10 DEGREES), THE
TARGET PATH AND THE BEAM PATH MAY BE APPROXIMATED BY TWO
STRAIGHT LINES
IN THE "TIME - AZIMUTH" COORDINATE SYSTEM. SOLUTION OF THE
BEAM/TARGET INTERSECTION THEN BECOMES A SIMPLE MATTER OF
SOLVING
THE CROSSING POINT OF TWO LINES.

CONTINUE
B = AZIMTH - TA0
IF( B .LT. 0. ) B = B + 360.
IF( B .GT. 180. ) B = B - 360.
DELTA = TA0 - BMZ0
IF( DELTA .LT. 0. ) DELTA = DELTA + 360.
TIME1 = TTIME + DELTA * ( TTIME1 - TTIME ) / ( AZINCR - B )

INTERSECTION THUS OCCURS AT TIME "TIME1". NOW WE CALCULATE THE
BEAM AND TARGET POSITIONS FOR THIS TIME.

CALL TIMEAZ( BMZ1, TIME1, OMEGA, THETA0 )
STEP = TIME1 - TTIME
CALL ADVANCE(TTLT, TTLNG, TTLAT, TLT1, TLTG1, TLT1, TTSPD,
X TTHED, TSPD1, THEB1, CLMB, ACCEL, TURNG, STEP,
X RG1, RG2, RG3, RE, V1, V2, V3)
CALL OBSERVE( RLNG, RLAT, RHGT, RE, RG1, RG2, RG3, RANGE,
X AZIMTH, ELVATN, RR1, RR2, RR3)

SEE IF TARGET POSITION IS IN "CONES OF SILENCE"

IF( ELVATN .LE. 89. ) GO TO 70
TTIME = TIME1
GO TO 90

VERIFY THAT BEAM AND TARGET AZIMUTHS ARE WITHIN 1.E-4 DEGREES
( = 1.75E-6 RADIANS )

DELTA = BMZ1 - AZIMTH
IF( DELTA .LT. 0. ) DELTA = DELTA + 360.
IF( DELTA .GT. 180. ) DELTA = DELTA - 360.
IF( ABS( DELTA ) .LT. 5.E-4 ) RETURN

GOSH-DARN, WE'RE NOT CLOSE ENOUGH. NARROW THE AZIMUTH
INCREMENTS AND REPEAT.

AZINCR = AZINCR / 2.
STSCAN = DINT( TTIME / PERIOD(RADAR) ) * PERIOD(RADAR)
GO TO 35

TARGET IS IN "CONES OF SILENCE" AT TIME "TTIME". SLEW BEAM 180
DEGREES AND REPEAT ENTIRE SUBROUTINE.
C FIRST, FIND BEAM AZIMUTH AT "TTIME"
C
90    CALL TIMEAZ( BMAZO, TTIME, OMEGA, THETA0 )
STSCAN = DINT( TTIME / PERIOD(RADAR) ) * PERIOD(RADAR)
C
C SLEW THE BEAM 180 DEGREES
C
BMAZO = BMAZO + 180.
IF( BMAZO .GE. 360. ) BMAZO = BMAZO - 360.
C
C FIND TIME BEAM AT "BMAZO"
C
CALL AZTIME( BMAZO, TTIME1, STSCAN, OMEGA, THETA0 )
C
C CHECK TO SEE IF NEW SCAN STARTED
C
IF( TTIME1 .GT. TTIME ) GO TO 95
TTIME1 = TTIME1 + PERIOD(RADAR)
STSCAN = STSCAN + PERIOD(RADAR)
95    TTIME = TTIME1
C
C GO BACK TO THE BEGINNING
C

GO TO 10
END
C=========================================

SUBROUTINE INIT(POSITION,RADARS,RLNG, 
    ,
    RLAT,RHGT,RE,PERIOD,RLIM,ELIM,PFA, 
    ,
    RBIAS,ABIAS,EBIAS,ENOIS,THETA0, 
    ,
    THQ,SNR1,WVL,TOW,BWA, 
    ,
    BWE,MXT,XNHT,PRF,NPROC,RADFN)
C SUBROUTINE TO READ IN RADAR DATA AND TO COMPUTE
C INITIAL RADAR PARAMETERS
C B. J. ROOK JUNE 1983
C INPUT PARAMETERS ARE:
C POSITION=UNIT NUMBER FOR READING IN INITIAL RADAR DATA
C RADARS=NUMBER OF RADARS TO BE USED
C OUTPUT PARAMETERS ARE:
C RLNG=RADAR LONGITUDE(RADS)
C RLAT=RADAR LATITUDE(RADS)
C RHGT=RADAR HEIGHT(M)
C RE=EARTH'S RADIUS
C PERIOD=ANTENNA PERIOD OF REVOLUTION(SECS)
C RLIM=MAX. RANGE OF DETECTION FOR SPECIFIED PROB. OF DET.
C AND PROB. FALSE ALARM
C ELIM=FIXED ANTENNA ELAVATION POINTING ANGLE(DEG)
C PFA=RADARS' PROB. OF FALSE ALARM
C RBIAS=SYSTEMATIC RANGE ERROR(M)
C ABIAS=SYSTEMATIC AZIMUTHAL ERROR(DEG)
C EBIAS=SYSTEMATIC ELEVATION ERROR(DEG)
C ENOIS=COMPUTED RECEIVER NOISE(W)
C THETA0=INITIAL AZIMUTHAL POSITION OF ANTENNA(DEG)
C THQ=COMPUTED 1/E-POWER BEAMWIDTH(DEG) (TWO-WAY)
SNR1 = COMPUTED SIGNAL-TO-NOISE RATIO PER PULSE FOR THE
SPECIFIED PROB. OF DET. & PROB. FALSE ALARM
WVL = RADAR WAVELENGTH(M)
TOW = PULSE LENGTH(SECS)
BWA = 1/2-POWER BEAMWIDTH IN AZIMUTH(DEG) (ONE-WAY)
BWE = 1/2-POWER BEAMWIDTH IN ELEVATION(DEG) (ONE-WAY)
MAXT = MAXIMUM TIME THE RADAR IS RUNNING(SECS)
XNHT = COMPUTED NO. OF HITS ON TARGET FROM 1/E-BEAMWIDTH
IN AZIMUTH
PRF = PULSE REPETITION FREQUENCY(HZ)
NPROC = NUMBER OF PULSES FOR COHERENT INTEGRATION

COMMON PI, TWOPI, DR, RD
COMMON /DML/ DEVPER(10), RNUMBR, X(10), DELAY(10)
REAL*8 RLGNG(10), RLAT(10), RE, PI, TWOPI, DR, RD
REAL MAXT(10)
INTEGER POSITION, RADARS, RNUMBR
CHARACTER*20 RADFN
DIMENSION RHGT(10), ELIM(10), PROBDET(10), PFA(10), RBIAS(10),
, ABIAS(10), EBIAS(10), TOW(10), PRF(10),
, PERIOD(10), ELIM(10), BWA(10), THETA0(10), BWE(10)
,, PK(10), WVL(10), XLS(10), ENOIS(10), SNR1(10), XNHT(10), THQ(10)
,, NPROC(10)

OPEN (POSITION, FILE = RADFN)
READ (POSITION, 11) RADARS
DO 100 I = 1, RADARS
C READ IN THE RADARS' POSITION AND CONVERT TO RADIANS
READ (POSITION, 12) LNGD, LNGM, LNGS
RLNG(I) = DR*SIGN(1, LNGD)*
+ (ABS(LNGD) + LNGM/60.00 + LNGS/3600.00)
READ (POSITION, 12) LATD, LATM, LATS
RLAT(I) = DR*SIGN(1, LATD)*
+ (ABS(LATD) + LATM/60.00 + LATS/3600.00)
C READ IN REMAINING RADAR DATA
READ (POSITION, 13) RHGT(I)
READ (POSITION, 14) ELIM(I), PERIOD(I), PK(I), THETA0(I),
* BWA(I), BWE(I), WVL(I), PROBDET(I), PFA(I), RBIAS(I), TOW(I),
* ABIAS(I), PRF(I), EBIAS(I)
, ENOIS(I), XLS(I), MAXT(I), NPROC

RNPROC IS READ AS A REAL NUMBER TO SATISFY THE FORMAT
STATEMENT WHICH WAS DEVELOPED ON THE CP6 MACHINE. IT IS
DONE THIS WAY TO MINIMISE CHANGES IN CONVERTING THE PROGRAM
TO RUN ON THE RLVAX
IT APPEARS THAT CP6 FORTRAN ALLOWS YOU TO READ AN INTEGER
USING A FORMAT STATEMENT WITH A G SPECIFIER (E.G. G10.0)

NPROC(I) = INT(RNPROC)
DEVPER(I) = 0.0
100 CONTINUE
11 FORMAT (13)
12 FORMAT (I3, 1X, I2, 1X, I2)
13 FORMAT (F8.0)
14 FORMAT(G10.0)
   RE=6378.D3
   DO 101 I = 1 , RADARS
C COMPUTE SIGNAL-TO-NOISE RATIO PER PULSE FROM SPECIFIED PROB.
C OF FALSE ALARM(PFA(I)) & PROB. OF DETECTION(PROBDET(I))
   SNR1(I)=ALOG10(PFA(I))/ALOG10(PROBDET(I))-1.
C COMPUTE RADAR LOSSES XLS(I)DB TO RATIO
   XLS(I)=10.**(XLS(I)/10.)
C CONVERT NOISE FIGURE ENOIS(I)DB TO RATIO
   FN=10.**(ENOIS(I)/10.)
C COMPUTE RECEIVER NOISE
   ENOIS(I)=FN*1.38E-23*290.
C COMPUTE ANTENNA ONE-WAY POWER GAIN
   XG=32000./(BWA(I)*BWE(I))
C COMPUTE 1/E BEAMWIDTH(DEC)
   THQ(I)=.85*BWA(I)
C COMPUTE NUMBER HITS ON TARGET
   NHT=THQ(I)*PRF(I)/(6.*PERIOD(I))
   XNHT(I)=NHT
C COMPUTE MAX, RANGE OF DETECTION FOR SPECIFIED PROB. OF DET.
C AND PROB. OF FALSE ALARM FOR 1M**2 TARGET
   RMX4=PK(I)*1000.*TOW(I)*SQRT(XNHT(I))*XG*XG*WVL(I)*WVL(I)/
   *(4.*PI)**3*XLS(I)*ENOIS(I)*SNR1(I))
   RLIM=RMX4**.25
C DETERMINE PERIOD OF REVOLUTION(SECS)
   PERIOD(I)=60./PERIOD(I)
101 CONTINUE
   RETURN
END
C=====================================================================

SUBROUTINE PRODUCE (PLAN,HISTORY,RLNG,RLAT,RHGT,RE,RADAR,
   OMEGA,THETA0,RLIM,ELIM,RIAS,
   ABIAS,EBIAS,ENOIS,MAXTIM,MINTIM,
   THQ,SNR1,WVL,TOW,THA,THE,PFA,ELMIN,
   XNHT,PRF,IDENT,NPROC)
C PRODUCE A TIME-HISTORY FOR ONE TARGET.
C COMMON /RANDOM/ SEED
C COMMON PI,TWOPI,DR,RD
REAL*8 PI,TWOPI,DR,RD
INTEGER PLAN, HISTORY, DAY, HRS, RADAR
REAL*8 RLNG, RLAT, RE, TIME1, GMT(16), TLNG, TLAT, TIME,
   X, TLNG1, TLAT1, RG1, RG2, RG3, MAXTIM, MINTIM,STEP
   *,V1,V2,V3,XV1,XV2,XV3,RR1,RR2,RR3
REAL ACCEL(16), TURNG(16), CLIMB(16)
C
C CALL READFF (PLAN, NSEG, ID, GMT, ACCEL, TURNG, CLIMB,
   TLNG, TLAT, TALT, TSPD, THED, MAXTIM, MINTIM
   ,A1,A2,A3,A4)
C
C INITIALIZE TARGET TIME
C
   TIME = GMT(1)
C SOLVE ALL BEAM/TARGET INTERSECTIONS FOR THIS TARGET
C
DO 10 I = 1, NSEG
C
C SOLVE BEAM/TARGET INTERSECTION. CURRENT TARGET POSITION IS AT
C TLAT, TLNG, TALT, TSPPD, THED AT TIME "TIME". AT INTERSECTION,
C NEW TARGET POSITION IS AT TLAT1, TLNG1, TALT1, TSPPD1, THED1
C AT TIME "TIME1". THE INPUT VALUES ARE NOT CHANGED.
C
C NOTE: ROUTINE "CLOCK" WILL NOT ALLOW INTERSECTION TO BE AT OR
C BEFORE "TIME".
C
5 CALL CLOCK(TIME, TLAT, TLNG, TALT, TSPPD, THED, TIME1, TLAT1,
  X TLNG1, TALT1, TSPPD1, THED1, RANGE, AZIMTH, ELVATN,
  X CLIMB(I), ACCEL(I), TURNG(I), RLAT, RLNG, RHGT,
  X RE, THETA0, OMEGA, V1, V2, V3, RR1, RR2, RR3)
  IF( TIME1 .LE. GMT(I+1) ) GO TO 15
C
C INTERSECTION TIME EXCEEDS END OF SEGMENT TIME. ADVANCE TARGET
C TO END OF SEGMENT AND TRY FOR INTERSECTION IN NEW SEGMENT.
C
  STEP = GMT(I+1) - TIME
  IF (STEP .EQ. 0) GO TO 12
  CALL ADVANCE(TALT, TLNG, TALT, TALT1, TLNG1, TLAT1, TSPPD,
    X THED, TSPPD1, THED1, CLIMB(I), ACCEL(I), TURNG(I),
    X STEP, RG1, RG2, RG3, RE, V1, V2, V3)
  TIME = GMT(I+1)
  TALT = TALT1
  TLNG = TLNG1
  TLAT = TLAT1
  TSPPD = TSPPD1
  THED = THED1
  GO TO 10
C
C BEAM/TARGET INTERSECTION WITHIN SEGMENT - SAVE NEW TARGET
C POSITION
C
15 TALT = TALT1
  TLNG = TLNG1
  TLAT = TLAT1
  TSPPD = TSPPD1
  THED = THED1
  TIME = TIME1
  IF(ELVATN.LT.ELMIN)GO TO 5
C TRANSLATE VELOCITY COMPONENTS INTO AN EARTH COR-ORD SYSTEM
C CALL TRANS2G(V1, V2, V3, TLNG1, TLAT1, XV1, XV2, XV3)
C COMPUTE RADIAL VELOCITY BETWEEN RADAR AND TARGET
C CALL RVEL(RLNG, RLAT, RANGE, XV1, XV2, XV3, RR1, RR2, RR3, RELVEL)
  RELVEL=-RELVEL
C CONVERT TO DOPPLER SHIFT
  FDOP=2.*RELVEL/WVL
C IF IDENT=0 MEASUREMENT ERRORS ARE NOT ADDED
  IF(IDENT.EQ.0)GO TO 144
C COMPUTE DIFFERENCE IN ANGLE BETWEEN ELVATION ANGLE OF TARGET
C AND ANTENNA ELVATION POINTING ANGLE
DANG=(ELIM-ELVATN)/57.29578
C COMPUTE TARGET CROSS-SECTION(SIGMA)
   CALL ASPECT(AZIMTH,THD,A1,A2,A3,A4,SIGMA)
C COMPUTE SNR TAKING INTO ACCOUNT ELECTRICAL SCANNING IN
C ELEVATION
   SNR=(RLIM/RANGE)**4*SNR1*SIGMA*(COS(DANG))**2
C COMPUTE PROB. OF DETECTION
   PWR=1/(1.+SNR)
   PROBDET=PFA**PWR
C
C REJECT REPORT IF DETECTION NOT PROBABLE
C
   IF( RAN(SEED) .GT. PROBDET ) GO TO 5
C
C ADD NOISE AND QUANTIZATION ERRORS AND WRITE TO REPORT FILE
C
   CALL ERRORS(RANGE,AZIMTH,ELVATN,RBIAS,ABIAS,
      EBIAS,RELVEL,SNR,WVL,TOW,THQ,THA,
      THE,XNHT,PRF,NPROC)
   IF(AZIMTH.GT.360.)AZIMTH=AZIMTH-360.
   IF(AZIMTH.LT.0.)AZIMTH=360.+AZIMTH
   IF(ELVATN.GT.90.)ELVATN=90.-(ELVATN-90.)
C ADD QUANTIZATION ERRORS TO DOPPLER
   CALL QUANTF(RELVEL,WVL,PRF,NPROC,FDOP)
C ADD QUANTIZATION ERROR TO AZIMUTH ANGLE
   CALL QUANTA(THA,XNHT,AZIMTH,NPROC)
   IF(AZIMTH.GT.360.)AZIMTH=AZIMTH-360.
   IF(AZIMTH.LT.0.)AZIMTH=360.+AZIMTH
C ADD QUANTIZATION ERROR TO ELEVATION ANGLE
   CALL QUANTE(THA,ELVATN)
   IF(ELVATN.GT.90.)ELVATN=90.-(ELVATN-90.)
C TEST IF ELVATN .LT. ELMIN
144 IF(ELVATN.LT.ELMIN)GO TO 5
C CONVERT TIME TO DAY,HRS,MIN,SEC
   CALL TIMECON(TIME,DAY,HRS,MIN,SEC)
   WRITE(HISTORY,2) RADAR.DAY,HRS,MIN,SEC,
      ID,RANGE,AZIMTH,ELVATN,FDOP,RELVEL
   X   FORMAT(I3,2X,I3,2(':','!2),':'!3,F6.3,1X,A2,2(1X,F8.1),F7.2,
      2F10.1)
   GO TO 5
10 CONTINUE
   RETURN
END

SUBROUTINE READFP (PLAN,LINES,ID,GMT,ACCEL,TURNG,CLIMB,
   TLNG,TLAT,TDL,TSRD,THED,MAXTIM,MINTIM
   A1,A2,A3,A4)
C
   READ A FLIGHT PLAN FOR ONE TARGET.
COMMON PI,TWOPI,DR,RD
INTEGER PLAN, LINES, ID
REAL*8 GMT(16), TLNG, TLAT, PI, TWOPI, DR, RD, MAXTIM, MINTIM
REAL ACCEL(16), TURNG(16), CLIMB(16), TALT, TSPD, THED,
SEC, LNGS, LATS
INTEGER DAY, HRS, MIN, EVENT,
  , LNSGD, LNGLM, LATD, LATOM,
  , VANISH /'VA '/, NO /'NO '/
C
C READ FROM FLIGHT PLAN FILE:
C ID OF TARGET:   ERROR IF ID = 'NO' IS ENCOUNTERED;
C            MORE TARGETS SPECIFIED THAN SUPPLIED.
C EVENT: NOT USED EXCEPT TO CHECK THAT THE LAST EVENT
C            IS 'VA'NISH.  NO MORE THAN 15 EVENTS ALLOWED.
C DAY
C HRS: USED TO COMPUTE GMT(I) OF EVENT I IN SECONDS.
C MIN USER MUST SUPPLY EVENTS IN ASCENDING TIME ORDER.
C SEC
C CONTROL VARIABLES IN EFFECT FROM GMT(I) TO GMT(I+1)
C ACCEL(I): ACCELERATION TANGENTIAL TO PATH.
C            FORWARD IS POSITIVE.  M/S/S.
C            CANNOT GUARANTEE RESULTS IF SPEED GOES < 0.
C TURNG(I): TURNING ACCELERATION, NORMAL TO PATH.
C            LEFT TURN IS POSITIVE.  M/S/S.
C CLIMB(I): RATE OF CLimb OR DESCENT.
C            CLIMB IS POSITIVE.  M/S.
C STATE VARIABLES AT INITIAL TIME, Gmt(I)
C LNGD: TARGET LONGITUDE IN D:M:S.  EAST IS POSITIVE.
C LNGLM: USED TO COMPUTE TARGET LONGITUDE IN RADIANS.
C LNSGD    C
C LATD: TARGET LATITUDE IN D:M:S.  NORTH IS POSITIVE.
C LATOM: USED TO COMPUTE TARGET LATITUDE IN RADIANS.
C LATS
C TALT: TARGET INITIAL ALTITUDE IN METRES.
C TSPD: TARGET INITIAL GROUNDSPEED IN KILOMETRES PER
C HOUR EXTERNALLY, METRES PER SECOND INTERNALLY.
C THED: TARGET INITIAL HEADING IN DEGREES EXTERNALLY,
C RADIANS INTERNALLY.  POSITIVE FROM NORTH (ZERO)
C CLOCKWISE TO EAST.
C A1,A2,A3,A4=COEFFICIENTS FOR COMPUTING TARGET
C CROSS-SECTION
C
READ (PLAN,3) ID, DAY, HRS, MIN, SEC,
  , ACCEL(1), TURNG(1), CLIMB(1),
  , LNGD, LNGLM, LNSGD, LATD, LATOM, LATS,
  , TALT, TSPD, THED,SIGMAB
Gmt(I) = SEC + MIN*60. + HRS*3600. + DAY*86400.D0
IF( GMT(I) .LT. MINTIM ) MINTIM = GMT(I)
XLNGD=LNGD
XLNGM=LNGLM
XLNGS=LNSGD
XLATD=LATD
XLATOM=LATOM
XLATS=LATS
TLNG=DR*(SIGN(1.,XLNGD)*
  + (ABS(LNGD)+XLNGM/60.D0+XLNGS/3600.D0))
TLAT=DR*(SIGN(1.,XLATD)*(ABS(LATD)+XLATOM/60.D0+
  *XLATS/3600.D0))
TSPD = TSPD/3.6D0
THED = THED*DR

\[ \text{C} \]
\[ \text{IF (ID .EQ. NO)} \]
\[ \text{. STOP 'SPACE.READFP: FEWER TARGETS THAN DECLARED'} \]
\[ \text{C} \]
\[ \text{ELSE READ FLIGHT PLAN FOR THIS TARGET} \]
\[ \text{DO 1 I = 2, 16} \]
\[ \text{READ (PLAN,4) EVENT, DAY, HRS, MIN, SEC,} \]
\[ \text{ACCEL(I), TURNG(I), CLIMB(I)} \]
\[ \text{GMT(I) = SEC + MIN*60. + HRS*3600. + DAY*86400.D0} \]
\[ \text{IF (EVENT .EQ. VANISH) GO TO 2} \]
\[ \text{C} \]
\[ \text{THEN; FINISHED THIS TARGET. EXIT FROM LOOP.} \]
\[ \text{C} \]
\[ \text{ELSE; CONTINUE READING.} \]
\[ \text{1 CONTINUE} \]
\[ \text{2 LINES = I-1} \]
\[ \text{IF (GMT(I) .GT. MAXTIM) MAXTIM = GMT(I)} \]
\[ \text{CALL COEFS(SIGMAB,A1,A2,A3,A4)} \]
\[ \text{RETURN} \]
\[ \text{3 FORMAT (A2,2X,I1,2I2, F6.3, 2F8.0, F5.0, 2(I4, I2, F4.1),} \]
\[ \text{* F7.0, F5.0, F6.0/G5.0)} \]
\[ \text{4 FORMAT (2X, A2, I1, 2I2, F6.3, 2F8.0, F5.0)} \]
\[ \text{END} \]

\[ \text{C=} \]
\[ \text{SUBROUTINE TRANSI2G (RI1, RI2, RI3, LNG, LAT,} \]
\[ \text{RG1, RG2, RG3)} \]
\[ \text{TRANSFORM A VECTOR IN THE TARGET FRAME} \]
\[ \text{TO THE GEOGRAPHIC FRAME.} \]
\[ \text{USING: RG = ù-LNGY3*ù+LATY2*RI} \]
\[ \text{C} \]
\[ \text{TARGET: ORIGIN AT EARTH CENTRE} \]
\[ \text{1I - OUTWARD NORMAL TO THE SPHERE (UP)} \]
\[ \text{C} \]
\[ \text{2I - EASTWARD} \]
\[ \text{C} \]
\[ \text{3I - NORTHWARD} \]
\[ \text{C} \]
\[ \text{GEOGRAPHIC: ORIGIN AT EARTH CENTRE} \]
\[ \text{1G - TOWARD INTERSECTION OF EQUATOR AND} \]
\[ \text{PRIME MERIDIAN.} \]
\[ \text{C} \]
\[ \text{2G - MAKES A RIGHT-HANDED SET WITH 1G AND 3G.} \]
\[ \text{C} \]
\[ \text{3G - NORTHWARD NORMAL TO EQUATORIAL PLANE.} \]

\[ \text{C=} \]
\[ \text{IMPLICIT REAL*8 (A-Z)} \]
\[ \text{CLNG = DCOS(LNG)} \]
\[ \text{SLNG = DSIN(LNG)} \]
\[ \text{CLAT = DCOS(LAT)} \]
\[ \text{SLAT = DSIN(LAT)} \]
\[ \text{RG1 = CLNG*CLAT*RI1 - SLNG*RI2 - CLNG*SLAT*RI3} \]
\[ \text{RG2 = SLNG*CLAT*RI1 + CLNG*RI2 - SLNG*SLAT*RI3} \]
\[ \text{RG3 = SLAT*RI1 + CLAT*RI3} \]
\[ \text{RETURN} \]
\[ \text{END} \]

\[ \text{C=} \]
\[ \text{SUBROUTINE OBSERVE (RLNG, RLAT, RHGT, RE,} \]
\[ \text{RG1, RG2, RG3,} \]
\[ \text{RANGE, AZIMTH, ELVATN,RR1,RR2,RR3)} \]
COMPUTE OBSERVATIONS FROM GEOGRAPHIC POSITION.

