SIMULATION IN THE DESIGN OF AUTOMATED AIR TRAFFIC CONTROL FUNCTIONS

Paul D. Flanagan, Judith B. Currier, Kenneth E. Willis
METIS Corporation

Abstract

This paper describes the design and use of a simulator of some of the newly automated safety separation functions for terminal air traffic control (ATC). The program was used not only for analysis and design of these functions but also as a testbed for the logic actually implemented in the Knoxville, Tennessee terminal. Imbedded in the program is an emulator of the Goodyear Aerospace Corporation STARAN IV Associative Processor used at Knoxville. The three major ATC functions simulated are: 1) advanced mid-air conflict prediction and evaluation, 2) conflict resolution maneuver generation, and 3) automated voice advisory message generation and scheduling.

INTRODUCTION

In early 1971, the Federal Aviation Administration (FAA) began a program to provide expanded automated air traffic control (ATC) functions at the FAA test site in Knoxville, Tennessee. At that time, there was an ongoing program (the ARTS III program) to provide automated radar tracking and alphanumeric display functions at the 63 largest terminals in the U.S. The Knoxville experiment was designed to extend these automated functions to provide safety separation functions, namely, mid-air conflict prediction, resolution maneuver generation, and automated vocal traffic advisories.

Another important aspect of the Knoxville experiment was the evaluation of the STARAN associative processor (AP) built by Goodyear Aerospace Corporation. This new type of computer was to be considered as an addition to the ATC system to provide large amounts of computational power for various specialized ATC functions. The STARAN would act in conjunction with a UNIVAC 1230 processor to provide the data processing required for the Knoxville terminal area.

As participants in this program, the authors designed and used a simulation of the ATC system to be used at Knoxville. The simulator was used for analysis and design of the safety separation software required for the experiment. In addition, the simulator functioned as a testbed for the logic actually implemented in Knoxville.

The major benefits derived from designing and testing the software on the simulator were low cost of implementation and ability to allow parallel software development. Many functions were being programmed for Knoxville, a unique system with only limited time available for program test. By designing and testing the safety separation logic on the simulator, other functions could use more time on the Knoxville system without impeding logic development. In addition, the simulation program was located in the programming facility in the Washington, D.C. metropolitan area. Thus, costs of simulation development were offset by decreased travel costs and reduction in time required on the Knoxville computers.

GENERAL DESIGN

The simulator was a model of the software functions used in the Knoxville experiment. No hardware was simulated explicitly, although hardware characteristics which impacted on the software functions were included in the model. Figure 1 displays the data processing functions of the original Knoxville experiment.

The simulator was written in FORTRAN for the CDC 6600 computer. The functions of the Executive, Beacon and Radar Target Return Processing, and Data Entry and Display Processing were modeled functionally. That is, the input and output of these functions were defined by the Knoxville system and used in the simulator. The logic of these simulated functions, however, was not identical to that used in Knoxville. The Conflict Resolution function was modeled logically. The simulator was used as a test bed for this function. The software of the Associative

Processor was modeled by using an emulator of the AP in the simulation. The same instructions used in the AP at Knoxville were input to the emulation portion of the simulator to provide these functions. In addition to the functions shown on Figure 1, a test track generator was added to the simulator so that the output of the data acquisition system could be simulated.

As the project progressed, various changes were made in the required functions. Conflict resolution was changed so that traffic advisories could be automatically transmitted on the ATC voice channel. Conflict prediction was examined as a serial processor function rather than as a parallel processor function. Thus, various modules not shown on Figure 2 were added to the basic simulation design.