C USING: RR = U90-RLATY2*U+RLNGY3*RG - U0, 0, RHGT+REYT
C RADAR FRAME: ORIGIN AT THE RADAR
C 1R - SOUTHWARD
C 2R - EASTWARD
C 3R - UP
C
C COMMON PI,TWOPI,D2R,R2D
REAL*8 RLNG, RLAT, RE, RG1, RG2, RG3, CLNG, CLAT, SLNG,
X SLAT, RR1, RR2, RR3, PI, TWOPI, D2R, R2D
C
CLA = DCOS(RLNG)
SLG = DSIN(RLNG)
CLAT = DCOS(RLAT)
SLAT = DSIN(RLAT)
C
RR1 = SLAT*CLNG*RG1 + SLAT*SLNG*RG2 - CLAT*RG3
RR2 = SLNG*RG1 + CLNG*RG2
RR3 = +CLAT*CLNG*RG1 + CLAT*SLNG*RG2 + SLAT*RG3
*DBLE(RHGT)-RE
C
RANGE = DSQRT(RR1**2 + RR2**2 + RR3**2)
AZIMTH = 180. - R2D*DATAN2(RR2,RR1)
IF( AZIMTH .LT. 0. ) AZIMTH = AZIMTH + 360.
IF( AZIMTH .GT. 360. ) AZIMTH = AZIMTH - 360.
YY=RR3/RANGE
IF (YY.GT.1.0) THEN
  IF (YY.GT.1.0000001) THEN
    WRITE(*,*)'WARNING - ASIN ARGUMENT CORRECTED TO 1. FROM '
    + ',YY
  ENDF
  YY=1.0
ENDIF
ELVATN=57.29578*ASIN(YY)
C
TYPE RANGE,AZIMTH,ELVATN
RETURN
END

SUBROUTINE TIMECON (TIME, DAY, HRS, MIN, SEC)
C
C CONVERT TIME IN SECONDS
C TO DAYS, HOURS, MINUTES AND SECONDS.

INTEGER DAY, HRS, MIN
REAL SEC
REAL*8 TIME, A
A = DINT( TIME * 1000. + 0.5 )
DAY = A / 86400.D3
A = A - DAY * 86400.D3
HRS = A / 3600.D3
A = A - HRS * 3600.D3
MIN = A / 60.D3
SEC = ( A - MIN * 60.D3 ) / 1000.D0
RETURN
END
SUBROUTINE ADVANCE (TALT, TLNG, TLAT, TALT1, TLNG1, TLAT1,
, TSPD, THED, TSPD1, THED1, 
, CLIMB, ACCEL, TURNG, STEP,
, RG1, RG2, RG3, RE,V1,V2,V3)

COMMON PI,TWOPI,DR, RD
REAL*8 TLNG, TLAT, RE, RG1, RG2, RG3, RI1, RI2, RI3,
X PI, TWOPI, DR, RD, STEP, TLNG1, TLAT1, HALFP1, PHII,
X PHIF, V1, V2, V3

REAL TALT, TSPD, THED, CLIMB, ACCEL, TURNG, TALT1, TSPD1, THED1
HALFP1 = .5D0*PI
TALT1 = TALT + CLIMB*STEP
RI1 = DBLE(TALT1) + RE

IF (TURNG .NE. 0.) GOTO 2
TSPD1 = TSPD + ACCEL*STEP
THED1 = THED
TERM = (TSPD + TSPD1)/2.
RI2 = STEP*DBLE(TERM*SIN(THED1))
RI3 = STEP*DBLE(TERM*COS(THED1))
GO TO 1

C ELSE
2
PHII = HALFP1 - THED
PHIF = PHII + TURNG/TSPD*STEP
THED1 = HALFP1 - PHIF
TSPD1 = TSPD
RI2 = DBLE( TSPD )**2/TURNG* ( DSIN(PHIF) - DSIN(PHII) )
RI3 = DBLE( TSPD )**2/TURNG* ( DCOS(PHII) - DCOS(PHIF) )

C CONTINUE
CALL TRANS12G (RI1, RI2, RI3, TLNG, TLAT, 
, RG1, RG2, RG3)

TLNG1 = DATAN2(RG2, RG1)
TLAT1 = DASIN(RG3/DSQRT(RG1**2 + RG2**2 + RG3**2))
V1= DBLE(CLIMB)
V2=RI2/STEP
V3=RI3/STEP
RETURN
END

SUBROUTINE ERRORS( R, A, E, RBIAS, 
, ABIAS, EBIAS, 
, FDOP, SNR, WVL, TOW, THQ, THA, THE, 
, XNHT, PRF, NPROC)

C APPLY BIAS AND GAUSSIAN NOISE TO MEASUREMENTS.

C DATA PI/3.14159265/
SNRC=SNR/SQRT(XNHT)

C COMPUTE RANGE ERRORS
DR=3.88*TOW/(4.*SQRT(SNR))
CALL GAUSS (RBIAS, DR, ERROR)
R=R+ERROR

C COMPUTE ANGULAR ERRORS
SIG=2./(.85*SQR(SNR))
    CALL GAUSS (ABIAS, SIG, ERROR)
DA=ERROR*THA
    A=A+DA
    CALL GAUSS (EBIAS, SIG, ERROR)
DE=ERROR*THE
    E=E+DE
C COMPUTE DOPPLER ERRORS
    DF=PRF/FLOAT(NPROC)
    D=WVL*DF/(4.*SQR(FLOAT(NPROC)*SNRC))
    CALL GAUSS(0.,D,ERROR)
    FDOP=FDOP+ERROR
    RETURN
END
C==============================================

SUBROUTINE GAUSS (BIAS, SIGMA, RANDOM)
COMPUTE A NORMALLY-DISTRIBUTED RANDOM NUMBER
WITH A GIVEN MEAN AND STANDARD DEVIATION.
C
C ALGORITHM IS THAT OF C.R.C. COMUS MANUAL
C
COMMON /RANDOM/  SEED
    RANDOM = SQRT(-2.*ALOG(RAN(SEED)))*COS(6.28319*RAN(SEED))
    RANDOM = RANDOM*SIGMA + BIAS
    RETURN
END
C===============================================

SUBROUTINE AZTIME( AZMUTH, TIME, STSCAN, OMEGA, THETA0 )
C
C CALCULATE TIME THAT BEAM IS AT AZMUTH
C
C ARGUMENTS:
C    AZMUTH - AZIMUTH OF BEAM (DEG.)
C    TIME - TIME WHEN BEAM IS AT "AZMUTH" TO NEAREST MSEC.
C          (SEC.)
C    STSCAN - TIME OF START OF THIS SCAN (SEC.)
C    OMEGA - ROTATION RATE OF RADAR (SEC.)
C    THETA0 - START AZIMUTH OF RADAR (DEG.)
C
COMMON PI, TWOPI, DR, RD
COMMON /DML/ DEVPER(10), RADAR, PERIOD(10), DELAY(10)
INTEGER RADAR
REAL*8 TIME, PI, TWOPI, DR, RD, STSCAN, LAZMTH, TIM1, TEMP,
     DTHETA, THETA
C
C MAKE THE INPUT AZMUTH RELATIVE TO THE START OF THE RADAR
C
    LAZMTH = AZMUTH - THETA0
    IF( LAZMTH .LT. 0. ) LAZMTH = LAZMTH + 360.
    IF( LAZMTH .GE. 360. ) LAZMTH = LAZMTH - 360.
    TIM1 = LAZMTH / OMEGA

71
AN ITERATIVE METHOD IS USED TO SOLVE FOR THE TIME. THE DO-LOOP LIMITS THE NUMBER OF ITERATIONS.

```
DO 10 I = 1, 10
  TEMP = OMEGA * DR * TIM1
  THETA = TEMP + DEVPER(RADAR) * ( 1. - DCOS( TEMP ) )
  DTHETA = LAZMTH * DR - THETA
  TIME = TIM1 + DTHETA / ( OMEGA * DR * ( 1. + )
  DEVPERS(RADAR) * DSIN( TEMP ) )
  IF( DABS( TIME - TIM1 ) .LT. 1.D-4 ) GO TO 20
  TIM1 = TIME
10 CONTINUE
20 TIME = TIME + STSCAN
RETURN
END
```

SUBROUTINE TIMEAZ( AZMUTH, TIME, OMEGA, THETA0 )

ROUTINE TO CALCULATE AZIMUTH OF RADAR BEAM AT "TIME"

ARGUMENTS:
AZMUTH - AZIMUTH OF RADAR BEAM (DEG.)
TIME - INPUT TIME (SEC.)
OMEGA - ROTATION RATE OF RADAR (SEC.)
THETA0 - STARTING AZIMUTH OF RADAR (DEG.)

COMMON PI,TWOPIDR, RD
COMMON / DML / DEVPER(10), RADAR, PERIOD(10), DELAY(10)
INTEGER RADAR
REAL*8 TIME, PI, TWOPI, DR, RD, TTIME, FPER

```
TTIME = TIME
10 IF( TTIME .LT. PERIOD(RADAR) ) GO TO 20
   TTIME = TTIME - PERIOD(RADAR)
   GO TO 10
20 FPER = TWOPI * TTIME / DBLE( PERIOD(RADAR) )
   AZMUTH = RD* ( FPER + DEVPER(RADAR) * ( 1. - DCOS(FPER)) )
   IF( AZMUTH .GE. 360. ) AZMUTH = AZMUTH - 360.
RETURN
END
```

SUBROUTINE CLUTTER(PLAN,HISTORY,RLNG,RLAT,RHGT,RE,TREV,
              RADAR,OMEGA,THETA0,
              SNR1,WVL,RLIM,FPA,TOW,THQ,BWA,BWE,
              ELMIN,ELIM,XNHT,PRF,NPROC)

ROUTINE GENERATES A CLUTTER PATCH AROUND THE GIVEN POLAR COORDINATES.
B. J. ROOK JUNE 1983
INPUT PARAMETERS ARE:

PLAN=UNIT NUMBER FOR READING CLUTTER DATA

HISTORY=UNIT NUMBER FOR WRITING CLUTTER REPORTS ON

RLNG=LONGITUDE OF RADAR(RADS)

LAT=LATITUDE OF RADAR(RADS)

RHGT=HEIGHT OF RADAR ABOVE GROUND(M)

RE=EARTH'S RADIUS(M)

TREV=RADAR ANTENNA PERIOD OF REVOLUTION(SECS)

RADAR=RADAR NUMBER CURRENTLY BEING USED

OMEGA=RATE OF ANTENNA REVOLUTION(DEG/SEC)

THETA0=INITIAL AZIMUTHAL ANGLE OF ANTENNA(DEG)

SNR1=SIGNAL-TO-NOISE RATIO FOR A SINGLE PULSE FOR A

SPECIFIED PROB. OF DET. AND PROB. OF FALSE ALARM

WVL=RADAR WAVELENGTH(M)

RLIM=MAX. RANGE OF DETECTION FOR SPECIFIED PROB. OF DET.

AND PROB. OF FALSE ALARM

PFA=THE RADAR'S SPECIFIED PROB. OF FALSE ALARM

TOW=PULSE WIDTH(SECS)

THQ=1/E ANTENNA BEAMWIDTH(DEG) IN AZIMUTH (TWO-WAY)

BWA=1/2-POWER BEAMWIDTH(DEG) IN AZIMUTH (ONE-WAY)

BWE=1/2-POWER BEAMWIDTH IN ELEVATION(DEG) (ONE-WAY)

ELMIN=MINIMUM ELEVATION ANGLE(DEG)

ELIM=ANTENNA FIXED POINTING ANGLE IN ELEVATION(DEG)

XNHT=NUMBER OF HITS ON TARGET FOR CURRENT RADAR

COMPUTED FROM THE 1/E-BEAMWIDTH

PRF=PULSE REPETITION FREQUENCY(HZ)

NPROC=NUMBER OF PULSES FOR COHERENT PROCESSING

COMMON /RANDOM/ SEED

COMMON PI,TWOPI,D2R,RD

REAL*8 PI,TWOPI,D2R,RD

INTEGER PLAN,HISTORY,DAY,HRS,MIN,RADAR

REAL*8 RLNG,RLAT,RE,GMT(2),TLNG,TLAT,TIME,FTIME(300),

X TLNG1,TLAT1,TLG,TLT,

X TIME1,V1,V2,V3,RR1,RR2,RR3,XV1,XV2,XV3

X ,RG1,RG2,RG3

REAL FD(300),VL(300)

REAL AF(300),RF(300),EF(300),CLIMB/0./,ACCEL/0./,

, TURNG/0./

READ POSITION ,TIME ,DENSITY ,AREA AND MOVEMENT PARAMETERS

CALL READCP(PLAN, ID, GMT, TLNG, TLAT, TALT, TSPD, THED,

, DENSITY, DX2E, DX1N, DX3D, SIG)

INITIALIZE BEAM POSITION

TIME = GMT(1)

CALL CLOCK TO COMPUTE TIME BETWEEN BEAM TARGET

COINCIDENCE

100 CALL CLOCK(TIME, TLAT, TLNG, TALT, TSPD, THED, TIME1, TLAT1,

X TLNG1, TALT1, TSPD1, THED1, RANGE, AZIMTH, ELVATN,
X = CLIMB, ACCEL, TURN, RLAT, RLNG, RHGT, RE, THETA0,
    OMEGA, V1, V2, V3, RR1, RR2, RR3
IF( TIME1 .GT. GMT(2) ) RETURN
TLNG = TLNG1
TLAT = TLAT1
TALT = TALT1
TSPD = TSPD1
THED = THED1
TIME = TIME1

C CALCULATE THE MEAN NUMBER OF CLUTTER POINTS WITHIN THE CLUTTER
C VOLUME DESCRIBED BY THE LENGTHS DX1N, DX2E, AND DX3D
C
XMEAN = DX1N * DX2E * DX3D * DENSITY
IF( XMEAN .GT. 300. ) XMEAN = 300.
C COMPUTE A RANDOM INTEGER NUMBER (NUM) FROM A POISSON
C DISTRIBUTION HAVING A MEAN VALUE OF XMEAN
CALL POISION(XMEAN, NUM)
C
C CONVERT AZIMUTH AND ELEVATION MEASUREMENTS TO RADIANS
C
AZZ = AZIMTH / 57.29578
ELL = ELVATN / 57.29578
C SET COUNTER (ICNT) TO ZERO
ICNT = 0
DO 200 I = 1, NUM
C COMPUTE THE RADAR'S LOOK DIRECTION IN A N-E-D COORDINATE
C SYSTEM
C AND ADD A RANDOM LENGTH IN THE RANGE OF +/- .5*DX1N TO THE
C N COMPONENTS, +/- .5*DX2E TO THE E COMPONENT AND
C +/- .5*DX3D TO THE D COMPONENT
C
X1 = N
X = RAN(SEED)
X1 = RANGE * COS(AZZ) * COS(-ELL) + DX1N * 1000. * (.5 - X)
C
X2 = E
X = RAN(SEED)
X2 = RANGE * SIN(AZZ) * COS(-ELL) + DX2E * 1000. * (.5 - X)
C
X3 = D
X = RAN(SEED)
X3 = RANGE * SIN(-ELL) - DX3D * 1000. * (.5 - X)
C
COMPUTE THE LOOK AZIMUTH ANGLE TO THIS CLUTTER POINT
THA = 57.29578 * ATAN2(X2, X1)
IF( THA.LT. 0. ) THA = 360. + THA
THAA = THA / 57.29578
C
COMPUTE THE RANGE TO THIS CLUTTER POINT
R = SQRT(X1*X1 + X2*X2 + X3*X3)
C
COMPUTE THE LOOK ELEVATION ANGLE TO THIS CLUTTER POINT
THE = -57.29578 * ASIN(X3/R)
THEE = THE / 57.29578
C
COMPUTE THE DIFFERENCE ANGLE BETWEEN THE LOOK ELEVATION ANGLE
C AND THE ANTENNA POINTING ANGLE
DANG = (ELIM - THE) / 57.29578
C
COMPUTE THE CLUTTER POINT POSITION IN LATITUDE (TLT)
C AND LONGITUDE (TLG)
CALL TPOS(RHGT, RE, RLNG, RLAT, R, THAA, THEE, 
   TLG, TLT, RG1, RG2, RG3, HT)
C RANDOMIZE THE VELOCITY COMPONENTS OF V2 & V3
   VV=V2/10.
   CALL GAUSS(0., VV, V)
   V2=V2+V
   VV=V3/10.
   CALL GAUSS(0., VV, V)
   V3=V3+V
C TRANSLATE VELOCITY COMPONENTS INTO AN EARTH CO-ORD. SYSTEM
   CALL TRANSI2G(V1, V2, V3, TLG, TLT, XV1, XV2, XV3)
C COMPUTE THE LOOK DIRECTION OF THE RADAR TO THE CLUTTER POINT
C IN AN S-E-UP CO-ORD SYSTEM
   CALL OBSERVE(RLNG, RLAT, RHGT, RE, 
   RG1, RG2, RG3, RA, AZ, EL, RR1, RR2, RR3)
C COMPUTE THE RADIAL VELOCITY BETWEEN RADAR AND TARGET
   CALL RVEL(RLNG, RLAT, R, XV1, XV2, XV3, RR1, RR2, RR3, RELVEL)
   RELVEL=RELVEL
C COMPUTE A RANDOM CROSS-SECTION FOR THE CLUTTER POINT
   CALL GAUSS(0., SIG, SIGX)
   CALL GAUSS(0., SIG, SIGY)
   SIGMA=SQRT(SIGX**2+SIGY**2)
C COMPUTE SNR TAKING INTO ACCOUNT ELECTRICAL SCANNING IN
C ELEVATION
   SNR=(RLIM/R)**4*SNR1*SIGMA*(COS(DANG))**2
C COMPUTE PROBABILITY OF DETECTION
   PWR=1./(1.+SNR)
   PROBDet=PPA**PWR
C REJECT REPORT IF DETECTION NOT POSSIBLE
   X=RAN(SEED)
   IF(X.GT.PROBDet)GO TO 200
   ICNT=ICNT+1
C INTRODUCE MEASUREMENT ERRORS
   CALL ERRORS(R, THA, THE, 0., 0., 0., RELVEL, SNR, WVL, TOW,
   *THQ, BWA, BWE, XNHT, PRF, NPROC)
   IF(THA.GT.360.)THA=THA-360.
   IF(THA.LT.0.)THA=360.+THA
   IF(THE.GT.90.)THE=90.-THE-90.
C ADD QUANTIZATION ERROR TO DOPPLER
   CALL QUANTF(RELVEL, WVL, PRF, NPROC, FDOP)
C ADD QUANTIZATION ERROR TO AZIMUTH ANGLE
   CALL QUANTA(BWA, XNHT, THA, NPROC)
C ADD QUANTIZATION ERROR TO ELEVATION ANGLE
   CALL QUANTE(BWE, THE)
   IF(THA.GT.360.)THA=THA-360.
   IF(THA.LT.0.)THA=360.+THA
   IF(THE.GT.90.)THE=90.-THE-90.
C ADD QUANTIZATION ERRORS TO RANGE
   CALL QUANTR(R, TOW)
   EF(ICNT)=THA
   RF(ICNT)=R
   FD(ICNT)=FDOP
   VL(ICNT)=RELVEL
   AF(ICNT)=THA
200 CONTINUE
C
C CALCULATE HIT TIMES FOR EACH CLUTTER POINT
C
AZREF=AZIMTH
DO 300 I=1,ICNT
AZCLT=AF(I)
DELTA=AZCLT-AZREF
IF(DELTA.GT.180.)DELTA=DELTA-360.
   FTIME(I)=TIME+((AF(I)/360.)*TREV
   FTIME(I)=TIME+DELTA*TREV/360.
C OUTPUT AZIMTH,AF(I),TIME,FTIME(I)
C OUTPUT DELTA
   IF(AF(I).GT.360.) AF(I)=AF(I)-360.
   IF(AF(I).LT.0.)AF(I)=360.+AF(I)
300 CONTINUE
C
C OUTPUT THE RESULTS
C
DO 400 I=1,ICNT
   IF(FTIME(I).LT.0.)GO TO 400
   CALL TIMECONV(FTIME(I),DAY,HRS,MIN,SEC)
   IF(EF(I).LT.ELMIN)GO TO 400
   WRITE(HISTORY,6) RADAR,DAY,HRS,MIN,SEC,
   X     ID,RF(I),AF(I),EF(I),FD(I),VL(I)
   6 FORMAT(I3,2X,I3,2((':',I2),':',F6.3,1X,A2,2(1X,F8.1),F7.2,
   X     2F10.1 )
400 CONTINUE
GO TO 100
END
C
C==============================================================================