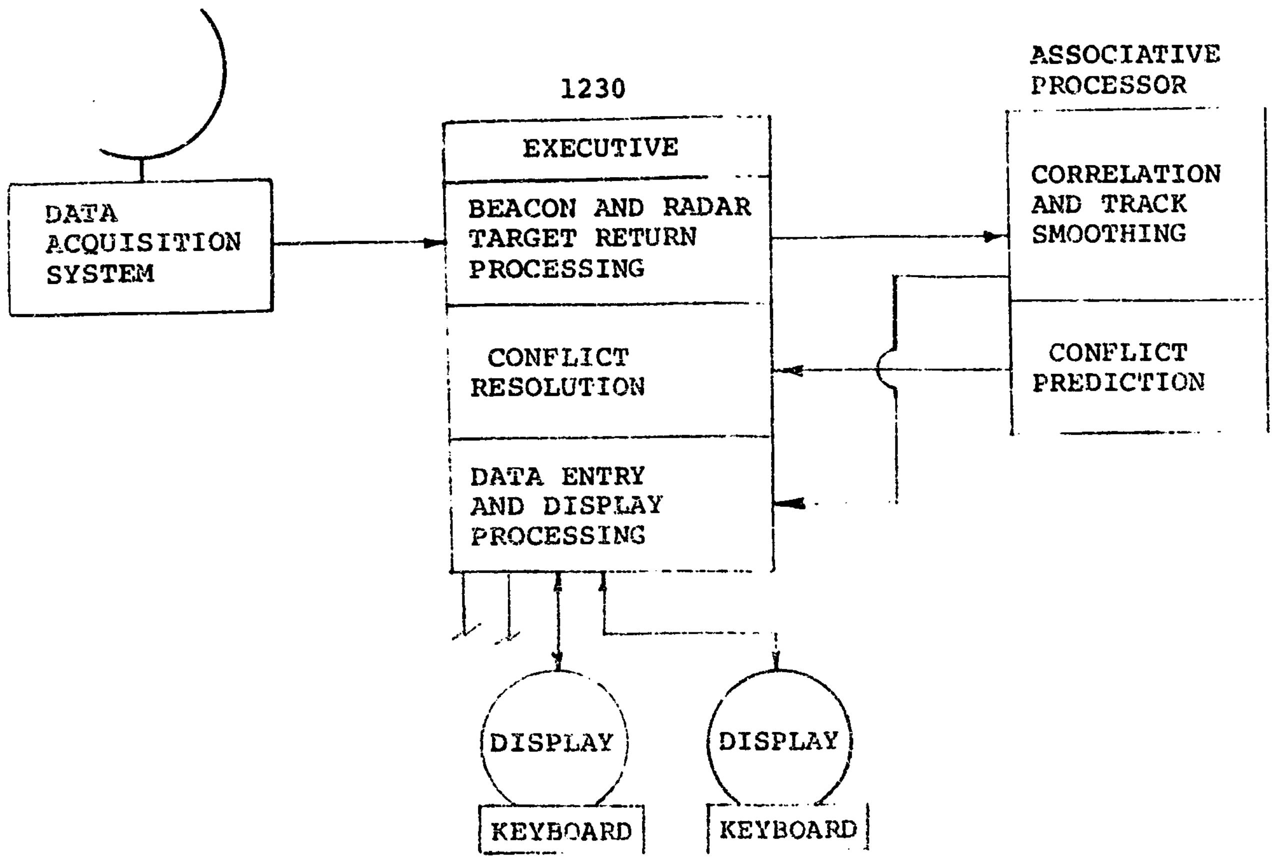

Figure 1

DATA PROCESSING FUNCTIONS - KNOXVILLE EXPERIMENT

Figure 2 shows the simulation structure. The box labeled APEX represents the associative processor emulator. This figure represents the initial simulation design.

These modules were: serial computer tracking logic, serial computer conflict prediction, and automated voice advisory message generation.

The remaining sections of this paper

describe the design of the various simulation modules.