SUBROUTINE READCP(PLAN, ID, GMT, TLNG, TLat, TALT, 
, TSPD, THED, DENSITY, DX2E, DX1N, DX3D, SIG)
C SUBROUTINE TO READ IN CLUTTER PLAN SCENARIO
C INPUT PARAMETERS ARE:
C PLAN=UNIT NUMBER FOR CLUTTER PLAN
C OUTPUT PARAMETERS ARE:
C ID=CLUTTER IDENTIFIER
C GMT=TIMES AT WHICH CLUTTER APPEARS AND ENDS(SECS)
C TLNG=LONGITUDE AT CENTRE OF PATCH(RADS)
C TLat=LATITUDE AT CENTRE OF PATCH(RADS)
C TALT=ALTITUDE AT CENTRE OF PATCH(M)
C TSPD=VELOCITY OF PATCH(KM/S)
C THED=HEADING OF CLUTTER PATCH
C DENSITY=DENSITY OF CLUTTER PATCH
C DX2E=LENGTH OF CLUTTER PATCH(KM)
C DX1N=DEPTH OF CLUTTER PATCH(KM)
C DX3D=WIDTH OF CLUTTER PATCH(KM)
C SIG=NUMBER FOR COMPUTING THE CLUTTER-POINT CROSS-SECTION
C
COMMON PI, TWOPI, D2R, R2D
INTEGER PLAN, DAY, HRS, MIN, LNGD, LNGM, LATD, LATM, VANISH

76
REAL*8 TLNG, TLAT, GMT(2), PI, TWOPI, D2R, R2D
REAL SEC, LNGS, LATS

C READ CLUTTER POSITION, SPEED, HEADING, DENSITY AND
C DIMENSIONS (DX1N, DX2E, DX3D)

C READ(PLAN, 1) ID, DAY, HRS, MIN, SEC,
C DX2E, DX1N, DX3D, LNGD, LNGM, LNGS,
C LATD, LATM, LATS, TALT, TSPD, THED, DENSITY, SIG

C 1 FORMAT(A2, 2X, I1, 2I2, F6.3, 3F7.0, 2(I4, I2, F4.1), F7.0
C , F5.0, F6.0/F7.4, F7.0)

C CLUTTER PATCH START TIME, LONGITUDE, LATITUDE, SPEED AND HEADING

C GMT(1) = SEC + MIN*60 + HRS*3600 + DAY*86400.0D0
C TLNG = D2R*(SIGN(1, LNGD)*(ABS(LNGD)+LNGM/60.0D0+LNGS/3600.0D0))
C TLAT = D2R*(SIGN(1, LATD)*(ABS(LATD)+LATM/60.0D0+LATS/3600.0D0))
C TSPD = TSPD/3.6D0
C THED = D2R*THED

C READ VANISH TIME FOR CLUTTER PATCH

C READ(PLAN, 2) VANISH, DAY, HRS, MIN, SEC
C 2 FORMAT(2X, A2, I1, 2I2, F6.3)
C GMT(2) = SEC + MIN*60 + HRS*3600 + DAY*86400.0D0
C RETURN
C END

C=================================================================================================

SUBROUTINE RVEL(RLNG, RLAT, RANGE, XV1, XV2, XV3, RR1, RR2, RR3, RELVEL)
C SUBROUTINE TO COMPUTE THE RADIAL VELOCITY BETWEEN RADAR &
C TARGET
C B. J. ROOK JUNE 1983
C INPUT PARAMETERS ARE:
C RLNG = LONGITUDE OF RADAR (RADS)
C RLAT = LATITUDE OF RADAR (RADS)
C RANGE = RADAR RANGE TO TARGET (M)
C XV1, XV2, XV3 = VELOCITY COMPONENTS OF TARGET IN AN EARTH
C CENTRED FRAME
C RR1, RR2, RR3 = LOOK DIRECTION OF RADAR IN AN UP-E-S COORD.
C SYST.
C OUTPUT PARAMETERS ARE:
C RELVEL = RADIAL VELOCITY BETWEEN RADAR & TARGET (M/S)
C
C REAL*8 RLNG, RLAT, XV1, XV2, XV3, CLNG, CLAT,
C *, SLNG, SLAT, RR1, RR2, RR3R1, R2, R3
C CLNG = DCOS(RLNG)
C SLNG = DSIN(RLNG)
C CLAT = DCOS(RLAT)
C SLAT = DSIN(RLAT)
C R1 = SLAT*CLNG*XV1 + SLAT*SLNG*XV2 - CLAT*XV3
C R2 = -SLNG*XV1 + CLNG*XV2
R3 = CLAT*CLNG*V1 + CLAT*SLNG*V2 + SLAT*V3
RELVEL = (R1*RR1 + R2*RR2 + R3*RR3) / RANGE
RETURN
END

SUBROUTINE TPOS(RHGT, RE, RLNG, RLAT, R, 
    TH, PHI, THT, PHT, X1, X2, X3, H)
C SUBROUTINE TO COMPUTE THE TARGET-CLUTTER POSITIONS IN A 
C EARTH CO-ORDINATE SYSTEM
C B. J. ROOK JUNE 1983
C INPUT PARAMETERS ARE:
C RHGT = RADAR HEIGHT (M)
C RE = EARTH'S RADIUS (M)
C RLNG = LONGITUDE OF RADAR (RADS)
C RLAT = LATITUDE OF RADAR (RADS)
C R = RADAR RANGE TO CLUTTER POINT (M)
C TH = AZIMUTH OF CLUTTER POINT AT THE RADAR (RADS)
C PHI = ELEVATION ANGLE OF CLUTTER POINT AT THE RADAR (RADS)
C OUTPUT PARAMETERS ARE:
C THT = COMPUTED CLUTTER POINT LONGITUDE (RADS)
C PHT = COMPUTED CLUTTER POINT LATITUDE (RADS)
C H = TARGET HEIGHT (M)
C REAL*8 X1, X2, X3, THT, PHT, RLAT, RLNG 
   +, RE, S, RL
DATA PI/3.14159265/
S = DBLE(RHGT) + RE
A = RLNG
B = RLAT + PHI*5
C SET UP COLUMN MATRIX IN A N-E-D CO-ORDINATE SYSTEM 
XN(1) = R*COS(TH)*COS(-PHI)
XN(2) = R*SIN(TH)*COS(-PHI)
XN(3) = R*SIN(-PHI)
C SET UP A 3X3 MATRIX ABOUT THE X2 AXIS OF ANGLE B 
CALL ST2(B, R2)
C SET UP A 3X3 MATRIX ABOUT THE X3 AXIS OF ANGLE -A 
CALL ST3(-A, R3)
C MULTIPLY THE TWO MATRICES, 3, 3, 3, RR, 3, IERROR)
CALL VMULPF(R3, R2, 3, 3, 3, 3, 3, RR, 3, IERROR)
IF (IERROR .EQ. 129) WRITE(*,*)
    + ' DIMENSION ERROR R3, R2 OR RR'
C MULTIPLY THE RESULTANT MATRIX RR BY THE COLUMN MATRIX XN 
CALL VMULFF(RR, XN, 3, 3, 1, 3, 3, X, 3, IERROR)
IF (IERROR .EQ. 129) WRITE(*,*)
    + ' DIMENSION ERROR R3, R2 OR RR'
C THE RESULTANT COLUMN MATRIX X IS IN THE EARTH CO-ORD. FRAME 
C ADD THESE COMPONENTS TO THE COMPONENTS OF THE POSITION OF THE 
C RADAR IN THE THE EARTH CO-ORD. FRAME 
X1 = S*DCOS(RLNG)*DCOS(RLAT) + DBLE(X(1))
X2 = S*DSIN(RLNG)*DCOS(RLAT) + DBLE(X(2))
X3 = S*DSIN(RLAT) + DBLE(X(3))
C COMPUTE THE CLUTTER-TARGET POINT LONGITUDE (THT) 
THT = DATAN2(X2, X1)
C COMPUTE THE RADIAL LENGTH OF TARGET-CLUTTER POINT
RL=DSQRT(X1*X1+X2*X2+X3*X3)
C COMPUTE THE CLUTTER-TARGET POINT LATITUDE(PHT)
PHT=ADASIN(X3/RL)
C COMPUTE CLUTTER-TARGET HEIGHT
H=RL-RE
RETURN
END

============================================

SUBROUTINE FALSE( PFA, RLIM, PERIOD, THETA0, RNUMBR,
                   X                      MAXTIM, HISTORY ,TOW,PRF,WVL,NPROC,
                   X                      XNHT,BWA,BWE)

C
C ADD FALSE ALARM REPORTS TO TRACK DATA
C B. J. ROOK JUNE 1983
C INPUT PARAMETERS ARE:
C PFA - PROBABILITY OF FALSE ALARM FOR CURRENT RADAR
C RLIM - RANGE LIMIT OF CURRENT RADAR ( M )
C PERIOD - PERIOD OF REVOLUTION OF CURRENT RADAR ANTENNA(SECS)
C THETA0 - STARTING AZIMUTH OF CURRENT RADAR (DEG.)
C RNUMBR - NUMBER (I.E. INDEX) OF CURRENT RADAR
C MAXTIM - MAXIMUM TIME THE RADAR IS RUNNING(SECS)
C HISTORY - UNIT NUMBER FOR RESULTS TO BE WRITTEN ON
C TOW - PULSE LENGTH OF CURRENT RADAR(SECS)
C PRF=PULSE-REPETITION-FREQUENCY(HZ)
C WVL=WAVELENGTH(M)
C NPROC=NUMBER OF PULSES FOR COHERENT INTEGRATION
C XNHT=NUMBER OF HITS ON TARGET
C BWA=ONE-WAY HALF-POWER AZIMUTHAL BEAMWIDTH(DEG)
C BWE=ONE-WAY HALF-POWER ELEVATION BEAMWIDTH

C
C COMMON /RANDOM/ SEED
COMMON PI, TOWPI, DR, RD
INTEGER DAY, HRS, MIN, RNUMBR, HISTORY
REAL*8 TIME
REAL MAXTIM

C
C COMPUTE THE MAXIMUM NUMBER OF SCANS
NSCANS=MAXTIM/PERIOD + .5
C COMPUTE FALSE-ALARM TIME(SECS)
TFA=XNHT*TOW/PFA
C COMPUTE THE NUMBER OF FALSE ALARMS/SCAN
NPPS=PERIOD/TFA + 1
C COMPUTE THE TOTAL OF FALSE ALARMS
NTOT=NPPS*NSCANS

C
DO 20 I = 1, NTOT
  X=RAN(SEED)
C COMPUTE A RANDOM TIME(DT) WITHIN THE INTERVAL TFA
  DT=TFA*X
C COMPUTE THE TRUE TIME FROM STARTING TIME OF 0.0 SECS
  TIME=DBLE(FLOAT(I)*TFA-DT)

79
IF(TIME.GT.MAXTIM)GO TO 20
C COMPUTE THE ANGLE THAT THE RADAR ANTENNA HAS SWEPT THROUGH
C AT THE TIME(TIME)
NT=TIME/PERIOD
AF=THETA0+TIME*360./PERIOD-FLOAT(NT)*360.
C RANDOMIZE THE RANGE
X=RAN(SEED)
RF = X * RLIM
C RANDOMIZE THE ELEVATION ANGLE
X=RAN(SEED)
EF = X * 90.
C RANDOMIZE THE DOPPLER FREQUENCY
X=RAN(SEED)
FDOP=PRF*(.5-X)
C COMPUTE A RELATIVE VELOCITY
V=FDOP*WVL/2.
C QUANTIZE THE DOPPLER FREQUENCY SHIFT
CALL QUANTF(V,WVL,PRF,NPROC,FDOP)
C ADD ANGULAR QUANTIZATION ERRORS
CALL QUANTA(EWE,XNHT,AF,NPROC)
IF(AF.GT.360.)AF=AF-360.
IF(AF.LT.0.)AF=360.+AF
CALL QUANTE(EWE,EF)
IF(EF.GT.90.)EF=90.-(EF-90.)
C CONVERT TIME INTO DAYS, HOURS, MINS AND SECS.
CALL TIMECON( TIME, DAY, HRS, MIN, SEC )
C REPORT THE RESULTS
WRITE(HISTORY,100) RNUMBR, DAY, HRS, MIN, SBC, RF, AF, EF, FDOP, V
100 FORMAT(I3,2X,I3,2(':':I2),'::',F6.3,1X,'FA',2(1X,F8.1),
X F7.2,2F10.1)
20 CONTINUE
C
RETURN
END

SUBROUTINE ST2(A,S2)
C SUBROUTINE TO COMPUTE THE ELEMENTS OF A 3 BY 3 ORTHOGONAL
C MATRIX ABOUT THE X2 AXIS
C B. J. ROOK APRIL 1983
DIMENSION S2(3,3)
S2(1,1)=COS(A)
S2(1,2)=0.
S2(1,3)=-SIN(A)
S2(2,1)=0.
S2(2,2)=1.
S2(2,3)=0.
S2(3,1)=-S2(1,3)
S2(3,2)=0.
S2(3,3)=S2(1,1)
RETURN
END

C==================================

80
SUBROUTINE ST3(A,S3)
C SUBROUTINE TO COMPUTE THE ELEMENTS OF 3 BY 3 ORTHOGONAL
C MATRIX ABOUT THE X3 AXIS
C B. J. ROOK APRIL 1983
DIMENSION S3(3,3)
S3(1,1)=COS(A)
S3(1,2)=SIN(A)
S3(1,3)=0.
S3(2,1)=-S3(1,2)
S3(2,2)=S3(1,1)
S3(2,3)=0.
S3(3,1)=0.
S3(3,2)=0.
S3(3,3)=1.
RETURN
END

SUBROUTINE COEFFF(SIGMAB,A1,A2,A3,A4)
C THESE COEFFICIENTS ARE USED FOR CALCULATING THE TARGET
C CROSS-SECTION FOR A GIVEN ASPECT ANGLE.
C B. J. ROOK JULY 1983
C INPUT PARAMETERS ARE:
C SIGMAB=TARGET CROSS-SECTION AT BROADSIDE(MAX. VALUE)(M**2)
C OUTPUT PARAMETERS ARE:
C A1,A2,A3,A4=COMPUTED COEFFICIENTS
C
C COMPUTE THE MINIMUM TARGET CROSS-SECTION
C THIS MINIMUM OCCURS AT 60DEG. FROM BROADSIDE
SIGM=0.03*SIGMAB
C COMPUTE THE END-ON TARGET CROSS-SECTION
SIGN=0.192*SIGMAB
C SEE HOVANESSIAN S. A. RADAR DETECTION & TRACKING SYSTEMS
C P3-7, FIG.3-6 IN DETERMINING THE CONSTANTS 0.03 & 0.192
C COMPUTE A4,A3,A2,A1
A4=(SIGMAB+(2.*SIGM))/3.
A3=(3.*SIGN-A4-2.*SIGM)/12
A2=(8.*A3+3.*A4-2.*SIGN-SIGN)/4.
A1=2.*A2-4.*A3
RETURN
END

SUBROUTINE ASPECT(АЗIMTH,THED,A1,A2,A3,A4,SIGMA)
C SUBROUTINE TO CALCULATE THE TARGET CROSS-SECTION FOR THE
C COMPUTED ASPECT ANGLE(TH)
C THIS CALCULATION IS OF THE FORM:
C THE ANGLE(TH) IS THE ANGLE BETWEEN A LINE PERPENDICULAR TO
C THE TARGET HEADING AND THE LINE-OF-SIGHT FROM RADAR TO TARGET.
C B. J. ROOK JULY 1983
C
C INPUT PARAMETERS ARE:
AZIMUTH=AZIMUTH ANGLE OF RADAR TO TARGET REL. TO TRUE
NORTH(DEG)
THED=TARGET HEADING REL. TO TRUE NORTH(RADS)
A1,A2,A3,A4=COEFFICIENTS FOR COMPUTING TARGET
CROSS-SECTION
OUTPUT PARAMETERS ARE:
SIGMA=COMPUTED TARGET CROSS-SECTION(M**2)

DATA PI/3.14159265/
CONVERT AZIMUTH TO RADIANS
AZ=AZIMUTH/57.29578
COMPUTE ASPECT ANGLE(TH)
TH=PI/2.-(AZ-THED)
COMPUTE SIGMA
RETURN
END

SUBROUTINE QUANTF(RELVEL,WVL,PRF,NPROC,FDO)
SUBROUTINE TO QUANTIFY THE DOPPLER FREQUENCY SHIFT TO FALL
WITHIN THE PROCESSING DOPPLER RANGE OF+/- (PRF/2)
B. J. ROOK JULY 1983
INPUT PARAMETERS ARE:
RELVEL=RADIAL VELOCITY BETWEEN RADAR AND TARGET(M/S)
PRF=PULSE REPETITION FREQUENCY(HZ)
WVL=RADAR WAVELENGTH(M)
NPROC=NUMBER OF PULSES FOR COHERENT INTEGRATION
OUTPUT PARAMETERS ARE:
FDO=QUANTIZED DOPPLER FREQUENCY SHIFT(HZ)

CONVERT RADIAL VELOCITY TO DOPPLER FREQUENCY SHIFT
FDO=2.*RELVEL/WVL
COMPUTE THE DOPPLER BIN WIDTH
DF=PRF/FLOAT(NPROC)
IF(FDO<0.)GO TO 3
QUANTIZE TO THE NEAREST DOPPLER BIN
N=FDO*2./DF + 0.5
COMPUTE THE QUANTIZED DOPPLER FREQUENCY
FDO=FLOAT(N)*DF/2.
IF THIS DOPPLER IS LESS THAN PRF/2 --NO AMBIGUITIES
RETURN
FOR DOPPLERS GREATER THAN PRF/2, COMPUTE THE FOLDED DOPPLER
NN=FDO*2./PRF
FDO=-PRF/2.+(FDO-FLOAT(NN)*PRF/2.)
RETURN
SAME PROCEDURE EXCEPT FOR NEGATIVE DOPPLER SHIFTS
N=FDO*2./DF - 0.5
FDO=FLOAT(N)*DF/2.
IF(FDO<0.)RETURN
NN=FDO*2./PRF
FDO=PRF/2.-FLOAT(NN)*PRF/2.-FDO
RETURN
END
SUBROUTINE QUANTA(B,XNHT,A,NPROC)
C SUBROUTINE TO COMPUTE AND ADD THE QUANTIZED ANGULAR ERROR TO
C THE AZIMUTHAL ANGULAR MEASUREMENT
C B. J. ROOK JULY 1983
C INPUT PARAMETERS ARE:
C B=ANTENNA ONE-WAY HALF-POWER BEAMWIDTH(DEG)
C XNHT=NUMBER OF HITS ON TARGET OVER THE TWO-WAY
C 1/E-BEAMWIDTH
C A=ANGULAR MEASUREMENT(DEG)
C OUTPUT PARAMETERS ARE:
C A=QUANTIZED ANGULAR MEASUREMENT(DEG)
C COMPUTE THE ANGULAR BIN WIDTH OVER THE COHERENT PROCESSING
C INTERVAL COMPRISING OF NPROC PULSES
DTH=0.85*B/XNHT*FLOAT(NPROC)
C COMPUTE THE NUMBER OF ANGULAR INCREMENTS TO THE ANGLE-A
NA=A*2./DTH + .5
C COMPUTE THE QUANTIZED ANGULAR VALUE
A=FLOAT(NA)*DTH*.5
RETURN
END

SUBROUTINE QUANTR(R,T)
C SUBROUTINE TO COMPUTE AND ADD THE QUANTIZED RANGE ERROR
C B. J. ROOK JULY 1983
C INPUT PARAMETERS ARE:
C R=MEASURED RANGE TO TARGET FROM RADAR(M)
C T=PULSE WIDTH(SECS)
C OUTPUT PARAMETERS ARE:
C R=QUANTIZED RANGE TO TARGET FROM RADAR(M)
C DATA C/3.88/
C COMPUTE RANGE BIN FROM PULSE WIDTH
DR=C*T/2.
C COMPUTE THE NUMBER OF RANGE CELLS
NR=R*2./DR+.5
C COMPUTE THE QUANTIZED RANGE FROM RADAR TO TARGET
R=NR*DR/2.
RETURN
END

SUBROUTINE QUANTE(BWE,E)
C SUBROUTINE TO COMPUTE AND ADD THE QUANTIZED ANGULAR ERROR TO
C THE ELEVATION ANGULAR MEASUREMENT
C B. J. ROOK JULY 1983
C INPUT PARAMETERS ARE:
C BWE=ONE-WAY HALF-POWER BEAMWIDTH(DEG)
C E=ELEVATION ANGULAR MEASUREMENT(DEG)
C OUTPUT PARAMETERS ARE:
C E=QUANTIZED ELEVATION ANGLE(DEG)
   IF(E.LT.0.)GO TO 2
C COMPUTE THE NUMBER OF ELEVATION ANGULAR BINS
C THE ANGULAR BIN SIZE IS TAKEN TO BE THE 1/E-BEAMWIDTH
   NE=2.*E/(.85*BWE) + 0.5
C COMPUTE THE QUANTIZED ELEVATION ANGLE
   E=NE*BWE*.85/2.
   RETURN
C SAME PROCEDURE EXCEPT FOR NEGATIVE ELEVATION ANGLES
2   NE=2.*E/(.85*BWE) - 0.5
   E=NE*BWE*.85/2.
   RETURN
END

SUBROUTINE POISON(MEAN,NUMBER)
C SUBROUTINE TO COMPUTE A RANDOM INTEGER FROM A POISON
DISTRIBUTION
C GIVEN A MEAN NUMBER
C B. J. ROOK JULY 1983
C
C INPUT PARAMETERS ARE:
C
C MEAN=MEAN NUMBER
C OUTPUT PARAMETERS ARE:
C NUMBER=RANDOM INTEGER
C ALGORITHM IS THAT OF HAHN & SHAPIRO, "STATISTICAL MODELS
C IN ENGINEERING", 1967, TABLE 7-2
C
COMMON /RANDOM/ SEED
REAL MEAN
   SUM=0.
   DO 1 I = 1, 5000
      SUM=SUM-ALOG(1.-RAN(SEED))
      IF(SUM.GT.MEAN)GO TO 2
1   CONTINUE
2   NUMBER=I
   RETURN
END

C *****ADDED BY JT************MATRIX MULTIPLICATION
SUBROUTINE VMULFF(R1,R2,I1,I2,J1,J2,K1,K2,IERRO)
REAL R1(I1,I2),R2(J1,J2),RR(K1,K2)
   DO 20 I=1,3
      DO 20 J=1,3
         SUM=0.0
         DO 10 K=1,3
            SUM=SUM+R1(I,K)*R2(K,J)
            RR(I,J)=SUM
   10   CONTINUE
   IERRO=0
   RETURN
END
REAL FUNCTION RAN(ISEED)  
INTEGER*4 IFF,M1,M2,M3,IA1,IA2,IA3  
INTEGER*4 IC1,IC2,IC3,IX1,IX2,IX3,ISEED  
REAL R(97)  
DATA M1/259200/,IA1/7141/,IC1/54773/  
DATA M2/134456/,IA2/8121/,IC2/28411/  
DATA M3/243000/,IA3/4561/,IC3/51349/  
DATA IFF/0/  
RM1=1.0/M1  
RM2=1.0/M2  
IF (ISEED .LT. 0 .OR. IFF .EQ. 0) THEN  
  IFF=1  
  IX1=MOD(IC1-ISEED,M1)  
  IX1=MOD(IA1*IX1+IC1,M1)  
  IX2=MOD(IX1,M2)  
  IX1=MOD(IA1*IX1+IC1,M1)  
  IX3=MOD(IX1,M3)  
  DO 10 J=1,97  
    IX1=MOD(IA1*IX1+IC1,M1)  
    IX2=MOD(IA2*IX2+IC2,M2)  
    R(J)=(FLOAT(IX1)+FLOAT(IX2)*RM2)*RM1  
  CONTINUE  
  ISEED=1  
END IF  
IX1=MOD(IA1*IX1+IC1,M1)  
IX2=MOD(IA2*IX2+IC2,M2)  
IX3=MOD(IA3*IX3+IC3,M3)  
J=1+(97*IX3)/M3  
IF (J .GT. 97 .OR. J .LT. 1) THEN  
  WRITE(*,*)'ERROR IN RANDOM NUMBER GENERATOR.'  
  STOP  
ENDIF  
RAN=R(J)  
R(J)=(FLOAT(IX1)+FLOAT(IX2)*RM2)*RM1  
RETURN  
END
APPENDIX 2C

Program Listing for the file conversion
program, SIMTRC2.C.

/*Program to convert data files from DREO simulator to format
used by the LRD site tracking program. This performs
the chronological sort and the conversion of range, azimuth,
elevation to latitude, longitude, height. */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <float.h>
#include <math.h>
#include <alloc.h>
#include <string.h>

#define MAXRAD 10
#define MAXSIZE 5000
#define MAXTARG 10

typedef float far *ptosinfo;
typedef int far *ptoinfo;
typedef char *string;
typedef string *radfiles;
typedef char far *farstr;
typedef farstr far *ptoidinfo;
typedef FILE *fptr;

void intogeo(float range,float az,float elev,float longr,
float latr,float hr,float *lng,float *lat,float *h);
void sort(float ss[],int orde[],int num);
void cross(float ro[],float ri1[],float ri2[]);

const float PI=3.141593;

int main()
{
    float posx[MAXRAD],posy[MAXRAD],rhgt[MAXRAD],likliid[MAXTARG];
    ptoinfo orde,radar,day,hrs,min;
    ptosinfo ss,sec,rng,az,elv;
    ptoidinfo id;
    float lat,lng,h,longr,lar,t,m TEMP,secr,fdop,vrel;
    float targnums,hr;
    fptr fparm,fin,ftrad,fout[MAXRAD];
    radfiles foutname;
    char foutpre[9],fradname[257],finname[257];
    char fplannme[257],junk[257];
    long xnum;
    int idmax,idval,ercode,radnum,radn,i,j,k,num,targnum;
    char q,suf[4],str30[31];
    int neg,newrad;
randomize();

/* begin by reading parameter file */
fp parm = fopen("simtrakd.dat","rt");
fgets(junk,257,fp parm); /* read a line */

i=0;
do /* Get the flight plan file name. */
{
    q=fgetc(fp parm);
    if (q != ' ')
    {
        fl planname[i]=q;
        i++;
    }
}
while (q != ' ');
fl planname[i]=\0;
fgets(junk,257,fp parm); /* move to next line */

i=0;
do /* Get the radar parameter file name. */
{
    q=fgetc(fp parm);
    if (q != ' ')
    {
        fr adname[i]=q;
        i++;
    }
}
while (q != ' ');
fr adname[i]=\0;
fgets(junk,257,fp parm); /* move to next line */
fgets(junk,257,fp parm); /* skip a line */

i=0;
do /* Get name of input file. */
{
    q=fgetc(fp parm);
    if (q != ' ')
    {
        fin name[i]=q;
        i++;
    }
}
while (q != ' ');
fin name[i]=\0;
fgets(junk,257,fp parm);

i=0;
do /* Get name of output file. */
{
    q=fgetc(fp parm);
    if (q != ' ')
    {

foutpre[i]=q;
i++;
}

while (q != ' ');
foutpre[i]='\0';
fclose(fparm);

frad=fopen(fradname,"rt");
 fscanf(frad,"%d",&radnum);    /* Get number of radars. */
fgets(junk,257,frad);
for (i=0;i<radnum;i++)       /* Get radar positions. */
{
    fscanf(frad,"%f:%f:%f",posx+i,&minr,&secr);
    fgets(junk,257,frad);
    if (posx[i] >= 0.0)
        posx[i]+=(minr/60.0+secr/3600.0);
    else
        posx[i]=-(minr/60.0+secr/3600.0);
    fscanf(frad,"%f:%f:%f",posy+i,&minr,&secr);
    fgets(junk,257,frad);
    if (posy[i] >= 0.0)
        posy[i]+=(minr/60.0+secr/3600.0);
    else
        posy[i]=-(minr/60.0+secr/3600.0);
    fscanf(frad,"%f",rhgt+i);    /* Get radar height. */
    fgets(junk,257,frad);
}

fclose(frad);
fparm=fopen(fplanname,"rt");
for (i=0;i<4;i++)
    {fgets(junk,257,fparm);}
fgets(str30,30,fparm);
 fscanf(fparm,"%f",&targnums); /* Get the number of targets. */
fgets(junk,257,fparm);
targnum=(int) (targnums+0.5);
for (i=0;i<12;i++)
    {fgets(junk,257,fparm);}

i=0;                            /* Get the liklihoods for each target. */

    {fscanf(fparm,"%c%s",&q,str30);
    if (q != ' ')
        {
            fgets(junk,257,fparm);
            suf[0]=q;
            suf[1]=str30[0]; /* Get first two characters. */
        }
suf[2] = '\0';          /* This is the target id. */
idval = atoi(suf);
fscanf(fp, "%f %f", &tmp, &likliid[idval]);
i++;
}
fgets(junk, 257, fp);
}
while (i < targnum);
fclose(fp);

xnum = MAXSIZE * sizeof(int);
if (farcoreleft() < 5 * xnum)
{
    printf("Insufficient memory in SIMTRC2.\n");
    exit(1);
}
orde = farmalloc(xnum);
if (orde == NULL)
{
    printf("Memory allocation error in SIMTRC2.\n");
    exit(1);
}
radar = farmalloc(xnum);
if (radar == NULL)
{
    printf("Memory allocation error in SIMTRC2.\n");
    exit(1);
}
day = farmalloc(xnum);
if (day == NULL)
{
    printf("Memory allocation error in SIMTRC2.\n");
    exit(1);
}
hrs = farmalloc(xnum);
if (hrs == NULL)
{
    printf("Memory allocation error in SIMTRC2.\n");
    exit(1);
}
min = farmalloc(xnum);
if (min == NULL)
{
    printf("Memory allocation error in SIMTRC2.\n");
    exit(1);
}
xnum = (MAXSIZE) * sizeof(float);
if (farcoreleft() < 5 * xnum)
{
    printf("Insufficient memory in SIMTRC2.\n");
    exit(1);
}
ss = farmalloc(xnum);
if (ss == NULL)
{  
    printf("Memory allocation error in SIMTRC2.\n");  
    exit(1);  
}

sec=farmalloc(xnum);  
if (sec == NULL)  
{  
    printf("Memory allocation error in SIMTRC2.\n");  
    exit(1);  
}

rng=farmalloc(xnum);  
if (rng == NULL)  
{  
    printf("Memory allocation error in SIMTRC2.\n");  
    exit(1);  
}

az=farmalloc(xnum);  
if (az == NULL)  
{  
    printf("Memory allocation error in SIMTRC2.\n");  
    exit(1);  
}

elv=farmalloc(xnum);  
if (elv == NULL)  
{  
    printf("Memory allocation error in SIMTRC2.\n");  
    exit(1);  
}

xnum=MAXSIZE*sizeof(farstr);  
if (farcoreleft() < xnum)  
{  
    printf("Insufficient memory in SIMTRC2.\n");  
    exit(1);  
}

id=(string far *) farmalloc(xnum);  
for (i=0; i<MAXSIZE; i++)  
    id[i]=(char far *) farmalloc(3);  

xnum=radnum*sizeof(string);  
foutname=(string *) malloc(xnum);  
for (j=0; j<radnum; j++)  /* Prepare output file names. */  
{  
    itoa(j,suf,10);  
    foutname[j]=(char *) malloc(257);  
    strcpy(foutname[j],foutpre);  
    strcat(foutname[j],".");  
    strcat(foutname[j],suf);  
    fout[j]=fopen(foutname[j],"wt");  
    if (fout[j] == NULL)  
    {  
        printf("File open error in SIMTRC2 on file #\%2d...\n",j);  
        for (k=0; k<j; k++)  
            fclose(fout[k]);  
        exit(1);  
    }
```c
    }
    fin=fopen(finname,"rt");
    newrad=0;
    for (j=0;j<radnum;j++)
    {
        printf("Processing data from radar # %02d\n",j);
        i=0;
        if (newrad == 1)
        {
            newrad=0;
            radar[0]=radar[num];
            radn=radar[0];
            day[0]=day[num];
            hrs[0]=hrs[num];
            min[0]=min[num];
            sec[0]=sec[num];
            strcpy(id[0],id[num]);
            elv[0]=elv[num];
            az[0]=az[num];
            rng[0]=rng[num];
            i=1;
        }
        do
        {
            if (i >= MAXSIZE)
            {
                printf("Program arrays too small in SIMTRC2.\n");
                fclose(fin);
                exit(1);
            }
            ercode=fscanf(fin,"%d",radar+i);
            if (ercode == EOF) break;
                /* exit from do {...} while loop */
            if (i == 0)
                radn=radar[i];
            else
                if (radar[i] != radn)
                newrad=1;
                fscanf(fin,"%d:%d:%d:%f",day+i,hrs+i,min+i,sec+i);
                q=fgetc(fin);
                for (k=0;k<2;k++)
                    id[i][k]=fgetc(fin);
                id[i][2]="\0";
                ercode=fscanf(fin,"%f %f %f %f %f %f", rng+i,az+i,elv+i,fdop,&vrel);
                fgets(junk,257,fin);
                i++;
        }
        while ((ercode != EOF) && (i < MAXSIZE) && (newrad == 0));
        num=i;
        printf("Number of detections %d\n",num);
    }
    if ((num >= MAXSIZE) && (ercode != EOF))
    {
```
printf("Data file too long to process in SIMTRC2.\n");
fclose(fin);
for (k=j;k<radnum;k++)
  fclose(fout[k]);
exit(1);
}
if (newrad == 0)
fclose(fin);
else
  num--;
for (i=0;i<num;i++)
{
  ss[i]=sec[i]+60.0*(min[i]+60.0*(hrs[i]+24.0*day[i]));
orde[i]=i;
}
printf("Sorting data...\n",13);
sort(ss,orde,num);
printf("\n",13);
rewind(fout[j]);
for (i=0;i<num;i++)
{
  printf("%d \n",num-i,13);
k=orde[i];
  togeo(rng[k],az[k],elv[k],posx[j],posy[j],
       rhgt[j],&lng,&lat,&h);
  idval=atoi(id[k]);
temp=((float)(rand()))/((float)(RAND_MAX));
  if ((idval > 0) && (temp <= likliid[idval]))
    fprintf(fout[j],"I %10.2f %10.6f %10.6f %10.2f %s\n",
          ss[k],lng,lat,h,id[k]);
  else
    fprintf(fout[j],"U %10.2f %10.6f %10.6f %10.2f\n",
          ss[k],lng,lat,h);
}
close(fout[j]);
}
farfree(orde);
farfree(radar);
farfree(day);
farfree(hrs);
farfree(min);
farfree(ss);
farfree(sec);
farfree(rng);
farfree(az);
farfree(elv);
for (i=0;i<MAXSIZE;i++)
  farfree(id[i]);
farfree(id);
for (i=0;i<radnum;i++)
  free(foutname[i]);
free(foutname);
return 0;
}
void intogeo(float range, float az, float elev, float longr, float latr, float hr, float *lng, float *lat, float *h) {
    float re=6378.0e3;
    float rr[3],rt[3],north[3],east[3],norm[3],tmp[3];
    double reff,sn,cs,radius;
    double dtemp1,dtemp2;

    reff=re+hr;
    dtemp1=(double) (PI/180.0*latr);
    dtemp2=(double) (PI/180.0*longr);
    rr[0]=(float) (cos(dtemp1)*cos(dtemp2));
    rr[1]=(float) (cos(dtemp1)*sin(dtemp2));
    rr[2]=(float) (sin(dtemp1));
    north[0]=-sin(dtemp1)*cos(dtemp2);
    north[1]=-sin(dtemp1)*sin(dtemp2);
    north[2]=cos(dtemp1);
    cross(east,north,rr);
    dtemp1=(double) (PI/180.0*az);
    tmp[0]=cos(dtemp1)*north[0]+sin(dtemp1)*east[0];
    tmp[1]=cos(dtemp1)*north[1]+sin(dtemp1)*east[1];
    cross(norm,rr,tmp); /* 2nd basis vector for great circle */
    /* start calculation for great circle angle */
    dtemp2=(double) (PI/180.0*elev);
    dtemp1=pow((double)range,2.0);
    dtemp1+=pow((double)reff,2.0);
    dtemp1+=2.0*range*reff*sin(dtemp2);
    radius=sqrt(dtemp1);
    sn=range*(float)(cos(dtemp2))/radius;
    if (abs(sn*sn) > 1.0e-6) {
        dtemp1=1.0.sn*sn;
        cs=sqrt(dtemp1);
        dtemp1=sn/cs;
        /* omega=(float) atan(dtemp1); */
    }
    else {
        omega=PI/2.0; /*
        cs=0.0;
        */
        printf("Omega %8.3f\n",180.0*omega/PI); /*
        rt[0]=cs*rr[0]+sn*tmp[0];
        dtemp2=(double)rt[2];
        dtemp2=pow(dtemp2,2.0);
        dtemp2=1.0-dtemp2;
        dtemp1=(double)(rt[2])/sqrt(dtemp2);
        *lat=atan(dtemp1);
        dtemp1=(double)(rt[1]/rt[0]);
        *lng=atan(dtemp1);
*h = radius-re;
}

void cross(ro, ri1, ri2)
float ro[], ri1[], ri2[];
{
    ro[0] = ri1[1]*ri2[2] - ri1[2]*ri2[1];
    ro[1] = ri1[2]*ri2[0] - ri1[0]*ri2[2];
    ro[2] = ri1[0]*ri2[1] - ri1[1]*ri2[0];
}

void sort(ss, orde, num)
ptosinfo ss;
ptoinfo orde;
int num;
{
    int temp, i, j, m, n;
    for (i = 0; i < num-1; i++)
        for (j = i+1; j < num; j++)
        {
            m = orde[i];
            n = orde[j];
            if ((n >= num) || (m >= num))
                printf("Array bounds error...\n");
            if (ss[n] < ss[m])
                {
                    temp = orde[i];
                    orde[i] = orde[j];
                    orde[j] = temp;
                }
        }
}
APPENDIX 2D

Program Listing for the site tracker, SIMTRC3.C.

/* Tracking program to perform tracking at remote radars */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <float.h>
#include <math.h>
#include <alloc.h>
#include <string.h>
#include <ctype.h>

extern _stklen=65000;

#define MAXRAD 10
#define L 100

typedef FILE *fptr;

struct target
{
    char message;
    float time,lng,lat,alt;
    int id,assoc,partner;
};

struct track
{
    float time,alpha,beta;
    float lats,lngs,zs,vlats,vlngs,vzs,vlato,vlngo,vzo;
    float latp,lngp,zp;
    float varx,vary,index;
    int id,observed,omissed;
};

void getscan(fptr f1, int *tsc, int *loaded, struct target observe[],float *scantime, float *stime);
void associate(struct target oldscan[], struct target newscan[],
int nold,int nnew, float gmax, float gmin, float scantime);
void update(struct track predict[], int *ntrack, struct target
oldscan[],struct target newscan[], int nnew, float scantime);

const float PI=3.141593;
const float mach1=331.0;  /* Mach 1 */
const float deg=180.0/3.141593;
const float rearth=6378000.0;  /* Radius of the earth. */
const float a=350.0;       /* Gate constant for old tracks. */
const int nd=5;            /* Number to deletion. */

int main()
{

}
char qq[257], radfile[257], infilpre[257];
char trakpre[257], trakout[257], infile[257];
char q, ext[4];
float radrot[MAXRAD];
float scantime, stime, range, azmth, elv, gmax, gmin;
float xmin, xmax, ymin, ymax, delx, dely, xtest, ytest;
fptr trak, fparm, f1;
int radar, radnum, i, j, scount, loaded;
nint nnew, ntrack;
int x, y;
struct target oldscan[L], newscan[L];
struct track predict[L];
unsigned char color;

fparm=fopen("SIMTRAKD.DAT","rt"); /* Get parameters.*/
if (fparm==NULL)
{
    printf("File open error in SIMTRC3.\n");
    exit(1);
}

fgets(qq, 257, fparm);
fgets(qq, 257, fparm);
i=0;
do
{
q=fgetc(fparm);
if (q != ' ')
{
    radfile[i]=q;
    i++;
}
} while (q != ' ');
radfile[i]='\0';
fgets(qq, 257, fparm); /* Move to next line. */
fgets(qq, 257, fparm); /* Move to next line. */
fgets(qq, 257, fparm); /* Move to next line. */
i=0;
do
{
q=fgetc(fparm);
if (q != ' ')
{
    infilpre[i]=q;
    i++;
}
} while (q != ' ');
infilpre[i]='\0';
fgets(qq, 257, fparm); /* Move to next line. */
q=fgetc(fparm);
if (q != ' ')
{
    /* Get output file prefix. */
    trakpre[i]=q;
    i++;
}
}
while (q != ' ');
trakpre[i]='0';
fgets(qq,257,fparm);
/* Move to next line. */
fclose(fparm);
fp=fopen(radfile,"rt");
scanf(fparm,"%d",&radnum); /* Get number of radars. */
fgets(qq,257,fparm);
/* Move to next line. */
for (i=1;i<=4;i++)
    fgets(qq,257,fparm); /* Move to next line. */
scanf(fparm,"%f",radrot); /* Radar rotation rate in revs/min. */
fgets(qq,257,fparm);
/* Move to next line. */
for (j=1;j<=radnum-1;j++)
{
    for (i=1;i<=20;i++)
        fgets(qq,257,fparm); /* Move to next line. */
    scanf(fparm,"%f",radrot+j);
    fgets(qq,257,fparm); /* Move to next line. */
}
fclose(fparm);
for (radar=0;radar<radnum;radar++) /* Loop over radars. */
{
    printf("Determining tracks for radar # %2d\n",radar);
    itoa(radar,ext,10);
    strcpy(infile,infilpre);
    strcat(infile,"."");
    strcat(infile,ext); /* Constructing file input name. */
    f1=fopen(infile,"rt");
    if (f1==NULL)
    {
        printf("File open error in SIMTRC3.\n");
        exit(1);
    }
    strcat(trakout,trakpre);
    strcat(trakout,"."");
    strcat(trakout,ext); /* Constructing file output name. */
   trak=fopen(trakout,"w+t");
    if (trak==NULL)
    {
        printf("File open error in SIMTRC3.\n");
        fclose(f1);
        exit(1);
    }
    rewind(trak);
    /* Open-rewind-close initiates the output file. */
    fclose(trak);
    scantime=60.0/radrot[radar]; /* Time for one rotation. */
    stime=-scantime; /* Initial scan time is 0. */
gmax=3.0*mach1*scantime; /* maximum speed is mach 3 */
gmin=90.0*scantime;  /* minimum speed is 90m/s */
scount=0;
loaded=-1;
nnew=-1;
ntrack=0;
do    /* loop through scans and track targets */
{
    /* get new scan */
    getscan(f1,&nnew,&loaded,newscan,&scantime,&stime);
scount++;
    printf("%4d %c",scount,13);
    associate(oldscan,newscan,ntrack,nnew,gmax,gmin,scantime);
    update(predict,&ntrack,oldscan,newscan,nnew,scantime);
    for (i=0;i<ntrack;i++)
    {
        if (predict[i].observed >= nd)
        {
            trak=fopen(trakout,"at");
            if (trak==NULL)
            {
                printf("File open error in SIMTRC3.\n");
                fclose(f1);
                exit(1);
            }
        } /* Output predicted values at next scan. */
        /* Note that the tracker assumes constant velocity so the
         * predicted velocities are equal to the state vector estimates.
         * Therefore, the predicted positions are used along
         * with the state vector estimates of the velocities. */
        fprintf(trak,"%3d %5d %10.2f %3d %8.5f %8.5f
%8.5f %11.8f %11.8f %8.5f\n", radar,scount,
predict[i].time,predict[i].id,predict[i].latp,
predict[i].lngp,predict[i].zp,predict[i].vlats,
predict[i].vlngs,predict[i].vzs);
        fclose(trak);
    }
    while (feof(f1) == 0);
    fclose(f1);
    return 0;
}