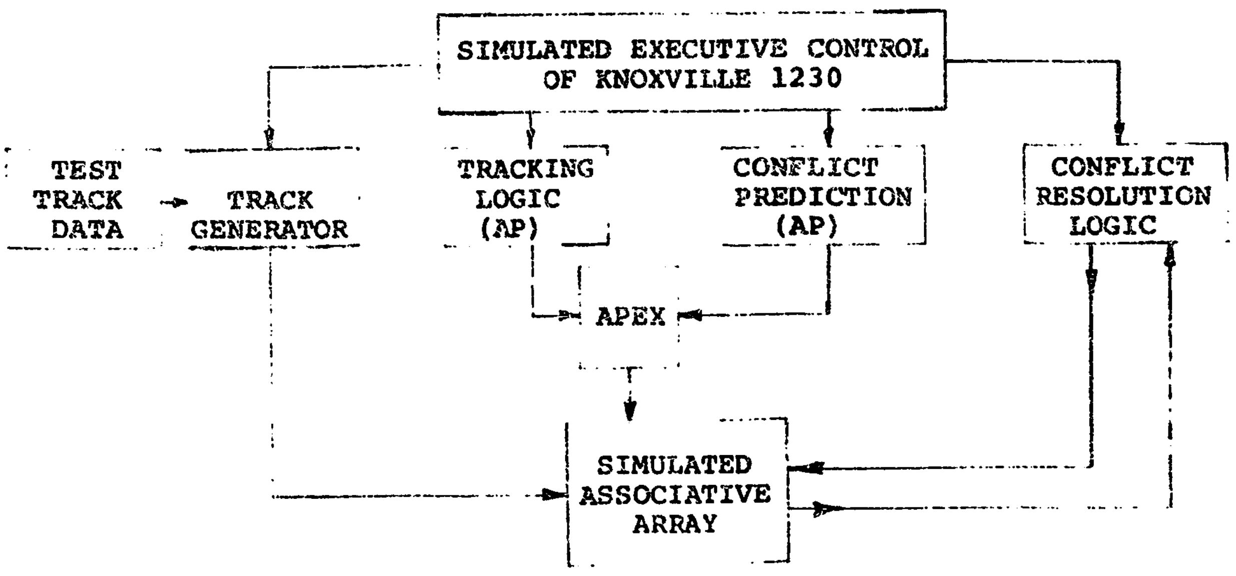
SIMULATED EXECUTIVE CONTROL OF KNOXVILLE 1230
TEST TRACK DATA
TRACK GENERATOR
TRACKING LOGIC (AP)
CONFLICT PREDICTION (AP)
CONFLICT RESOLUTION LOGIC
APEX
SIMULATED ASSOCIATIVE ARRAY

Figure 2

Simulation Structure

position would be reported to the tracking function after a suitable amount of error was added to model the radar position errors.

EXECUTIVE AND TEST TRACK GENERATOR

These two modules were the simplest modules in the simulator. The executive function merely called the other modules in the order they were called at Knoxville. The system worked on a radar scan time basis. Each scan represented four seconds of time, equal to one radar antenna rotation. The track smoothing logic was called eight times a scan (once every half second of simulated time) and all other functions were called once per scan.

The test track generator merely read in a description of the desired test track flight paths. At each scan, the position of each aircraft would be updated to a new position. This new

ASSOCIATIVE PROCESSOR EMULATOR

This section describes the design of the module which functionally simulates the Goodyear Associative Processor, STARAN IV. Although the simulation was designed as a research and development tool, primarily for use in assisting in software development for the Knoxville experiment, the simulator is flexible in design, and is capable of accepting any algorithms written in the associative processor instruction set. It can easily be adapted to accept new or modified AP instructions.

The simulator system accepts as inputs a program written in the associative processor instruction set. The system then interprets and executes the AP instructions so as to provide outputs identical to those that the AP would provide. The simulator also can output the exact configuration of the associative array at any desired point in the AP program. Finally, it provides as output the exact time that would be consumed by the AP in executing the programmed instructions. (These timing calculations include the time required to page new instructions or data into the control memory of the AP.) The simulator is capable of maintaining statistics on the execution sequence in order to identify the time binds in the actual AP.

The Goodyear Associative Processor accepts programs written in an assembly language form. An assembler program executed on a XDS Sigma V computer translates such programs into the machine language instructions required by the AP. In order to make the simulator faster running, and be more useful, it was designed to accept as inputs programs written primarily in the higher order AP assembly language. The simulator maintains a simulated associative array in the core of the host computer. This array is manipulated by the AP instructions precisely as the AP manipulates its associative array.

The AP simulator is designed to operate in two parts. The first part is an assembler or encoder (GAPE - Goodyear Associative Processor Encoder). This program takes AP assembly language instructions as input and produces an interpretative code to be executed by the second part of the simulator. The second part of the simulator actually manipulates the simulated associative memory by executing the interpretative code produced by GAPE. The second part is named APEX for Associative Processor Executor. This division of the simulator is analagous to the standard assemble (or compile) and execute steps usually used in any higher level language. GAPE assembles the code into executable form and APEX loads the interpretative code and simulates the functions of the associative processor. Figure 3 displays the operational flow for AP algorithm execution.

The assembly step (GAPE) reads STARAN IV code directly from card input and translates the instructions into an "executable" form suitable for processing by the AP ecuter (APEX). The output is normally punched cards.