void getscan(fpotr f1, int *tsc, int *loaded, struct target
observe[],float *scantime, float *stime)
{
    char mssge,blk,blk2[3],time2[3],day[3];
    char year[3],month[4],junk[257];
    float xtest,ytest,gmt,lng,lat,alt,theta;
    int id,sflag;
*tsc=0;
(*stime)+=(*scantime); /* Increment time. */
if ((*loaded >= 0) && (observe[*loaded].time < *stime))
{
  *tsc=1;
  observe[0]=observe[*loaded]; /* shift loaded observation */
  *loaded=-1; /* into current scan */
}
if ((*loaded < 0) ||
  (*loaded >= 0) && (observe[*loaded].time < *stime))
  && (feof(f1) == 0))
{
  do
  {
    mssge=fgetc(f1);
    mssge=toupper(mssge);
    switch (mssge)
    {
      case 'I': /* Target with an ID. */
        fscanf(f1,"%f %f %f %f %d",&gmt,&lng,&lat,&alt,&id);
        fgets(junk,256,f1);
        observe[*tsc].message=mssge;
        observe[*tsc].time=gmt;
        observe[*tsc].lng=lng;
        observe[*tsc].lat=lat;
        observe[*tsc].alt=alt;
        observe[*tsc].assoc=0;
        observe[*tsc].partner=0;
        observe[*tsc].id=id;
        (*tsc)++;
        break;
      case 'U': /* Target with no ID. */
        fscanf(f1,"%f %f %f %f",&gmt,&lng,&lat,&alt);
        fgets(junk,256,f1);
        observe[*tsc].message=mssge;
        observe[*tsc].time=gmt;
        observe[*tsc].lng=lng;
        observe[*tsc].lat=lat;
        observe[*tsc].alt=alt;
        observe[*tsc].assoc=0;
        observe[*tsc].partner=0;
        observe[*tsc].id=-1;
        (*tsc)++;
        break;
      default: /* Skip line with */
        fgets(junk,256,f1); /*improper target information.*/
    }
  }
while ((feof(f1) == 0) && (*tsc < L) && (gmt < *stime));
if (gmt > *stime)
{
  *loaded=*tsc-1; /*Save last observation for next scan.*/
  (*tsc)--;
}
if (*tsc > L)
{
    printf("Too many targets in this scan (SIMTRC3).\n");
    fclose(f1);
    exit(1);
}
}

void associate(struct target oldscan[], struct target newscan[],
      int nold, int nnew, float gmax, float gmin, float scantime)
{
    int i, j, k, ii, jj, jmin, itemp;
    int partner[L][L];
    int ptrack[L][L];
    int cmc[L], rmc[L];
    float assoc[L][L];
    int flag;
    float dlat, dlng, rmin, rmax, min, dist, dt;
    double temp1, temp2;

    for (i=0; i<nnew; i++)
        newscan[i].assoc = 0;
    for (i=0; i<nold; i++)   /* Calculate association matrix. */
    {
        oldscan[i].assoc = 0;
        for (j=0; j<nnew; j++)
        {
            dlat = newscan[j].lat - oldscan[i].lat;
            dlng = newscan[j].lng - oldscan[i].lng;
            temp1 = dlng * rearth * cos((double)oldscan[i].lat);
            temp2 = (double)(dlat * rearth);
            assoc[i][j] = (float) sqrt(pow(temp1, 2.0) + pow(temp2, 2.0));
        }
    }
    for (i=0; i<nold; i++)   /* Step 1. */
    if (oldscan[i].id >= 0)   /* Check ID. */
    {
        rmin = gmin;
        rmax = gmax;
        if (oldscan[i].message == '0')
        {
            rmin = 0.0;
            rmax = a * scantime;
        }
        for (j=0; j<nnew; j++)
        if (oldscan[i].id == newscan[j].id)   /* ID observed. */
        {
            /* Last scan. */
            dt = fabs(newscan[j].time - oldscan[i].time);
            if ((assoc[i][j] >= rmin) &&
                (assoc[i][j] <= rmax) &&
                (dt < scantime/2.0)) /* gates and */
            {
                oldscan[i].partner = j;
            }
oldscan[i].assoc=1;
newscan[j].assoc=1;
}

for (i=0; i<nold; i++)
{
    rmc[i]=0;
    /* Set gates, default is for a new track. */
    rmax=gmax;
    rmin=gmin;
    if (oldscan[i].message == 'O')
    {
        rmin=0.0; /* Old track. */
        rmax=a*scantime;
    }
    for (j=0; j<nnew; j++) /* Count potential matches in rows. */
    {
        dt=fabs(newscan[j].time-oldscan[i].time);
        if (((newscan[j].assoc == 0) && (assoc[i][j]>=rmin)
            && (assoc[i][j]<rmax) && (dt < scantime/2.0))
            
            { itemp=rmc[j];
                partner[i][itemp]=j;
                rmc[i]++;
            }
        }
    }
}

for (j=0; j<nnew; j++) /* Step 2. */
{
    cmc[j]=0;
    for (i=0; i<nold; i++)
    {
        dt=fabs(newscan[j].time-oldscan[i].time);
        rmax=gmax; /* Set gates. */
        rmin=gmin;
        if (oldscan[i].message == 'O')
        {
            rmin=0.0;
            rmax=a*scantime;
        }
        if (((!oldscan[i].assoc) && (assoc[i][j]<rmax)
            && (assoc[i][j]>=rmin) && (dt < scantime/2.0))
            
            { itemp=cmc[j];
                ptrack[j][itemp]=i;
                /* Count potential matches in columns. */
                cmc[j]++;
            }
        }
    }
}

for (i=0; i<nold; i++) /* Step 3. */
{
    /* Look for single matches. */
    if (((oldscan[i].assoc == 0) && (rmc[i]==1))
```c
{
    j=partner[i][0];
    if (((newscan[j].assoc == 0) && (cmc[j] >= 1))
    {
        oldscan[i].partner=j;
        oldscan[i].assoc=1;
        newscan[j].assoc=1;
    }

    if (((newscan[j].assoc == 0) && (cmc[j] > 1)) /* Step 4 */
    {
        flag=1;
        for (jj=0; jj<cmc[j]; jj++)
        {
            ii=ptrack[j][jj];
            if (ii != i)
            {
                if (flag && ((rmc[ii] > 1) || (rmc[ii] == 0)))
                flag=1;
                else
                flag=0;
            }
        }

    /* If tracks can be merged,
    this is where the associations should be done. */
    }

    if (flag) /* Step 5 */
    {
        oldscan[i].partner=j;
        oldscan[i].assoc=1;
        newscan[j].assoc=1;
    }
}

/* If not merging tracks,
select one track out of the cmc[j] available. */
for (i=0; i<nold; i++)
{
    if (!oldscan[i].assoc) && (rmc[i] == 1))
    {
        j=partner[i][0];
        if (((!newscan[j].assoc) && (cmc[i] > 1))
        {
            min=gmax;
            if (oldscan[i].message == 'O') /* Set gates. */
                min=a*scantime;
            jmin=-1;  
            for (jj=0; jj<cmc[j]; jj++)
            {
                ii=ptrack[j][jj];
                if (((!oldscan[ii].assoc) && (assoc[ii][j] < min)
                    && (rmc[ii] == 1))
                {
                    /* Compare the single matches */
                    min=assoc[ii][j]; /* i.e. only the data that*/
                    jmin=ii; /* can't be matched */
```
if (jmin>=0)
{
    ii=jmin;
    oldscan[ii].partner=j;
    oldscan[ii].assoc=1;
    newscan[j].assoc=1;
}

for (i=0;i<nold;i++)
{
    rmax=gmax;
    if (oldscan[i].message == 'O') /* Set gates. */
        rmax=a*scantime;
    if (!oldscan[i].assoc && (rmc[i]>1)) /* Step 6. */
        { /* Find closest matches. */
            jmin=-1;
            min=rmax;
            for (ii=0;ii<rmc[i];ii++)
            {
                j=partner[i][ii];
                if (!newscan[j].assoc && (assoc[i][j]<min))
                    {
                        min=assoc[i][j];
                        jmin=j;
                    }
            }
            if (jmin>=0)
                {
                    oldscan[i].partner=jmin;
                    oldscan[i].assoc=1;
                    newscan[jmin].assoc=1;
                }
        }
}

void update(struct track predict[], int *ntrack, struct target oldscan[],
            struct target newscan[], int nnew, float scantime)
{

    int i,j,k;
    float dt,dlat,dlng,dz,dvlat,dvlnq,dvz,ind;
    double dtemp1;

    for (i=0;i<*ntrack;i++)
    {
        if (oldscan[i].assoc)
            /* Update confirmed, existing tracks. */
            {

j=oldscan[i].partner;
dlat=newscan[j].lat-oldscan[i].lat;
dlng=(newscan[j].lng-oldscan[i].lng)
    *(float) cos((double)oldscan[i].lat);
dt=newscan[j].time-oldscan[i].time;
dt+=scantime;
dz=newscan[j].alt-oldscan[i].alt;
predict[i].id=newscan[j].id; /*Use the most recent id.*/
if (predict[i].id<0)
    predict[i].id=oldscan[i].id;
dvlat = predict[i].vlats-predict[i].vlato;
dvlng = predict[i].vlngs-predict[i].vlngo;
    dvz = predict[i].vzo; /*
    /* Not currently used. */
}
j = predict[i].observed+1;

if (j<=1)
    {
    predict[i].varx=0.0;
    predict[i].varv=0.0;
    }
else /* Calculate variances for tracking index. */
    {
    predict[i].varx = (dlat*dlat+dlng*dlng+
        ((float)j-1.0)*predict[i].varx)/((float)j);
    predict[i].varv = ((dvlat*dvlat+dvlng*dvlng)/(dt*dt)
        +((float)j-1.0)*predict[i].varv)/((float)j);
    }
if (predict[i].varx < 1.0e-4)
    predict[i].index=1.0;
else /* Tracking index is calculated */
    {
    dtemp1=(double)(predict[i].varv/predict[i].varx);
    predict[i].index= dt*dt*(float)sqrt(dtemp1);
    }
if (predict[i].index < 1e-4) /* alpha = beta = 0 */
    {
    predict[i].alpha = 0.0;
    predict[i].beta = 0.0;
    }
else
    {
    ind = predict[i].index;
    if (predict[i].observed <= 2)
        {
        predict[i].alpha = 1.0;
        predict[i].beta = 1.0;
        }
    else /* Calculate coefficients */
        /* based on tracking index.*/
            {dtemp1=pow((double)ind,2.0);
            predict[i].alpha=(float)((-1.0*dtemp1+
                sqrt(dtemp1*dtemp1+16.0*dtemp1))/8.0);
            dtemp1=sqrt((double)(1.0-predict[i].alpha));
            predict[i].beta=(float)(4.0-
2.0*(double)predict[i].alpha-4.0*dtemp1);
}
}
predict[i].lats=predict[i].latp+predict[i].alpha*dlat;
predict[i].lngs=predict[i].lngp+predict[i].alpha*dlng;
predict[i].zs=predict[i].zp+dz;
predict[i].vlat0=predict[i].vlats;
predict[i].vln0=predict[i].vlngs;
predict[i].vzo=predict[i].vzs;
predict[i].vlats+=predict[i].beta*dlat/dt;
predict[i].vlngs+=predict[i].beta*dlng/dt;
predict[i].vzs+=dz/dt;
predict[i].latp=predict[i].lats+dt*predict[i].vlats;
predict[i].lngp=predict[i].lngs+dt*predict[i].vlngs;
predict[i].zp=predict[i].zs+dt*predict[i].vzs;
predict[i].observed++;
predict[i].omitted=0;
predict[i].time+=dt;
}
else
   /* Missed observations. */
predict[i].omitted++;
if (predict[i].omitted < nd)
   /* Predict new position. */
{
   dt=scantime;
predict[i].lats=predict[i].latp;
predict[i].lngs=predict[i].lngp;
predict[i].zs=predict[i].zp;
predict[i].vlato=predict[i].vlats;
predict[i].vln0=predict[i].vlngs;
predict[i].vzo=predict[i].vzs;
predict[i].latp=predict[i].lats+dt*predict[i].vlats;
predict[i].lngp=predict[i].lngs+dt*predict[i].vlngs;
predict[i].zp=predict[i].zs+dt*predict[i].vzs;
predict[i].time+=dt;
}
if (*ntrack > 0)
{
i=0;
j=0;
do
   if (predict[i+j].omitted >= nd) /* Delete bad tracks. */
   {
      do
         if ((i+j<*ntrack) && (predict[i+j].omitted >= nd))
            j++;
      while ((i+j<*ntrack) && (predict[i+j].omitted >= nd));
   }
   if ((i+j)<*ntrack)
      predict[i]=predict[i+j];
i++;
}
while (i+j < *ntrack);
*ntrack=*ntrack-j;
}
for (j=0;j<nnew;j++) /* Initiate new tracks. */
{
  if (!newscan[j].assoc)
  {
    predict[*ntrack].lats=newscan[j].lat;
predict[*ntrack].lats=newscan[j].lats[0];
predict[*ntrack].lats=newscan[j].lats[1];
predict[*ntrack].lats=newscan[j].lats[2];
predict[*ntrack].vlats=0.0;
predict[*ntrack].vlns=0.0;

  /* Initial velocities are zero. */
predict[*ntrack].vzs=0.0;
predict[*ntrack].vlato=0.0;
predict[*ntrack].vln.go=0.0;
predict[*ntrack].vzo=0.0;
predict[*ntrack].var=x=0.0;
predict[*ntrack].vary=0.0;
predict[*ntrack].alpha=1.0;

  /* Initial coefficients are 1.0 */
predict[*ntrack].beta=1.0;
predict[*ntrack].index=1.0;
predict[*ntrack].latp=newscan[j].lat;
predict[*ntrack].lngp=newscan[j].lng;
predict[*ntrack].zp=newscan[j].z.p;
predict[*ntrack].observed=1;
predict[*ntrack].omissed=0;
predict[*ntrack].id=newscan[j].id;
predict[*ntrack].time=newscan[j].time+scantime;
(*ntrack)++;
  if (*ntrack >= L)
  {
    printf("Out of memory in update procedure
(SIMTRC3).\n");
    exit(1);
  }
}
for (i=0;i<*ntrack;i++)
/* Load oldscan with predicted positions. */
{
  oldscan[i].lat=predict[i].lat;
  oldscan[i].lng=predict[i].lng;
  oldscan[i].alt=predict[i].z.p;
  oldscan[i].assoc=0;
  oldscan[i].id=predict[i].id;
  oldscan[i].time=predict[i].time;
  if (predict[i].observed < 2)
    oldscan[i].message = 'N'; /* Label scan as New. */
  else
    oldscan[i].message = 'O'; /* Label scan as Old. */
APPENDIX 3A

Program Listing for the System Tracker, SIMTRC4.C.

/* Program SIMTRC4.C is a system tracker and display unit.
The program is written in Borland Turbo C++ and requires
the huge memory model for proper compilation. A VGA card
is required to operate the program. This is a low
resolution version suitable for standard VGA. */

#include <alloc.h>
#include <conio.h>
#include <dos.h>
#include <fcntl.h>
#include <math.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

extern unsigned _stklen=65000;   /* Stack size. */
define maxrad 4   /* Maximum number of radars. */
define maxnum 25  /* Maximum number of tracks for each radar. */
/* Note that maxnum does not have to equal maxrad since there can
be a system queue for each target. Therefore maxnum is at most
maxrad*maxtarg. */
define maxtarg 25  /* Maximum number of targets. */
define bigdist 1000000 /* A big number. */
typedef FILE *fptr;
typedef float *row;
typedef row *matrix;
struct targdat {    /* Observed target data. */
    float time[maxnum],lat[maxnum],lng[maxnum],z[maxnum];
    float vlat[maxnum],vlng[maxnum],vz[maxnum];
    int id[maxnum],scan[maxnum];
};
struct systrack {   /* System track information. */
    float time,lat,lng,z,vlat,vlng,vz;
    int id,scan;
    int x1,y1,x2,y2;
};
struct probinfo {  /* Association probabilities. */
    float prb[maxrad][maxtarg][maxtarg][4];
    /* The last index is: [0] targets are associated
     [1] targets are not associated
     [2] uncertainty */
    float ptmp[maxtarg][maxtarg][4];
};
struct chronoref {   /* Time keeping information. */
    int rad,targ;
    float time;
};
const int screenx=319;   /* Maximum screen size x-direction. */
const int screeny=199;   /* Maximum screen size y-direction. */
const int mn=19;
const int kaln=6;   /* Array sizes for Kalman filter. */
const int kalm=4;
const float aspect=0.77;   /* Aspect ratio of screen. */
const float rearth=6378000.0; /* Radius of the earth in metres. */
const float degtorad=3.14159265/180.0;
  /* Degree to radian factor. */
const float maxvel=0.0002;
  /* Maximum angular velocity of targets. */

void putpix(unsigned char mag, int col, int row);
void setmode(unsigned char mode, unsigned char noclclear);
void defpal(unsigned char);
void setcolors(void);
void rectangle(unsigned char color, int x1, int y1, int x2, int y2);
void line(unsigned char color, int x1, int y1, int x2, int y2);
int round(float);
void putblock(unsigned char, int, int, int);
void outchar(unsigned char, char);
void outstring(unsigned char, char*);
void grid(unsigned int color, float griddist, float nsmax);
void plotsite(unsigned char color, float rlat[], float rlng[], int
  radnum, float latorig, float lngorig, float nsmax);
void convxy(float lat, float lng, float *x, float *y,
  float latorig, float lngorig);
void steady(double *t, float ttmp);
void getscan(fptr fin, int scnum, int *num,
  int *loaded, float *lttime, struct targdat *targ, float stime
  , float *rbias, float *azbias, float *ebias, float *rlat,
  float *rlng, float *rhgt, int vga);
void convbias(float *rbias, float *abias, float *ebias,
  float *lat, float *lng, float *zz, float *rlat, float *rlng,
  float *rhgt);
void cross(double ro[], double ri1[], double ri2[]);
void intogeo(double range, double az, double elev, double
  longr, double latr, double z, double *lng, double *lat, double *zn);
void disp(float latorig, float lngorig, float griddist,
  float nsmax, float tail, int headsize, unsigned char gridcolor,
  struct systrack systarg[], int sysnum, unsigned char idcolor,
  unsigned char noidcolor, fptr sysfile, int hist);
void gridre(unsigned char color, float griddist, float nsmax,
  int x1, int y1, int x2, int y2, int headsize);
void getprobs(int rad, int *num1, struct targdat *targ1,
  int *sysnum, struct systrack systarg[], float sdt[], float svd[],
  struct probinfo *prob, float height, float pinit[], int vga);
float sgr(float *x);
float gauss(float *dd, float sd);
void sort(struct chronoref queue[], int *glen);
void makenew(int radnum, struct probinfo *prob,
  struct targdat *targ[], int num[], struct systrack systarg[],
  int *sysnum, float pthresh, float puncthr, float sdt[],
  float svd[], int vga);
void deltrack(struct systrack systarg[], int *sysnum,
  float pdelthr, float griddist, float nsmax,
  unsigned char gridcolor, float stime, int headsize,
  struct probinfo *prob, float pinit[], int radnum,
  int vga, int hist);
void delcrt(float griddist, float nsmax, unsigned char gridcolor,
struct systrack starg, int headsize);
void getassoc(int *newnew, struct systrack *starg,
               struct targdat *tar, int j, float sdt, float svd, float pthresh,
               int vga);
void update(struct systrack *starg, struct chronoref *que,
            struct targdat targ[], float rbias, float azbiass, matrix p,
            matrix pi, matrix kk);
void kgain(int N, int M, matrix A, matrix C, matrix P, matrix P1,
           matrix Q, matrix R, matrix K);
void kal(int N, int M, matrix x, matrix y, matrix A, matrix C,
         matrix K, matrix b);
void mul(matrix A, matrix B, matrix C, int L, int M, int N);
void trn(matrix A, matrix B, int N, int M);
void add(matrix A, matrix B, int N, int M);
void inv(matrix A, int N);
matrix mtrn(matrix A, int N);
void frmtrx(matrix A, int N);

int main()
{
    char q;
    char message[100], junk[257], trakpre[257], fradname[257];
    char ext[257], infile[257], sysout[257];
    double tt;
    float latorig, lngorig, tail, timestep, nsmax, griddist;
    float pdelthr, puncthr, sdrv[maxrad];
    float rlat[maxrad], rlng[maxrad], rhgt[maxrad], rbias[maxrad];
    float azbias[maxrad], ebias[maxrad], minr, secr, ttmp, pthresh;
    float tmp, pinit[3], tinc, loadtime[maxrad], sdt[maxrad];
    fptr fprom, frad, fin[maxrad], sysfile;
    int i, j, k, scrum, loaded[maxrad], sysnum, qlen;
    matrix p, pi, kk;
    struct chronoref queue[maxrad+1];
    struct probinfo prob;
    struct systrack systarg[maxtarg];
    struct targdat targ[maxrad];
    unsigned char gridcolor, sitecolor, eofflag, loadflag;
    unsigned char idcolor, noidcolor;

    fprom=fopen("simtrakd.dat","rt");  /*Read parameter file.*/
    fgets(junk,257,fprom);
    fgets(junk,257,fprom);
    i=0;
    do
        {q=fgetc(fprom);
         if (q!='')
             {
                /*Get radar parameter file name.*/
                if (q!='')
                    {fradname[i]=q;
                     i++;}
             }
        while (q!='
        fradname[i]='\0';
    fgets(junk,257,fprom);
    for (i=1; i<=3; i++)
fgets(junk,257,fparm);
i=0;
do
  {q=fgetc(fparm);
   if (q!=')
     {trakpre[i]=q; /*Get site tracker output file name.*/
      i++;}
  }
while (q!='));
trakpre[i]='\0';
fgets(junk,257,fparm);
i=0;
do
  {q=fgetc(fparm);
   if (q!='
     {sysout[i]=q; /*Get system output file name.*/
      i++;}
  }
while (q!='
sysout[i]='\0';
fgets(junk,257,fparm);
fscanf(fparm, "%f", &nsmax);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &griddist);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &latorig);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &lingorig);
fgets(junk,257,fparm);
fscanf(fparm, "%d", &j);
gridcolor=(unsigned char) (j);
fgets(junk,257,fparm);
fscanf(fparm, "%d", &j);
sitecolor=(unsigned char) (j);
fgets(junk,257,fparm);
fscanf(fparm, "%d", &j);
idcolor=(unsigned char) (j);
fgets(junk,257,fparm);
fscanf(fparm, "%d", &j);
noidcolor=(unsigned char) (j);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &tail);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &tmp);
headsize=round(tmp);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &timestep);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &tinc);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &pthresh);
fgets(junk,257,fparm);
fscanf(fparm, "%f", &puncthr);
fgets(junk,257,fparm);
fscanf(fparm,"%f",&pdelthr);
fgets(junk,257,fparm);        /* Get initial probabilities. */
fscanf(fparm,"%f",&pinit[0]);
fgets(junk,257,fparm);
fscanf(fparm,"%f",&pinit[1]);
fgets(junk,257,fparm);
fscanf(fparm,"%f",&pinit[2]);
fgets(junk,257,fparm);
fscanf(fparm,"%d",&hist);
fgets(junk,257,fparm);
fscanf(fparm,"%d",&vga);
fgets(junk,257,fparm);
fgets(junk,257,fparm);
fgets(junk,257,fparm);
fgets(junk,257,fparm);
fscanf(fparm,"%f",&pinit[2]);
for (i=0; i<maxrad; i++)
{
    fgets(junk,257,fparm);
    fscanf(fparm,"%f %f %f %f",&rbias[i],&azbias[i],
    &ebias[i],&sdt[i],&sdv[i]);
    azbias[i]=azbias[i]*degtorad;
    ebias[i]=ebias[i]*degtorad;
    sdv[i]=sdv[i]*degtorad;
}
fclose(fparm);
latorig=latorig*degtorad;
lngorig=lngorig*degtorad;