Additionally, the assembler flags errors in the STARAN IV input deck, such as illegal instructions, doubly defined symbols, and undefined symbols. The GAPE assembler is written in FORTRAN IV

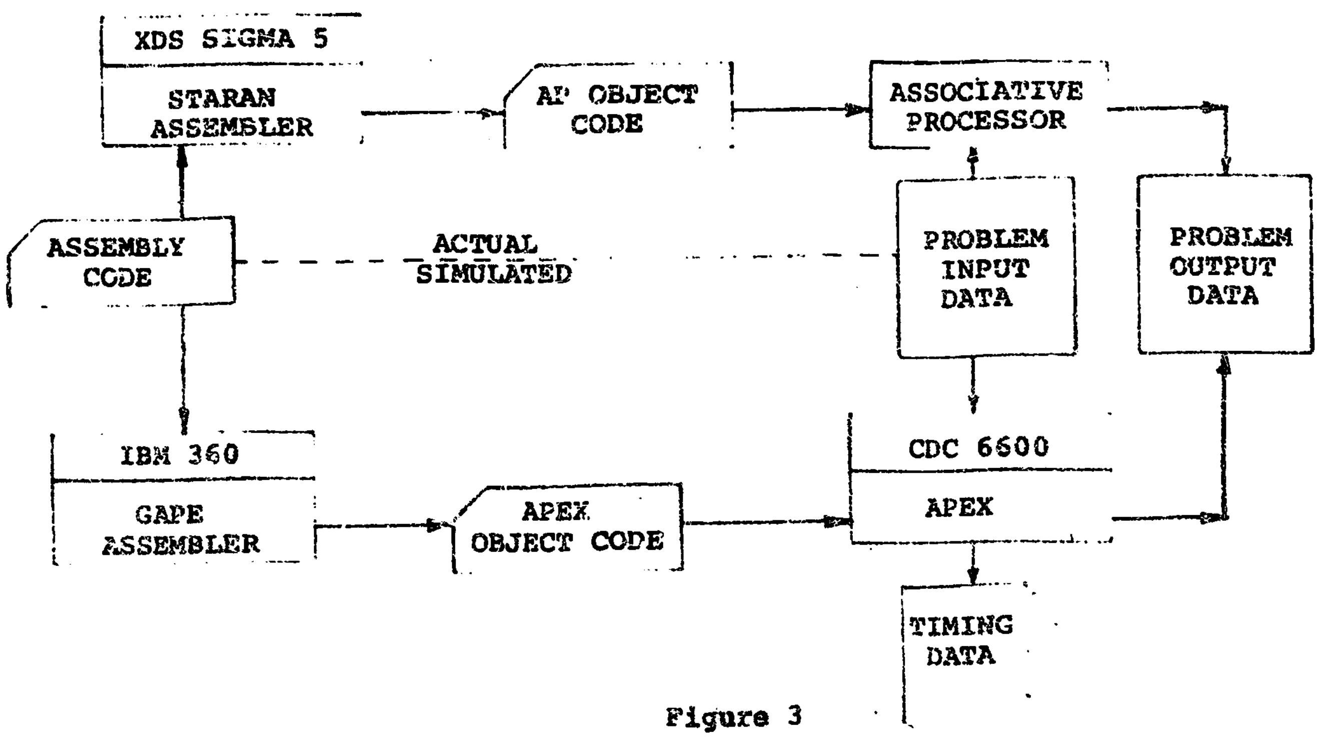

Figure 3

Problem Solution Process

and BAL for the IBM 360.* It occupies approximately 80K bytes of core. Its device requirements are a card reader, a card punch, a printer, and a scratch file. The CPU time required is approximately .1 sec per AP assembly instruction on a 360/65.

The output deck of the assembler is a series of 30 bit words (punched in octal format). The general form of an operation to be performed by APEX is:

If T: then: A .op. B(I)→S(J)

: otherwise: no operation

where T is a parameter to be tested to determine if the instruction is to be executed,

	A	is the first parameter used for the operation,
	.op.	is the binary operation to be performed,
	B	is the second parameter used for an operation,
	I	is the location of the "address" of B if indirect addressing is used,
	S	is the parameter field to be replaced by the results of the operation,
and	J	is the location of the "address" of S if indirect addressing '~ used.

*An alternate FORTRAN version is implemented on a CDC 6600.

The first word (30 bits) of each group of words transferred to APEX is called the command word. It contains basic instruction information. This information includes the instruction operation and suboperation codes (nine bits and four bits respectively), the number of parameter words following the command word (three bits), a parameter descriptor list (five bits), a data register index (five bits), an instruction test parameter (T) (one bit), an argument usage parameter (one bit), and the two most significant bits of the argument if present. If the argument usage bit is set, the command word is followed by a word containing the 30 least significant bits of the argument. Following the command word (or argument word if present) is a series of 15 bit half-words, packed two per word. These parameter half-words each contain a parameter identifier (e.g. A) (two bits), the address of the most significant bit of the parameter field (eight bits), and the number of bits in the parameter field minus one (five bits).

The second part of the emulator, APEX, is implemented in the FORTRAN IV language on the CDC 6600. The code is written to be compatible with the FORTRAN implemented on the UNIVAC 1230 computer.

APEX reads the instruction list provided by GAPE, reads the timing information, and simulates the logical and arithmetic functions of the STARAN IV processor. Figure 4 presents a diagram displaying the relationships between the subroutines comprising APEX. The design of APEX allows the user to input five different AP algorithms for selective execution by an external control program. This mode of operation closely simulates the mode of operation of the AP at Knoxville. As each algorithm is selected, APEX modifies the simulated associative memory according to the assembly instructions which created the algorithm.

There are two basic kind of instructions for the AP. The first kind uses inputs from words of associative memory or the response store and produces output in the associative memory or response store. These instructions must be accomplished sequentially in the simulator through each simulated word of associative memory. Subroutine DOIT provides for execution of these instructions. There are 15 operations or tests which can be performed by the AP on the bit fields of associative memory. These are: less than, less than or equal, equal, not equal, greater than, greater than or equal, logical not, logical and, logical or, logical exclusive or, absolute value, add, subtract, multiply, and divide.

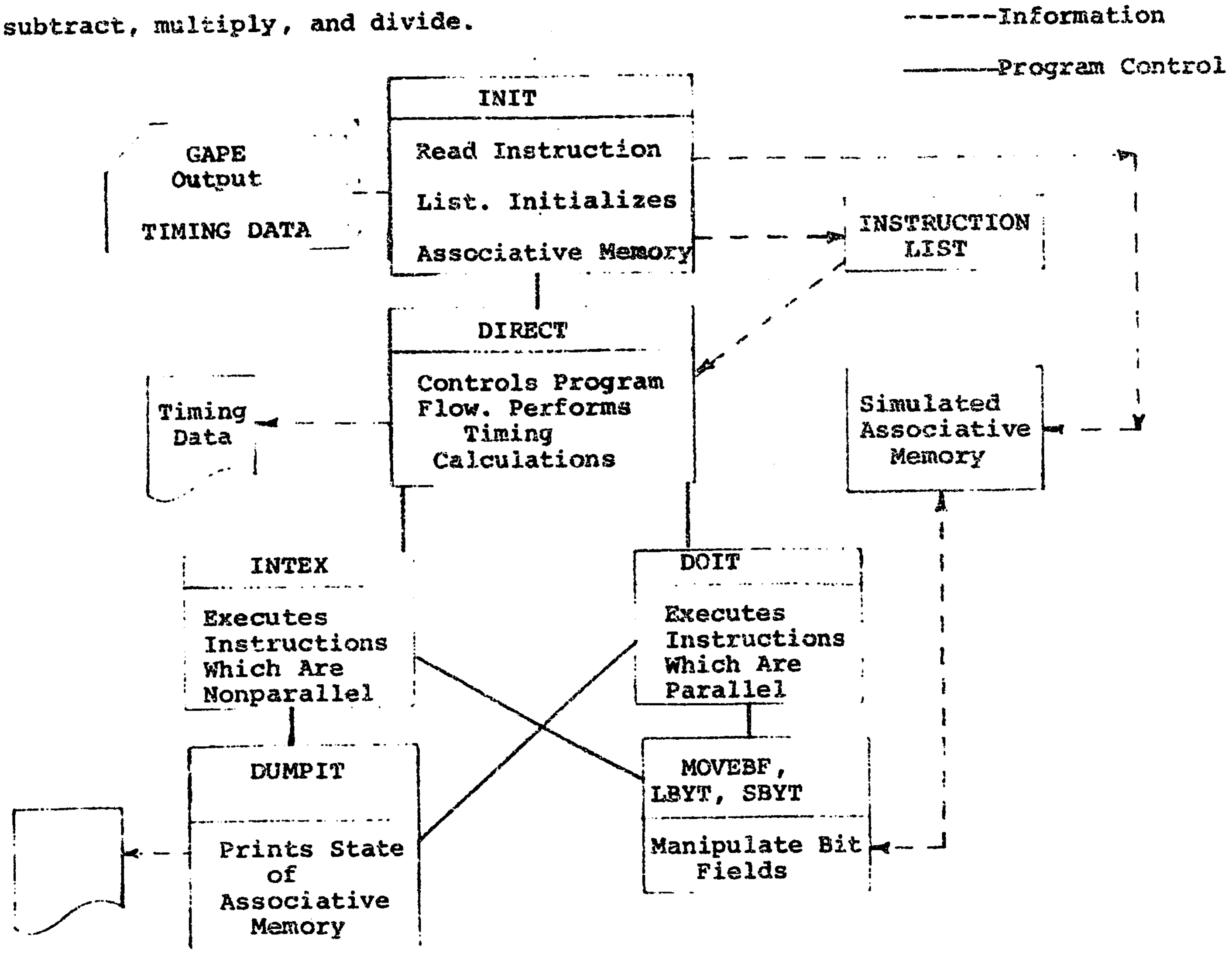
------Information
———Program Control
INIT
Read Instruction List. Initializes Associative Memory
GAPE Output
TIMING DATA
INSTRUCTION LIST
DIRECT
Controls Program Flow. Performs Timing Calculations
Timing Data
Simulated Associative Memory
INTEX
Executes Instructions Which Are Nonparallel
DOIT
Executes Instructions Which Are Parallel
DUMPIT
Prints State of Associative Memory
MOVEBF, LBYT, SBYT
Manipulate Bit Fields