frad=fopen(fradname,"rt");
fscanf(frad,"%d",&radnum);    /*Get number of radars.*/
fgets(junk,257,frad);
/* The following loop determines the latitude and longitude
   position of each radar. */
for (i=0;i<radnum;i++)
{fscanf(frad,"%f:%f:%f",rlng+i,&minr,&secr);
fgets(junk,257,frad);
if (rlng[i] >= 0.0)
    rlng[i]+=(minr/60.0+secr/3600.0);
else
    rlng[i]-=(minr/60.0+secr/3600.0);
fscanf(frad,"%f:%f:%f",rlat+i,&minr,&secr);
fgets(junk,257,frad);
if (rlat[i] >= 0.0)
    rlat[i]+=(minr/60.0+secr/3600.0);
else
    rlat[i]-=(minr/60.0+secr/3600.0);
fscanf(frad,"%f",rhgt+i);
fgets(junk,257,frad);
if (i < radnum)
{for (j=0;j<18;j++)
    fgets(junk,257,frad);}
}
fclose(fr); /* Close radar parameter file. */
for (i=0; i<radnum; i++)
    {rlat[i]=rlat[i]*degtorad; /* Convert radar positions to */
     llng[i]=llng[i]*degtorad;} /* degrees. */

sysfile=fopen(sysout,"wt"); /* File of system track information. */
if (sysfile==NULL)
    {
        printf("File open error in SIMTRC4. \n ");
        exit(1);
    }
rewind(sysfile);

if (vga>0)
    {
        setmode(mm,0); /* Change to graphics mode. */
        setcolors(); /* Set the colours. */
        grid(gridcolor,griddist,nsmax); /* Plot grid on screen. */
        /* Plot radar sites. */
        plotsite(sitecolor,rlat,llng,radnum,latorig,lngorig,nsmax);
        gotoxy(1,1);
        strcpy(message,"London Research and Development");
        outstring(63,message);
    }
else
    {
        printf("\n \n London Research and Development \n ");
    }
for (i=0; i<radnum; i++) /* Open track files. */
    {
        seta(i,ext,10);
        strcpy(infile,trakpre);
        strcat(infile,"."));
        strcat(infile,ext);
        fin[i]=fopen(infile,"rt");
    }

for (k=0; k<maxrad; k++)
    for (i=0; i<maxtarg; i++) /* Initialise probabilities. */
        for (j=0; j<maxtarg; j++)
            {
                prob.prb[k][i][j][0]=pinit[0];
                prob.prb[k][i][j][1]=pinit[1];
                prob.prb[k][i][j][2]=pinit[2];
                prob.prb[k][i][j][3]=0.0;
            }

for (j=0; j<maxtarg; j++)
    {
        systarg[j].time=-1.0;
        systarg[j].vlat=0.0;
        systarg[j].vlng=0.0;
        /* The following initialisation is required so that no erase */
        /* attempt is made before any system targets have been formed. */
        systarg[j].xl=screеннx+1;
    }
for (i=0; i<maxrad; i++)
    {loaded[i]=-1;  /* No scans loaded initially. */
     loadtime[i]=-1;
     num[i]=-1;}

p=mtrx(kaln,kaln);
p1=mtrx(kaln,kaln);
k_k=mtrx(kaln,kaln);
/* Define values of the matrices. Note p1 and k_k are determined by the program from p. */
for (i=0; i<kaln; i++)
    for (j=0; j<kaln; j++)
        {if (i==j)
         p[i][j]=1e-4;
         else
         p[i][j]=0.0;
        }

scnum=0;  /* Number of scans. */
ttmp=0.0; /* Start time. */
systnum=-1; /* Number of system tracks. */
/* Steady saves the initial time. Later calls make the time length between calls equal to the interval parameter in the parameter file. When the parameter is equal to the scan time the screen appears as it would in "real" time. */
steady(&tt,ttmp);

    do
        {scnum++;
         ttmp=tinc*scnum;
         if (vga>0)
            {
             gotoxy(34,1);
             itoa(scnum,message,10);
             /* Display scan count on graphics screen */
             outstring(63,message);
            }
         else
            {
             printf("Processing scan %d %.c",scnum,13);
            }
         for (i=0; i<radnum; i++)
             /* Get information from track files. */
            {
             getscan(fin[i],scnum,&num[i],&loaded[i],
                        &loadtime[i],&targ[i],
                        ttmp,&r_bias[i],&azbias[i],&ebias[i],&rlat[i],&rlng[i],
                        &rhgt[i],vga);
             if (num[i]>=maxnum)
                 {
                   if (vga>0)
                     {
                       gotoxy(1,5);
                       strcpy(message,"Too many targets in this scan! Press a

114
key.");
    outstring(63,message);
    q=getch();
    defpal(1);    /* Set colours back to normal. */
    setmode(3,0);

/* Switch to text mode and clear the screen. */
}
else
{
    printf("\n Too many targets in this scan! \n ");
}
for (i=0; i<radnum; i++)  /* Close track files. */
    {fclose(fin[i]);
    fclose(sysfile);
    frmtrx(p,kaln);
    frmtrx(pi,kaln);
    frmtrx(kk,kaln);
    exit(1);
}

/* The following uses the Dempster-Shafer method to associate the
 targets with system tracks. */
for (i=0; i<radnum; i++)
    {getprobs(i,&num[i],targ[i],&sysnum,systarg,sdt,sdv,
     &prob,rhgt[i],pinit,vga);
}

/* Get scan has got information from each radar. Getprobs has
determined the likelihood that each piece of data is associated
with a system track. The data can now be used to update the
system tracks, however, the update needs to be done in
chronological order. So targets seen by more than one radar must
be recognised and the time stamps taken into account. */
for (i=0; i<maxtarg; i++)
    /* Make queue for updating system track i. */
    {
        qlen=-1;
        for (j=0; j<radnum; j++)
            for (k=0; k<maxtarg; k++)
                {if ((prob[prb[j][k][i][0]>pthresh] &&
                    (prob[prb[j][k][i][3]>0.5))

/* Note the queue length has a maximum. Presently nothing is
done to indicate excessive numbers of associations to a system
track. This means that a flock of say five targets may register
as one system track. A change in target head size according to
the queue size would alleviate the problem. */
        {if(qlen<maxrad)
            {qlen=qlen+1;
             queue[qlen].rad=j;
             queue[qlen].targ=k;
             queue[qlen].time=targ[j].time[k];}
        else
            {
                if (vga>0)
{gotoxy(1,2);
    strcpy(message,"Queue length exceeded.");
    outstring(63,message);
}

else
{
    printf("\n Queue length exceeded.");
}

/* If the queue has more than one entry sort by time. */
if (qlen>0)
{sort(queue,&qlen);}

/* Update system track i using the queue. */
if (qlen>=0)
{
    for (j=0; j<=qlen; j++)
    {
        update(&systarg[i],&queue[j],targ,
            rbias[queue[j].rad],azbias[queue[j].rad],
            p,p1,kk);
    }
}

/* Start new system tracks. */
deltrack(systarg,&sysnum,pdelthr,griddist,nsmax,
    gridcolor,ttmp,headsize,&prob,
    pinit,radnum,vga,hist);
makenew(radnum,&prob,targ,num,systarg,&sysnum,
    pthresh,punctrh,sdt,sv,vga);

if (sysnum>=maxtarg)
{
    if (vga>0)
    {
        gotoxy(1,5);
        strcpy(message,"Too many system tracks! Press a key.");
        outstring(63,message);
        q=getch();
        defpal(1);          /* Set colours back to normal. */
        setmode(3,0);
        /* Switch to text mode and clear the screen. */
    }
    else
    {
        printf("\n Too many system tracks! \n ");
    }
    for (i=0; i<radnum; i++) /* Close track files. */
    {
        fclose(fin[i]);
    }
    fclose(sysfile);
    frmtrix(p,kaln);
    frmtrix(p1,kaln);
    frmtrix(kk,kaln);
    exit(1);
}
/* Display system tracks. */
if (vga>0)
    disp(latorig,lngorig,griddist,nsmx,tail,headsize,
        gridecolor,systarg,sysnum,idcolor,noidcolor,
        sysfile,hist);

eoflag=1;  /* Check if any data left in track files. */
for (i=0; i<radnum; i++)
    {if (feof(fin[i])==0)
     {eoflag=0;}}
loadflag=0;  /* Check if any data is currently loaded. */
for (i=0; i<radnum; i++)
    {if (loaded[i]>0)
     {loadflag=1;}}
ttmp=timestep*scnum;  /* Increment time. */
steady(&tt,ttmp);    /* Fix interval length. */
if (kbhit()!=0)
    /* Check keyboard - exit if a key has been hit. */
    {q=getch();
     eoflag=1;
     loadflag=0;}
}while ((eoflag<1) || (loadflag>0));  /*Loop over scans.*/

for (i=0; i<radnum; i++)  /* Close track files. */
    {fclose(fin[i]);
    fclose(sysfile);
    frmtrx(p,kaln);
    frmtrx(p1,kaln);
    frmtrx(kk,kaln);

    if (vga>0)
    {
        gotoxy(1,25);
        strcpy(message,"Data analysed.");
        outstring(63,message);
        q=getch();  /* Hold last screen until a key is pressed. */
        defpal(1);  /* Set colours back to normal. */
        setmode(3,0);  /* Switch to text mode and clear the screen. */
    }
else
    {
        printf("\n Data analysed.\n ");
    }
return 0;
}

void convxy(float lat,float lng,float *x,float *y,
        float latorig,float lngorig)
{  *x=(lng-lngorig)*(float)(cos((double)latorig))*rearth/1000.0;
   *y=(lat-latorig)*rearth/1000.0;
}

void getscan(fptr fin,int scnum,int *num,
        int *loaded,float *ltime,struct targdat *targ,float stime
        ,float *rbiw, float *azbias, float *ebias,float *rlat,
float *rlng,float *rhgt,int vga)
{"/ * Routine to retrieve track information for the current time
interval. */
    int scan,radar;
    char junk[257];

    *num=-1;
    scan=scan-1;
    if (((*loaded)>0) && (*ltime<=stime))
    {*num=0;
    targ->time[0]=targ->time[*loaded];
    targ->lat[0]=targ->lat[*loaded];
    targ->lng[0]=targ->lng[*loaded];
    targ->z[0]=targ->z[*loaded];
    targ->vlat[0]=targ->vlat[*loaded];
    targ->vlng[0]=targ->vlng[*loaded];
    targ->vz[0]=targ->vz[*loaded];
    targ->id[0]=targ->id[*loaded];
    targ->scan[0]=targ->scan[*loaded];
    *loaded=-1;
    *ltime=-1;}
    if (((*loaded<0) && (feof(fin)==0))
    {do
    {*num=*num+1;
    if (*num>=maxnum)
    {
        if (vga>0)
        {
            defpal(1);
            setmode(0,0);
        }
        printf("Program arrays to small. ");
        fclose(fin);
        exit(1);
    }
    fscanf(fin,"%d %f %d %f %f %f %f %f %f %d","%d %f %f %f %f %f %f %f %f %f %d",
            &radar,&scan,&targ->time[*num],&targ->id[*num],
            &targ->lat[*num],
            &targ->lng[*num],&targ->z[*num],&targ->vlat[*num],
            &targ->vlng[*num],&targ->vz[*num]);
    /* Calculate bias correction. */
    /*
        convbias(rbias,azbias,ebias,&targ->lat[*num],&targ->lng[*num],
            &targ->z[*num],rlat,rlng,rhgt); */
    fgets(junk,257,fin);}
    while ((feof(fin)==0) && (targ->time[*num]<=stime));
    if (((targ->time[*num]>stime) && (*loaded<0))
    {*loaded=0;
    *ltime=targ->time[*num];
    *num=*num-1;}}
}

void convbias(float *rrbias,float *aabias,float *eebias,
    float *lat,float *lrg, float *zr, float *rlat,float *rlng,
    float *rhgt)
double latr,longr,az,ang,rr[3],north[3],p[3],tmp[3];
double range,omega,tlat,tlng,rrdotp,rbias,azbias,tt[3],east[3];
double orient,latn,lngn,zn,ebias,z,elev,hr,ztest,reff;

latr=(double) *rlat;
longr=(double) *rlng;
tlat=(double) *lat;
trlng=(double) *lng;
rbias=(double) *rrbias;
azbias=(double) *aabias;
zbias=(double) *eebias;
z=(double) *zz;
hr=(double) *rhgt;
reff=rearth+hr;

rr[0]=cos(latr)*cos(longr);
rr[1]=cos(latr)*sin(longr);
rr[2]=sin(latr);
north[0]=sin(latr)*cos(longr);
north[1]=sin(latr)*sin(longr);
north[2]=cos(latr);
cross(east,north,rr);
p[0]=cos(tlat)*cos(tlng);
p[1]=cos(tlat)*sin(tlng);
p[2]=sin(tlat);
cross(tmp,rr,p);    /* if tmp=0 then r=0 */
if (fabs(ang)<1e-6)
{
    range=z-reff;
    az=0.0;
    elev=1.5707963;
}
else
{
    cross(tt,tmp,rr);
    ang=sqrt(tt[0]*tt[0]+tt[1]*tt[1]+tt[2]*tt[2]);
tt[0]=tt[0]/ang;
tt[1]=tt[1]/ang;
    ang=tt[0]*north[0]+tt[1]*north[1]+tt[2]*north[2];
    if (fabs(ang)>1.0)
    {
        if (ang>1.0)
            ang=1.0;
        if (ang<-1.0)
            ang=-1.0;
    }
    az=acos(ang);
    ang=tt[0]*east[0]+tt[1]*east[1]+tt[2]*east[2];
    orient=acos(ang);
    if (orient>1.5707963)
        az=-az;

119
```
rrdotp=rr[0]*p[0]+rr[1]*p[1]+rr[2]*p[2];
if (fabs(rrdotp)>1.0)
{
    if (rrdotp>1.0)
        rrdotp=1.0;
    if (rrdotp<-1.0)
        rrdotp=-1.0;
}
omega=acos(rrdotp);
range=reff*tan(omega);
range=sqrt(reff*reff+z*z-2.0*reff*z*rrdotp);
elev=acos(z*sin(omega)/range);
ztest=reff/cos(omega);
/* Check the sign of the elevation. */
if (ztest>z)
    elev=-elev;
}
az=az+azbias;
range=range+rbias;
elev=elev+ebias;
intogeo(range,az,elev,longr,latr,reff,&lngn,&latn,&zn);
*lat=(float) latn;
*lng=(float) lngn;
*zz=(float) zn;
}

void cross(double ro[],double ri1[],double ri2[])
{
    ro[0]=ri1[1]*ri2[2]-ri1[2]*ri2[1];
    ro[1]=ri1[2]*ri2[0]-ri1[0]*ri2[2];
    ro[2]=ri1[0]*ri2[1]-ri1[1]*ri2[0];
}

void intogeo(double range,double az,double elev,double longr,double latr,double reff,double *lng,double *lat,double *zn)
{
    double rr[3],rt[3],north[3],east[3],norm[3],tmp[3];
    double sn,cs,radius;
    double dtemp1,dtemp2;

    rr[0]=cos(latr)*cos(longr);
    rr[1]=cos(latr)*sin(longr);
    rr[2]=sin(latr);
    north[0]=-sin(latr)*cos(longr);
    north[1]=-sin(latr)*sin(longr);
    north[2]=cos(latr);
    cross(east,north,rr);
    tmp[0]=cos(az)*north[0]+sin(az)*east[0];
    tmp[1]=cos(az)*north[1]+sin(az)*east[1];
    cross(norm,rr,tmp);
    cross(tmp,norm,rr); /* 2nd basis vector for great circle */
    dtemp1=pow(range,2.0);
    dtemp1+=pow(reff,2.0);
```
dtemp1=2.0*range*reff*sin(elev);
radius=sqrt(dtemp1);
sn=range*cos(elev)/radius;
if ((1.0-abs(sn*sn)) > 1.0e-6)
    {  
dtemp1=1.0-sn*sn;
cs=sqrt(dtemp1);
    }
else
    {
    cs=0.0;
    }
rt[0]=cs*rr[0]+sn*tmp[0];
dtemp2=rt[2];
dtemp2=pow(dtemp2,2.0);
dtemp2=1.0-dtemp2;
dtemp1=(rt[2])/sqrt(dtemp2);
*lat=atan(dtemp1);
dtemp1=rt[1]/rt[0];
*lng=atan(dtemp1);
*zn=radius;
}

void disp(float latorig,float lngorig,float griddist,
          float nsmmax,float tail,int headsize,unsigned char gridcolor,
          struct systrack systarg[],int sysnum,unsigned char idcolor,
          unsigned char noidcolor,fptr sysfile,int hist)
{ /* Routine to plot current tracks on the screen with tail length
    indicating target velocity. The colour indicate the presence or
    absence of an identification code for the target. */
  int ix1,ix2,yy1,yy2,i;
  float centx,centy,scalenx,scaleny,x1,y1,x2,y2,scx,scy;
  unsigned char color,newplot;

  scx= (float) screenx;
  scy= (float) screeny;
  centx=scx/2.0;
  centy=scy/2.0;
  scaley=scy/nsmmax/2.0;
  scalex=aspect/nsmmax*scx/2.0;
  for (i=0; i<maxtarg; i++)
    {if (((hist>0)&(&(systarg[i].x1<=screenx)))
      {putblock(0,systarg[i].x1,systarg[i].y1,headsize);
       line(0,systarg[i].x1,systarg[i].y1,
            systarg[i].x2,systarg[i].y2);
       gridre(gridcolor,griddist,nsmmax,
              systarg[i].x1,systarg[i].y1,
              systarg[i].x2,systarg[i].y2,headsize);
       systarg[i].x1=screenx+1;}}
  for (i=0; i<sysnum; i++)
    {
convxy(systarg[i].lat,systarg[i].lng,&x1,&y1,latorig,lngorig);
convxy(systarg[i].lat-tail*systarg[i].vlat,
systarg[i].lng-tail*systarg[i].vlna, &x2,&y2, latorig, lngorig);
newplot=0;
if (((float)fabs((double)x1*scalex))*{centx-1})&&
((float)fabs((double)y1*scaley))*{centy-1})&&
((float)fabs((double)x2*scalex))*{centx-1})&&
((float)fabs((double)y2*scaley))*{centy-1})&&
(systarg[i].vlat*(float)cos((double)systarg[i].lat)<maxvel) &&(systarg[i].vlna<maxvel))
{

fprintf(sysfile,
"%11.1f %11.6f %11.6f %11.6f %11.6f %11.6f %11.6f %4d\n ",
systarg[i].time,systarg[i].lat,systarg[i].lng,systarg[i].z,
systarg[i].vlat,systarg[i].vlna,systarg[i].vz,systarg[i].id);
ix1=round(centx+x1*scalex);
ix2=round(centx+x2*scalex);
jy1=round(centy-y1*scaley);
jy2=round(centy-y2*scaley);
color=oidcolor;
if (systarg[i].id>=0)
{color=oidcolor;}
newplot=1;
}

if (newplot>0)
{
    putblock(color,ix1,jy1,headsize);
    if (hist>0)
    {
        line(color,ix1,jy1,ix2,jy2);
        systarg[i].x1=ix1;
        systarg[i].y1=jy1;
        systarg[i].x2=ix2;
        systarg[i].y2=jy2;
    }
    else
    {systarg[i].x1=screenx+1;}
}
}

void getprobs(int rad,int *num1,struct targdat *tagl,
int *sysnum,struct systrack systarg[],float sdt[],float sdv[],
struct probinfo *prob,float height,float pinit[],int vga)
{ /* Routine to determine probabilities of association using the
    Dempster-Shafer method. */
    char message[100],q;
double tmp;
int i,j,k;
unsigned char match;
float t1,t2,t3,pd,pv,mgv1,mgv2,v1dotv2,ct,ang12,d12;
float ml,m2,m3,masstot,dt;

if ((*num1>=0) && (*sysnum>=0))
{
for (i=0; i<maxtarg; i++) /*Initialise update probabilities.*/
    for (j=0; j<maxtarg; j++)
    {
        prob->ptmp[i][j][0]=pinit[0];
        prob->ptmp[i][j][1]=pinit[1];
        prob->ptmp[i][j][2]=pinit[2];
        prob->prb[rad][i][j][3]=0.0;
    }

for (i=0; i<=*num1; i++) /*For this scan loop over targets*/
    for (j=0; j<=*sysnum; j++) /*recorded by each radar.*/
    {
        /*Determine seperation of targets.*/
        prob->prb[rad][i][j][3]=1.0;
        if (((prob->prb[rad][i][j][0]<0.97) &&
             (prob->prb[rad][i][j][1]<0.97))
        {
            /*usual case*/
            dt=targ1.time[i]-systarg[j].time;
            t1=(float) fabs((double)(rearth+height)*
               (targ1.lat[i]-
                systarg[j].lat-dt*systarg[j].vlat));
            t2=(float) fabs((double)(rearth+height)*
               (float) (cos((double)(targ1.lat[i])))*
               (targ1.lng[i]-systarg[j].lng-dt*systarg[j].vlng));
            t3=targ1.z[i]-systarg[j].z-dt*systarg[j].vz;
            if (((((float) fabs((double) t1)>5.0*(sdtr[rad]))
                 ||((float) fabs((double) t2)>5.0*(sdtr[rad])))
                 ||((float) fabs((double) t3)>5.0*(sdtr[rad])))
                {d12=bigdist;}
            else
                {d12=(float)sqrt((double)
                    (sqr(&t1)+sqr(&t2)+sqr(&t3)));
                pd=gauss(&d12,sdtr[rad]);
                /*Probability of distance association.*/
                /*The following determines the Probability of velocity
                association using the angular difference between the velocity
                vectors. Note that the z-components are not used because it has
                a different interpretation (altitude change in m/s).* /
                magv1=(float)((sqrt((double)(sqr(&targ1.vlat[i])
                    +sqr(&targ1.vlng[i]))));
                magv2=(float)((sqrt((double)(sqr(&systarg[j].vlat)
                    +sqr(&systarg[j].vlng)));
                if ((magv1>0) && (magv2>0))
                    {
                        /*Calculate dot product.*/
                        v1dotv2=targ1.vlat[i]*systarg[j].vlat+
                               targ1.vlng[i]*systarg[j].vlng;
                        ct=v1dotv2/magv1/magv2; /*Normalise.*/
                        /*Determine magnitude of the angle.*/
                        tmp=(double) ct;
                        if (((tmp>1.0) && (tmp<1.0001))
                            {tmp=1.0;}
                        if (((tmp<-1.0) && (tmp>-1.0001))
                            {tmp=-1.0;}
                        if (((tmp>1.0))>({tmp<-1.0})
                            {
                                if (vga>0)
{  
defpal(1);
  setmode(0,0);
}
printf("Error in inverse cosine.");
exit(1);
}
ang12=(float)(acos(tmp));
pv=gauss(&ang12,sdv[rad]);
/*Probability of velocity association.*/

/*Calculation of probabilities given velocity information.  
Probability that they are the same target.*/
prob->ptmp[i][j][0]=
(float)(sqrt((double)(pd*pv)))//*P(1)*/
prob->ptmp[i][j][1]=
(float)(sqrt((double)((1.0-pd)*
(1.0-pv))));/*P(2)*/
/*The uncertainty as a proportion of what is left.*/
prob->ptmp[i][j][2]=1.0-/*P(UNC)*/
prob->ptmp[i][j][0]-
prob->ptmp[i][j][1];
else
/*Calculation of probabilities when no velocity information is 
given. The values used are arbitrary.*/
{
  prob->ptmp[i][j][0]=0.5*pd; /*P(1)*/
  prob->ptmp[i][j][1]=0.2; /*P(2)*/
  prob->ptmp[i][j][2]=0.8-0.5*pd;/*P(UNC)*/
}
}
/*Calculate the "masses" and new probabilities.*/
for (i=0; i<maxtarg; i++)/*Loop over id range of radar 1.*/
  for (j=0; j<maxtarg; j++)/*Loop over id range of radar 2.*/
{  
  if ((prob->prb[rad][i][j][0]<=0.97) & 
   (prob->prb[rad][i][j][1]<=0.97))
    /*usual case*/
    m1=prob->ptmp[i][j][0]*prob->prb[rad][i][j][0]+
      prob->ptmp[i][j][0]*prob->prb[rad][i][j][2]+
      prob->ptmp[i][j][2]*prob->prb[rad][i][j][0];
    m2=prob->ptmp[i][j][1]*prob->prb[rad][i][j][1]+
      prob->ptmp[i][j][1]*prob->prb[rad][i][j][2]+
      prob->ptmp[i][j][2]*prob->prb[rad][i][j][1];
    m3=prob->ptmp[i][j][2]*prob->prb[rad][i][j][2];
    if (m1<0.0)
      {m1=0.0;}
    if (m2<0.0)
      {m2=0.0;}
    if (m3<0.0)
      {m3=0.0;}
}
else
    {if (prob->prb_rad[i][j][0]>=0.97)
        {m1=0.95;
         m2=0.0;
         m3=0.05;
        }
    else
        {m1=0.0;
         m2=0.95;
         m3=0.05;
        }
    }
    masstot=m1+m2+m3;               /*Normalise the masses.*/
if (masstot==0.0)
    {
        if (vga>0)
        {
            defpal(1);
            setmode(0,0);
        }
        printf("Error in normalisation.");
        exit(1);
    }
    prob->prb_rad[i][j][0]=m1/masstot;
    prob->prb_rad[i][j][1]=m2/masstot;
    prob->prb_rad[i][j][2]=m3/masstot;
}
}

float sqr(float *x)
{float tmp;
    tmp=(*x)*(*x);
    return(tmp);}

float gauss(float *dd,float sd)
{ float tmp;
    if (((float) fabs((double) (*dd)/(sd))<20)
        {tmp=*dd/(sd);
         tmp=exp(-sqr(&tmp));}
    else
        {tmp=0.0;}
    return(tmp);
}

void sort(struct chronoref queue[],int *qlen)
/* Routine to sort a queue in chronological order. */
{ int i,j,trad,ttarg;
    float ttime;
    for (i=0; i<*qlen-1; i++)
        for (j=i+1; j<*qlen; j++)
{if (queue[i].time>queue[j].time)
 {ttime=queue[j].time;
  trad=queue[j].rad;
  ttarg=queue[j].targ;
  queue[j].time=queue[i].time;
  queue[j].rad=queue[i].rad;
  queue[j].targ=queue[i].targ;
  queue[i].time=ttime;
  queue[i].rad=trad;
  queue[i].targ=ttarg;}}

void makenew(int radnum,struct probinfo *prob,
    struct targdat targ[],int num[],struct systrack systarg[],
    int *sysnum,float pthresh,float puncthr,float sdt[],float
    sdv[],int vga)
 {/* Routine to identify new system tracks. */
    char q,message[100];
    int i,j,k,m,newtar,totnew,systot,newnew;

totnew=0;
for (i=0; i<radnum; i++)
{     /* Loop over all radars. */
    if (num[i]>=0)
    {
        for (j=0; j<=num[i]; j++)
        {          /* Loop over all targets at the radar. */
            newtar=1;
            /* Loop over all targets at the radar. */
            if (*sysnum>=0)
            {
                k=-1;
                do{    /* Check for association with a system track */
                    k++;
                    if (((prob->prb[i][j][k][0]>pthresh)&&(prob->prb[i][j][k][2]>puncthr))
                    {newtar=0; } /* If associated it is not new. */
                }while ((k<*sysnum) && (newtar>0));
            }
            if (newtar==1)
            /* If not associated check for an association */
            {                  /* with one of the new tracks. */
                } /* with one of the new tracks. */
                newnew=1;
                for (m=0; m<totnew; m++)
                {getassoc(&newnew,&systarg[sysnum+1+m],&targ[i],j,
                sdt[i],sdv[i],pthresh,vga);}
/* Note if the track is associated with one of the new tracks no attempt is currently made to combine the information. I.E. if a new target is initially observed by more than one radar than only one observation will be used to initiate the track. */
    if (newnew>0)
    {totnew++;
    /* Initiate a system track. */
systot=sysnum+totnew;
if (systot>=maxtarg)
{
  if (vga>0)
  {
    gotoxy(1,5);
    strcpy(message,"Too many system tracks! Press a key.");
    outstring(63,message);
    q=getch();
    defpal(1); /* Set colours back to normal. */
    setmode(3,0);
    /* Switch to text mode and clear the screen. */
  }
  else
    printf("Too many system tracks! \n ");
    exit(1);
  }
  systarg[systot].time=targ[i].time[j];
  systarg[systot].lat=targ[i].lat[j];
  systarg[systot].lng=targ[i].lng[j];
  systarg[systot].z=targ[i].z[j];
  systarg[systot].vlat=targ[i].vlat[j];
  systarg[systot].vlngr=targ[i].vlngr[j];
  systarg[systot].vz=targ[i].vz[j];
  systarg[systot].id=targ[i].id[j];
  systarg[systot].scan=targ[i].scan[j];
}

*sysnum=*sysnum+totnew;
}

void deltrack(struct systrack systarg[],int *sysnum,
  float pdelthr,float griddist,float nsmax,
  unsigned char gridcolor,float stime,int headsize,
  struct probinfo *prob,float pinit[],int radnum,
  int vga,int hist)
{
  int i,j,k,m,rad,del;

  if (*sysnum>=0)
  {
    i=-1;
    del=0;
    do
    {
      if (del==0)
        {i++;}
      del=0;
      if (((float)fabs((double)(systarg[i].time-stime)))>pdelthr)
      {
        if ((vga>0)&&(hist>0))
          delcrt(griddist,nsmax,gridcolor,systarg[i],headsize);
        for (j=i; j<=(*sysnum-1); j++)
          {

```
```
systarg[j]=systarg[j+1];
for (m=0; m<maxstarg; m++)
    for (rad=0; rad<radnum; rad++)
    {
        prob->prb[rad][m][j][0]=prob->prb[rad][m][j][1][0];
        prob->prb[rad][m][j][1]=prob->prb[rad][m][j][1][1];
        prob->prb[rad][m][j][2]=prob->prb[rad][m][j][1][2];
    }
}
systarg[*sysnum].x1=screenx+1;
systarg[*sysnum].time=-1.0;

for (m=0; m<maxstarg; m++)
    for (rad=0; rad<radnum; rad++)
    {
        prob->prb[rad][m][*sysnum][0]=pinit[0];
        prob->prb[rad][m][*sysnum][1]=pinit[1];
        prob->prb[rad][m][*sysnum][2]=pinit[2];
    }
*sysnum=*sysnum-1;
del=1;
}
}
while ((i<*sysnum)||(i==*sysnum)&&(del==1));
}
}

void delcrt(float griddist,float nsmax, unsigned char gridcolor,
    struct systrack *starg, int headsize)
 {/* Routine to remove deleted tracks from the screen. */

    if (starg.x1<=screenx)
        {putblock(0,starg.x1,starg.y1,headsize);
         line(0,starg.x1,starg.y1,starg.x2,starg.y2);
         gridre(gridcolor,griddist,nsmax,starg.x1,starg.y1,
             starg.x2,starg.y2,headsize);}
}

void getassoc(int *newnew, struct systrack *starg,
    struct targdat *tar, int j,float sdt,float sdv,
    float pthresh,int vga)
 {/* Routine to determine if two new tracks are associated. */
    double tmp;
    float dt,t1,t2,t3,pd,pv,ang12,mgv1,mgv2,d12,v1dotv2,ct,pp;

dt=tar->time[j]-starg->time;
    t1=(float) fabs((double) rearth*(tar->lat[j]-starg->lat
                  -dt*starg->vlat));
    t2=(float) fabs((double) rearth*(float)(cos((double)
                  (tar->lat[j])))*(tar->lng[j]-starg->lng
                  -dt*starg->vlng));
    t3=starg->z[j]-starg->z-starg->vz;
    if (((((float) fabs((double) t1))>5.0*sdt)
         &&(((float) fabs((double) t2))>5.0*sdt)
         &&(((float) fabs((double) t3))>5.0*sdt)
         &&(((float) fabs((double) t4))>5.0*sdt))
!*((float) fabs((double) t3))>5.0*sd)
    {d12=bigdist;}
  else
    {d12=(float) (sqrt((double) (sqr(&t1)+sqr(&t2)+sqr(&t3))));}
  pd=gauss(&d12,sdt);
  /*Probability of distance association.*/
  mag1v=(float) (sqrt((double) (sqr(&tar->vlat[j])
       +sqr(&tar->vlng[j]))));
  mag2v=(float) (sqrt((double) (sqr(&starg->vlat)
       +sqr(&starg->vlng))));
  if ((mag1v>0) && (mag2v>0))
    { /*Calculate dot product.*/
      v1dotv2=tar->vlat[j]*starg->vlat+tar->vlng[j]*starg->vlng;
      ct=v1dotv2/mag1v/mag2v;  /*Normalise.*/
      tmp=(double) ct;
      if ((tmp>1.0) && (tmp<1.0001))
        {tmp=1.0;}
      if ((tmp<-1.0) && (tmp>-1.0001))
        {tmp=-1.0;}
      if ((tmp>1.0) && (tmp<-1.0))
        {
          if (vga>0)
            {
              defpal(1);
              setmode(0,0);
            }
          printf("Error in inverse cosine."
          exit(1);
          }
      ang12=(float) (acos(tmp));
      /*Determine magnitude of the angle.*/
      pv=gauss(&ang12,sdv);
      /*Probability of velocity association.*/
      else
        {pv=1.0;
          pp=(float) (sqrt((double) (pd*pv)));
          if (pp>pthresh)
            {newnew=0;}
        }
  }

void update(struct sysstrack *starg, struct chronoref *que,
        struct targdat targ[], float rbias, float azbias, matrix p,
        matrix p1, matrix kk)
{ matrix xhat, y, b, a, c, r, q;
  float dt;
  int i,j;

  /* "dt" is the time step involved in the update. */
  dt=targ[que->rad].time[que->targ]-starg->time;

  y=mtrx(kalm,1);  /* The observation vector. */
  y[0][0]=targ[que->rad].lat[que->targ];
  y[1][0]=targ[que->rad].lng[que->targ];
  y[2][0]=targ[que->rad].vlat[que->targ];
y[3][0]=targ[que->rad].vln[que->targ];

b=mtrx(kaln,1);
for (i=0; i<6; i++)
    {b[i][0]=0.0;}

xhat=mtrx(kaln,1);
    xhat[0][0]=starg->lat;
    xhat[1][0]=starg->lng;
    xhat[2][0]=starg->vlat;
    xhat[3][0]=starg->vlng;
    xhat[4][0]=rbias;
    xhat[5][0]=azbias;

a=mtrx(kaln,kaln);
for (i=0; i<6; i++)
    for (j=0; j<6; j++)
        {if (i==j)
            a[i][j]=1.0;
        else
            a[i][j]=0.0;
        }
a[0][2]=dt;
a[1][3]=dt;

c=mtrx(kalm,kalm);
for (i=0; i<4; i++)
    for (j=0; j<4; j++)
        {if (i==j)
            c[i][j]=1.0;
        else
            c[i][j]=0.0;
        }

r=mtrx(kalm,kalm);
for (i=0; i<kalm; i++)
    for (j=0; j<kalm; j++)
        {if (i==j)
            r[i][j]=1e-4;    /* The variances. */
        else
            r[i][j]=0.0;
        }

q=mtrx(kaln,kaln);
for (i=0; i<kaln; i++)
    for (j=0; j<kaln; j++)
        {q[i][j]=0.0;
        }

kgain(kaln,kalm,a,c,p,p1,q,r,kk);
kal(kaln,kalm,xhat,y,a,c,kk,b);

starg->lat=xhat[0][0];
starg->lng=xhat[1][0];
starg->vlat=xhat[2][0];
starg->vlng=xhat[3][0];
rbias=xhat[4][0];
azbias=xhat[5][0];
starg->time=targ[que->rad].time[que->targ];
if (targ[que->rad].id[que->targ]>=0)
{
    starg->id=targ[que->rad].id[que->targ];
}
frmtrx(y,kalm);
frmtrx(xhat,kalm);
frmtrx(b,kalm);
frmtrx(a,kalm);
frmtrx(c,kalm);
frmtrx(r,kalm);
frmtrx(q,kalm);
}

void kgain(int N,int M,matrix A,matrix C,matrix P,matrix P1,
            matrix Q,matrix R,matrix K)
{
    matrix T1,T2,T3,T4,T5;
    int i,j;

    T1=mtrx(N,N);
    T2=mtrx(N,N);
    T3=mtrx(N,M);
    T4=mtrx(N,M);
    T5=mtrx(M,M);

    /* find P1 */
    mul(A,P,T1,N,N,N);
    trn(A,T2,N,N);
    mul(T1,T2,P1,N,N,N);
    add(P1,Q,N,N,N);
    /* find K */
    trn(C,T3,M,N);
    mul(P1,T3,T4,N,N,M);
    mul(C,T4,T5,M,N,M);
    add(T5,R,M,M);
    inv(T5,M);
    mul(T3,T5,T4,N,M,M);
    mul(P1,T4,K,N,N,M);
    /* find P */
    mul(K,C,T1,N,M,N);
    mul(T1,P1,T2,N,N,N);
    for (i=0;i<N;i++)
        for (j=0;j<N;j++)
            P[i][j]=P1[i][j]-T2[i][j];
    frmtrx(T1,N);
    frmtrx(T2,N);
    frmtrx(T3,N);
    frmtrx(T4,N);
    frmtrx(T5,M);
void kal(int N, int M, matrix x, matrix y, matrix A, matrix C, 
        matrix K, matrix b)
{
    int i;
    matrix T1, T2, T3;

    T1 = mtr(x(N, 1));
    T2 = mtr(M, 1);
    T3 = mtr(N, 1);

    /* Predict new state vector and store in T1 */
    mul(A, x, T1, N, N, 1);
    add(T1, b, N, 1);

    /* Extract measurement from predicted state vector */
    mul(C, T1, T2, M, N, 1);

    /* Calculate error and store in T3 */
    for (i = 0; i < M; i++)
        T3[i][0] = y[i][0] - T2[i][0];

    /* Apply Kalman gain to error and store in T2 */
    mul(K, T3, T2, N, M, 1);

    /* New state vector estimate is sum of T2 and T1 */
    for (i = 0; i < N; i++)
        x[i][0] = T2[i][0] + T1[i][0];

    frmtr(T1, N);
    frmtr(T2, M);
    frmtr(T3, N);
}

void mul(matrix A, matrix B, matrix C, int L, int M, int N)
{
    float S;
    int i, j, k;

    S = 0.0;
    for (i = 0; i < L; i++)
        for (j = 0; j < N; j++)
        {
            S = 0.0;
            for (k = 0; k < M; k++)
                S += A[i][k] * B[k][j];

            C[i][j] = S;
        }
}

void trn(matrix A, matrix B, int N, int M)
{
    int i, j;

    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            B[j][i] = A[i][j];
}

void add(matrix A, matrix B, int N, int M)
{
    int i, j;
}
for (i=0; i<N; i++)
    for (j=0; j<M; j++)
        A[i][j]+=B[i][j];

void inv(matrix A, int N)
{ int i, j, k;

    for (k=0; k<N; k++)
    {
        for (j=0; j<N; j++)
            if (j != k)
                A[k][j]=A[k][j]/A[k][k];
        A[k][k]=1.0/A[k][k];
        for (i=0; i<N; i++)
            if (i != k)
                for (j=0; j<N; j++)
                    if (j != k)
        for (i=0; i<N; i++)
            if (i != k) A[i][k]=-A[i][k]*A[k][k];
    }
}

/* function to allocate memory for a matrix */
matrix mtrx(int N, int M)
{ matrix m;
    int i;
    unsigned long memfree;
    memfree=N*M*sizeof(float);
    if (farcoreleft() < memfree)
    {
        defpal(1);
        setmode(0, 0);
        printf("Not enough memory available.\n ");
        printf("%ul requested; %ul available.\n ", memfree, farcoreleft());
        exit(1);
    }
    if((m=(float far **)) farmalloc(N*sizeof(float))) == NULL)
    {
        defpal(1);
        setmode(0, 0);
        perror("Memory allocation error");
        exit(1);
    }
    for (i=0; i<N; i++)
    {
        m[i]=(float far *) farmalloc(M*sizeof(float));
        if (m[i] == NULL)
        {
            defpal(1);
            setmode(0, 0);
            perror("Memory allocation error");
        }
exit(1);
}
}
return m;
}

/* function to free memory allocated using mtrx */
void frmtrx(matrix A, int N)
{ int i;
  for (i=0; i<N; i++)
    free((float *) A[i]);
    free((float *) A);
}

void putpix(unsigned char mag, int col, int row)
{ /* Routine to place a graphics pixel of colour "mag" at
   position
   column "col" and row "row". */
  int tcol, trow;
  tcol=col;
  trow=row;
  _AH = 0xc;
  _AL = mag;
  _BH = 0x0;
  _CX = tcol;
  _DX = trow;
  geninterrupt(0x10);
}

void setmode(unsigned char mode, unsigned char nuclear)
{ /* Routine to set the video mode. The "nuclear" flag controls
   the clearing of the screen. */
  _AL=mode;
  if (nuclear>0)
    {_AL=_AL|128;}
    _AH=0x0;
    geninterrupt(0x10);
}

void defpal(unsigned char ans)
{ /* Enable (1) or disable (0) loading the default colors upon
   mode set. */
  if (ans>0) {_AL=0;}
  else {_AL=1;}
    _AH=0x12;
    _BL=0x31;
    geninterrupt(0x10);  }

void setcolors(void)
{ unsigned char g;
  /* addresses : $3c8 - PEL address during write
  $3c9 - PEL Data Register */
  for (g=0; g<255; g++)
    {
outportb(0x3c8,g);  /* Select color register to change. */
if (g<64)
  { outportb(0x3c9,g);  /* Send the data to: red */
    outportb(0x3c9,g);  /* green */
    outportb(0x3c9,g);  /* blue */
  } else if (g<128)
  { outportb(0x3c9,0);  /* Send the data to: red */
    outportb(0x3c9,0);  /* green */
    outportb(0x3c9,0);  /* blue */
  } else if (g<192)
  { outportb(0x3c9,g);  /* Send the data to: red */
    outportb(0x3c9,0);  /* green */
    outportb(0x3c9,0);  /* blue */
  } else /* g<256 */
  { outportb(0x3c9,0);  /* Send the data to: red */
    outportb(0x3c9,g);  /* green */
    outportb(0x3c9,0);  /* blue */
  }
}

void rectangle(unsigned char color,int x1,int y1,int x2,int y2)
{ line(color,x1,y1,x2,y1);
  line(color,x2,y1,x2,y2);
  line(color,x2,y2,x1,y2);
  line(color,x1,y2,x1,y1);
}

void line(unsigned char color,int x1,int y1,int x2,int y2)
{ /* Routine to plot a line. The const "dens" controls the density of
        points and influences the speed. */
  const float dens=800.0;
  int i;
  float dx,dy,x,y;

dx=(float) x2;
  dx=(dx-(float) x1)/dens;
  dy=(float) y2;
  dy=(dy-(float) y1)/dens;
  x=((float) x1)-dx;
  y=((float) y1)-dy;
  i=0;
  do
    {x=x+dx;
     y=y+dy;
      putpix(color,round(x),round(y));
     i=i+1;}
  while (i<round(dens));
}

int round(float x)
{ int xx;
  xx=(int) (x+0.5);
  return xx;}

135
void putblock(unsigned char color, int x, int y, int size)
{ /* Routine to place a block of size "size" at (x,y). */
  int i, j, k;
  double tmp;
  tmp = ((double) size) / 2.0;
  k = (int) floor(tmp);
  for (i = x - k; i <= x + k; i++)
    for (j = y - k; j <= y + k; j++)
      { putpix(color, i, j); }
}

void outchar(unsigned char color, char q)
{ /* Routine to write a character on a graphics screen. */
  _AH = 0x0A;
  _AL = q;
  _BH = 0x00;
  _BL = color;
  _CX = 0x01;
  geninterrupt(0x10);}

void outstring(unsigned char color, char qq[])
{ /* Routine to write a string on a graphics screen. */
  char q;
  int i, slen;

  slen = strlen(qq) - 1;
  for (i = 0; i <= slen; i++)
    { q = qq[i];
      outchar(color, q);
      gotoxy(wherey() + 1, wherey());}
}

void grid(unsigned int color, float griddist, float nsmax)
{ /* Routine to draw a grid on the screen. "nsmax" is the distance
    from the screen centre to the top. "griddist" is the distance
    between grid lines. */
  int i, ii, d;
  float scale, centx, centy;

  centx = ((float) screenx) / 2.0;
  centy = ((float) screeny) / 2.0;
  scale = griddist / nsmax * ((float) screeny) / 2.0;
  ii = (int) floor((double)(centy / scale));
  for (i = -ii; i <= ii; i++)
    { d = round(centy + i * scale);
      line(color, 1, d, screenx, d); }
  scale = aspect * griddist / nsmax * screenx / 2.0;
  ii = (int) floor((double)(centx / scale));
  for (i = -ii; i <= ii; i++)
    { d = round(centx + i * scale);
      line(color, d, 1, d, screeny); }
}
void plotsite(unsigned char color, float rlat[], float rlng[], int radnum, float latorig, float lngorig, float nmax)
{ /* Routine to plot the radar sites on the screen. */
  int i, ii, jj;
  float centx, centy, scalex, scaley, x, y;
  centx=((float) screenx)/2.0;
  centy=((float) screeny)/2.0;
  scaley=((float) screeny)/nmax/2.0;
  scalex=aspect/nmax*((float) screenx)/2.0;
  for (i=0; i<radnum; i++)
  {
    convxy(rlat[i], rlng[i], &x, &y, latorig, lngorig);
    if (((double) fabs(x*scalex))<(centx-1.0))
      {ii=round(centx+x*scalex);
       if (((double) fabs(y*scaley))<(centy-1.0))
         {jj=round(centy+y*scaley);
          putblock(color, ii, jj, 5);}
      }
  }
}

void steady(double *tt, float ttmp)
{ /* Routine to make computer pause until the end of the current
time interval. "tt" is the initial time. "ttmp" is the end of the
current interval. */
  time_t gmt;
  double ttmpd, tgm;

  if (ttmp==0)
    {gmt=time(NULL);
     *tt=(double) gmt;}
  else
    {ttmpd=(double) (ttmp);
     do
       tgm=(double)(time(NULL))-(*tt);
     while (tgm<ttmpd);
    }

void gridre(unsigned char color, float griddist, float nmax,
             int x1, int y1, int x2, int y2, int headsize)
{ /* Routine to "repair" the grid after tracks have been erased.
   Currently the radar sites are not refreshed. */
  int i, ii, i2, ii, bb, d;
  float scale, centx, centy, scx, scy;
  double tmp;

  tmp= (double) (headsize);
  tmp=tmp/2.0;
  bb=(int) floor(tmp);
  centx=((float) screenx)/2.0;
  centy=((float) screeny)/2.0;
  scale=griddist/nmax*screeny/2.0;
  if (y1>y2)
    {i=y1;
     y1=y2;
y2=i;
if (x1>x2)
{i=x1;
 x1=x2;
 x2=i;
}ii=(int) floor((double)(centy/scale));
i1=x1-bb;
if (ii<i1)
{i1=1;}
i2=x2+bb;
if (i2>screenx)
{i2=screenx;}
for (i=-ii; i<=ii; i++)
{d=round(centy+i*scale);
 if (((d>=y1-bb) && (d<=y2+bb))
 {line(color,i1,d,i2,d);}}
scale=aspect*griddist/nmax*screenx/2.0;
ii=(int) floor((double)(centx/scale));
i1=y1-bb;
if (ii<i1)
{i1=1;}
i2=y2+bb;
if (i2>screeny)
{i2=screeny;}
for (i=-ii; i<=ii; i++)
{d=round(centx+i*scale);
 if (((d>=x1-bb) && (d<=x2+bb))
 {line(color,d,i1,d,i2);}}
APPENDIX 3B

Program listing for the coordinate transformations program, GEOCONV.C.

#include <math.h>
#include <conio.h>
#include <stdio.h>
void cross(double ro[],double ri1[],double ri2[]);
void intogeo(double range,double az,double elev,double longr,double latr,
     double z,double *lng,double *lat,double *zn);
void convbias(float *rrbias,float *aabias,float *eebias,
    float *lat,float *lng,float *zz,float *rlat,float *rlng,
    float *hhr);
const float rearth=6378000.0;  /* Radius of the earth in metres. */
const float degtorad=3.14159265/180.0;  /* Degree to radian factor. */
int main()
{
    float rbias,abias,ebias,lat,lng,zz,rlat,rlng,hhr;
    char q;

    printf("CONVERSION FROM RADAR TO AND BACK FROM GEOCENTRIC COORDINATES.\n");
    printf("Enter the radar position. (lat, lng, height (m)) \n");
    scanf("%f %f %f",&rlat,&rlng,&hhr);
    rlat=rlat*degtorad;
    rlng=rlng*degtorad;
    printf("Enter the target position. (lat, lng, height (m)) \n");
    scanf("%f %f %f",&lat,&lng,&zz);
    lat=lat*degtorad;
    lng=lng*degtorad;
    zz=zz+rearth;
    printf("Enter the biases. (range (m), azimuth (°), elevation (°)) \n");
    scanf("%f %f %f",&rbias,&aabias,&ebias);
    abias=abias*degtorad;
    ebias=ebias*degtorad;
    printf("\n Target location in geocentric coordinates.\n");
    printf("Latitude (°), longitude (°), altitude (m) %f %f %f\n",
        lat/degtorad,lng/degtorad,zz-rearth);
    convbias(&rbias,&aabias,&ebias,&lat,&lng,&zz,&rlat,&rlng,&hhr);
    printf("Target location after conversion back to geocentric coordinates.\n");
    printf("Latitude (°), longitude (°), altitude (m) %f %f %f\n", 
        lat/degtorad,lng/degtorad,zz-rearth);
    return 0;
}

void convbias(float *rrbias,float *aabias,float *eebias,
    float *lat,float *lng,float *zz,float *rlat,float *rlng,
float *hhr
{
    double latr, longr, az, ang, rr[3], north[3], p[3], tmp[3];
    double range, omega, tlat, tlng, rrdotp, rbias, azbias, tt[3], east[3];
    double orient, latn, lngn, zn, ebias, z, elev, hr, ztest, reff;
    char q;

    latr=(double) *rlat;
    longr=(double) *rlng;
    tlat=(double) *lat;
    tlng=(double) *lng;
    rbias=(double) *rrbias;
    azbias=(double) *aaabias;
    ebias=(double) *eebias;
    z=(double) *zz;
    hr=(double) *hhr;
    reff=rearth+hr;

    rr[0]=cos(latr)*cos(longr);
    rr[1]=cos(latr)*sin(longr);
    rr[2]=sin(latr);
    north[0]=-sin(latr)*cos(longr);
    north[1]=-sin(latr)*sin(longr);
    north[2]=cos(latr);
    cross(east,north,rr);
    p[0]=cos(tlat)*cos(tlng);
    p[1]=cos(tlat)*sin(tlng);
    p[2]=sin(tlat);
    cross(tmp,rr,p);
    if (fabs(ang)<1e-6)
    {
        range=z-reff;
        az=0.0;
        elev=1.5707963;
    }
    else
    {
        cross(tt,tmp,rr);
        ang=sqrt(tt[0]*tt[0]+tt[1]*tt[1]+tt[2]*tt[2]);
        tt[0]=tt[0]/ang;
        tt[1]=tt[1]/ang;
        ang=tt[0]*north[0]+tt[1]*north[1]+tt[2]*north[2];
        if (fabs(ang)>1.0)
        {
            if (ang>1.0)
                ang=1.0;
            if (ang<-1.0)
                ang=-1.0;
        }
        az=acos(ang);
        ang=tt[0]*east[0]+tt[1]*east[1]+tt[2]*east[2];
        orient=acos(ang);
if (orient>1.5707963)
    az=-az;

    rrdotp=rr[0]*p[0]+rr[1]*p[1]+rr[2]*p[2];
    if (fabs(rrdotp)>1.0)
    {
        if (rrdotp>1.0)
            rrdotp=1.0;
        if (rrdotp<-1.0)
            rrdotp=-1.0;
    }
    omega=acos(rrdotp);
    range=reff*tan(omega);
    range=sqrt(reff*reff+z*z-2.0*reff*z*rrdotp);
    elev=acos(z*sin(omega)/range);
    ztest=reff/cos(omega);
    if (ztest>=z)
        elev=-elev;
}

printf("In radar coordinates coordinates before bias correction.\n");
    printf("The range (m), azimuth (*), elevation (*) %lf %lf %lf\n",
    range,az/deg2rad,elev/deg2rad);
    az=az+azbias;
    range=range+rbias;
    elev=elev+ebias;
    printf("In radar coordinates coordinates after bias correction.\n");
    printf("The range (m), azimuth (*), elevation (*) %lf %lf %lf\n",
    range,az/deg2rad,elev/deg2rad);
    intgeo(range,az,elev,longr,latr,reff,&lgnr,&ltnr,&zn);
    *lat=(float) ltn;
    *lgn=(float) lgnr;
    *zz=(float) zn;
}

void cross(double ro[],double ri1[],double ri2[])
{
    ro[0]=ri1[1]*ri2[2]-ri1[2]*ri2[1];
    ro[1]=ri1[2]*ri2[0]-ri1[0]*ri2[2];
    ro[2]=ri1[0]*ri2[1]-ri1[1]*ri2[0];
}

void intgeo(double range,double az,double elev,double longr,double latr,double reff,double *lgnr,double *ltnr,double *zn)
{
    double rr[3],rt[3],north[3],east[3],norm[3],tmp[3];
    double sn,cs,radius;
    double dtemp1,dtemp2;

    rr[0]=cos(latr)*cos(longr);
    rr[1]=cos(latr)*sin(longr);
    rr[2]=sin(latr);
    north[0]=-sin(latr)*cos(longr);
north[1]=\sin(latitude)*\sin(longitude);
north[2]=\cos(latitude);
cross(east,north,rr);
tmp[0]=\cos(az)*north[0]+\sin(az)*east[0];
tmp[1]=\cos(az)*north[1]+\sin(az)*east[1];
cross(norm,rr,tmp);
cross(tmp,norm,rr); /* 2nd basis vector for great circle */
dtemp1=pow(range,2.0);
dtemp1+=pow(reff,2.0);
dtemp1+=2.0*range*reff*\sin(elevation);
radius=sqrt(dtemp1);
sn=range*\cos(elevation)/radius;
if ((1.0-\abs(sn*sn)) > 1.0e-6)
{
    dtemp1=1.0-sn*sn;
    cs=sqrt(dtemp1);
}
else
{
    cs=0.0;
}
rt[0]=cs*rr[0]+sn*tmp[0];
dtemp2=rt[2];
dtemp2=pow(dtemp2,2.0);
dtemp2=1.0-dtemp2;
dtemp1=(rt[2])/sqrt(dtemp2);
*lat=atan(dtemp1);
dtemp1=rt[1]/rt[0];
*lng=atan(dtemp1);
*zn=radius;
APPENDIX 4A

Program Listing for the Aiyer Neural Network.

{ $N+$ }
{ aiyer.pas  Hopfield neural network for the travelling salesman
  problem according to Aiyer et al, 1990. }

uses crt;
const
  maxsize=11;
  st=10;
type
  arrays=array[1..maxsize,1..maxsize] of single;
  arrayi=array[1..maxsize] of single;
  rkarray=array[1..5] of single;
var
  rk4:rkarray;
  A,B,C,DD,u0,tau,dt,offset:single;
  step,m:integer;
  uold,u,v,d:arrays;
  sumrow,sumcol:arrayi;
  t,n,du,e,e1,e2,e3,e4,tempe4,tempsingle;
  length:single;
  iter,ii,jj,i,j,k,p,q:integer;
  ans,char;
  frep,fpar:text;

{ piecewise linear function }
{ if (x <= -0.5) then g(x) = 0.0 }
{ if (x > -0.5 and x < 0.5) then g(x) = x+0.5 }
{ if (x >= 0.5) then g(x) = 1.0 }
function glin(var x:single):single;
var
  temp:single;
beginn
  if (abs(x) < 0.5) then
    temp:=x+0.5
  else
    if (x > 0.0) then
      temp:=1.0
    else
      temp:=0.0;
  glin:=temp;
end;

begin
  randomize;
  assign(frep,'junk.dat');
  rewrite(frep);
  repeat
    { read in parameters from data file }
    clrscr;
    assign(fpar,'aiyer.dat');
    reset(fpar);
readln(fpar,a,b,c,DD,u0,tau,dt,m,offset);
writeln(' a b c d u0 tau dt m offset ');
writeln(a:4:1,b:4:1,c:4:1,DD:4:1,u0:5:2, 
  tau:5:1,dt:9:3,m:5,offset:10:4);
close(fpar);
{ read in city data }
assign(fpar,'city.dat');
reset(fpar);
for i:=1 to m do 
  begin 
    for j:=1 to m do 
      read(fpar,d[i,j]);
      readln(fpar);
    end;
close(fpar);

{ begin initialisation }
rk4[1]:=1.0;
rk4[2]:=0.5;
rk4[3]:=0.5;
rk4[4]:=1.0;
rk4[5]:=1.0;
temp:=(1.0/m-0.5)*u0;
writeln('Initial states ');
for i:=1 to m do 
  begin 
    for j:=1 to m do 
      begin 
        u[i,j]:=temp+(random-0.5)*u0/50.0;
        writeln(u[i,j]:7:3);
      end;
    writeln;
  end;

writeln('Initial states ');
{ calculate initial values for first update }
n:=0.0;
for i:=1 to m do 
  begin 
    sumcol[i]:=0.0;
    sumrow[i]:=0.0;
  end;
for i:=1 to m do 
  begin 
    for j:=1 to m do 
      begin 
        temp:=u[i,j]/u0;
        v[i,j]:=glin(temp); {linear function of Aiyer}
        n:=n+v[i,j];
        sumrow[i]:=sumrow[i]+v[i,j];
        sumcol[j]:=sumcol[j]+v[i,j];
        writeln(v[i,j]:7:3);
      end;
    writeln;
  end;
end;
{ writeln('Press any key to start.');
 ans:=readkey;}
c1rscr;
{ begin updates, loop over time }
iter:=0;
repeat
iter:=iter+1;
if (iter mod st = 0) then
begin
gotoxy(1,1);
writeln('iteration ',iter:5);
for i:=1 to m do
 writeln(i:5,sumcol[i]:10:3,sumrow[i]:10:3);
end;
n:=0.0;
for j:=1 to m do
begin
sumcol[j]:=0.0;
sumrow[j]:=0.0;
end;
for i:=1 to m do
begin
for j:=1 to m do
begin
n:=n+v[i,j];
sumrow[i]:=sumrow[i]+v[i,j];
sumcol[j]:=sumcol[j]+v[i,j];
uold[i,j]:=u[i,j];
end;
end;
for step:=1 to 4 do
for ii:=1 to m do
for jj:=1 to m do
begin
e1:=sumrow[ii]-v[ii,jj];
e2:=sumcol[jj]-v[ii,jj];{signals according to Hopfield}
{ calculate new input and output for the neuron }
 du:=0.0;  { no decay }
du:=du-A*e1-B*e2-C*(m-n)+offset*n;  { neural forces }
du:=du-2.0*(31.0/32.0)*a*v[ii,jj];  { self inhibition }
p:=jj-1;
if (p<1) then p:=m;
q:=jj+1;
if (q>m) then q:=1;
temp:=0.0;
for k:=1 to m do
 temp:=temp+d[ii,k]*(v[k,q]+v[k,p]);
 du:=du-dd*temp;
u[ii,jj]:=u[ii,jj]+du*dt*rk4[step+1];
uold[ii,jj]:=uold[ii,jj]+du*dt/6.0/rk4[step];
end;
temp := u[i,j]/u0;
    v[i,j] := glin(temp);  \{ linear function of Aiyer \}
    end;
end;
for i := 1 to m do
    for j := 1 to m do
        begin
            u[i,j] := uold[i,j];
            temp := u[i,j]/u0;
            v[i,j] := glin(temp);  \{ linear function of Aiyer et al, 1990 \}
        end;
    n := 0.0;
    for j := 1 to m do
        begin
            sumcol[j] := 0.0;
            sumrow[j] := 0.0;
        end;
    for i := 1 to m do
        begin
            for j := 1 to m do
                begin
                    n := n + v[i,j];
                    sumrow[i] := sumrow[i] + v[i,j];
                    sumcol[j] := sumcol[j] + v[i,j];
                    if (iter mod st = 0) then
                        begin
                            write(v[i,j]:7:3);
                        end;
                    if (iter mod st = 0) then
                        writeln;
                end;
        end;
\{ calculate energy of network \}
e1 := 0.0;
    for i := 1 to m do
        for j := 1 to m do
            for k := 1 to m do
                if \( j \neq k \) then
                    e1 := e1 + v[i,j] * v[i,k];
e1 := e1 * A/2.0;
e2 := 0.0;
    for i := 1 to m do
        for j := 1 to m do
            for k := 1 to m do
                if \( k \neq j \) then
                    e2 := e2 + v[j,i] * v[k,i];
e2 := e2 * B/2.0;
e3 := C/2.0*(n-m)*(n-m);  \{ for m neurons on \}
e4 := 0.0;
    for i := 1 to m do
        for j := 1 to m do
            for k := 1 to m do
                begin
                    p := k-1;
                    if (p<1) then p := m;
                    q := k+1;
if (q>m) then q:=1;
e4:=e4+d[i,j]*v[i,k]*(v[j,p]+v[j,q]);
end;
e4:=e4*dd/2.0;
if (iter mod st = 0) then
begin
writeln('Energy terms: A B C D Total ');
writeln(' ');
writeln(' e1:8:3,' ' ,e2:8:3,' ' ,e3:8:3,' ' ,e4:8:3,
' ' ,e1+e2+e3+e4:8:3);
end;
e1:=e1+e2+e3;
{ readln; }
until ((iter >= 1000) or (e1 < a/1000.0)); {exit loop }
if (e1 <= a/1000.0) then
begin
 e4:=0.0;
 for i:=1 to m do
 for j:=1 to m do
 for k:=1 to m do
 begin
  p:=k-1;
  if (p<1) then p:=m;
  q:=k+1;
  if (q>m) then q:=1;
  if ((v[i,k] >= 0.95) and ((v[j,p] >= 0.95) or
  (v[j,q] >= 0.95))) then
  e4:=e4+d[i,j];
 end;
 length:=e4/2.0;
 writeln('Tour length ' ,length:10:3);
 writeln(frep,'Tour length ' ,length:8:4,' iter ' ,iter:5);
end
else
begin
 writeln('Not a valid tour. ');
 writeln(frep,'Not a valid tour. ');
end;
{ readln; }
until keypressed;
close(frep);
end.
**DOCUMENT CONTROL DATA**

<table>
<thead>
<tr>
<th>1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor’s report, or tasking agency, are entered in section 8.)</th>
<th>2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>London Research and Development Corporation 1495 Geary Avenue, London, Ontario N5X 1G6</td>
<td>UNCLASSIFIED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Demonstration of Advanced Techniques for Multi-Radar Target Tracking (U)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. AUTHORS (Last name, first name, middle initial)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SIBBALD, Timothy M., BULLER Eric H. and TUNALEY, James K.E.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. DATE OF PUBLICATION (month and year of publication of document)</th>
<th>6a. NO. OF PAGES (total containing information, Include Annexes, Appendices, etc.)</th>
<th>6b. NO. OF REFS (total cited in document)</th>
</tr>
</thead>
<tbody>
<tr>
<td>March, 1993</td>
<td>147</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memoranda. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Contractor Report</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)</th>
<th>9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)</th>
<th>9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defence Research Establishment Ottawa Ottawa, Ontario K1A 0Z4</td>
<td>041 LO 43</td>
<td>041ST.W7714-2-9637</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10a. ORIGINATOR’S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)</th>
<th>10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(X) Unlimited distribution</td>
<td></td>
</tr>
<tr>
<td>( ) Distribution limited to defence departments and defence contractors; further distribution only as approved</td>
<td></td>
</tr>
<tr>
<td>( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved</td>
<td></td>
</tr>
<tr>
<td>( ) Distribution limited to government departments and agencies; further distribution only as approved</td>
<td></td>
</tr>
<tr>
<td>( ) Distribution limited to defence departments; further distribution only as approved</td>
<td></td>
</tr>
<tr>
<td>( ) Other (please specify):</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Unlimited Announcement</td>
<td></td>
</tr>
</tbody>
</table>

UNCLASSIFIED
13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

This report describes a demonstration system for combining track information from a number of radar sites into system tracks. The target positions and velocities are displayed graphically using earth centred coordinates: latitude and longitude.

The site tracking program is a track-while-scan (TWS) system that uses a relatively simple algorithm to associate observations with tracks. The track updating is performed using an α-β algorithm.

The system tracker uses the Dempster-Shafer method of data fusion to assign probabilities of association between site tracks and system tracks. Following the association, system tracks are updated chronologically using a Kalman filter.

The system tracker has been tested using the Dempster-Shafer method of data fusion with a non-zero uncertainty and also with a Bayesian approach. No significant difference in the approaches was observed, however, reasons for preferring the Dempster-Shafer method are discussed in the report.

The problem of correcting biases in the site radars has been examined and an initial attempt at bias correction has been included in the system tracker. There are currently several unresolved issues concerning bias correction which are discussed in the report.

The application of a neural network to the travelling salesman problem has been studied because it is related to the problem of associating data to tracks. A neural network that works well on the Travelling Salesman Problem (TSP) has been simulated and tested with several sets of ten cities. This neural network can be related to the data association problem and could be used by site and system trackers.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Multi-Target Tracking
Sensor Fusion
Multi-Radar Tracking
<table>
<thead>
<tr>
<th>NO. OF COPIES</th>
<th>COPY NO.</th>
<th>INFORMATION SCIENTIST'S INITIALS</th>
<th>AQUISESSION ROUTE</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>DAR</td>
<td>DREGO</td>
<td>99-Nov-94</td>
</tr>
<tr>
<td>DSIS ACCESSION NO.</td>
<td>NUMÉRO DSIS</td>
<td>95-02496</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PLEASE RETURN THIS DOCUMENT TO THE FOLLOWING ADDRESS:
DIRECTOR
SCIENTIFIC INFORMATION SERVICES
NATIONAL DEFENCE
HEADQUARTERS
OTTAWA, ONT. - CANADA K1A 0K2

PRIÈRE DE RETOURNER CE DOCUMENT À L'ADRESSE SUIVANTE:
DIRECTEUR
SERVICES D'INFORMATION SCIENTIFIQUES
QUARTIER GÉNÉRAL
DE LA DéFENSE NATIONALE
OTTAWA, ONT. - CANADA K1A 0K2