Figure 4

APEX Logic Relationships

Subroutine DOIT extracts the bit fields for each operand (A,B) from each word of associative memory; performs the requested operation, and stores the result in the specified field (S) in each associative memory word.

The other instruction type makes less use of associative memory. Instructions of this type manipulate the arithmetic register, output register, field pointers, or date registers. These instructions may also shift the response store values, control program flow, provide input/output, etc. This second type of instruction is executed by subroutine INTEX. This subroutine contains a small internal subprogram for each instruction of this type. As the instruction is encountered, the internal subprogram manipulates the AP registers in the appropriate fashion.

The use of DOIT and INTEX is controlled by subroutine DIRECT. This subroutine controls the selection of instructions and the interpreting of the command words and parameter identification words of the language produced by GAPE. Subroutine DIRECT also maintains the instruction execution frequency and timing information data.

Subroutine DUMPIT prints the state of the associative array on command. Subroutines MOVEBF, LBYT, and SBYT perform bit field manipulations in the simulated associative array.

CONFLICT PREDICTION

The prediction function is responsible for identifying (1) all pairs of aircraft that are in hazardous positions; and (2) all aircraft flying too close to the terrain. The basic design of this module was inspired by the implicit geometric filter concept described in the paper "Intermittent Positive Control" in the March 1970 issue of the Proceedings of the IEEE.

The airspace is divided into 1024 horizontal square cells, or "boxes", each 4 miles on one side. This grid covers a square area 128 miles to a side. Each aircraft is entered into a number of boxes according ot its location, speed, and direction of flight. The method of placement in cells is as follows:

1. Place the aircraft in the cell containing its present position. (This is the primary cell.)
2. If the aircraft is within 1/2 the minimum safe separation distance of a cell edge, place the aircraft in the adjacent cell. (This is a positional secondary cell.)
3. Place the aircraft in cells (called velocity secondary cells) according to three indices:
   a. speed class
   b. cell quadrant position
   c. heading

The speed class is an index to the aircraft's indicated ground speed. The cell quadrant position is the quadrant within the cell within which the aircraft lies. The heading is an index derived by dividing the compass into twelve equal sized (30 degree) sectors. This placement of the aircraft into adjacent cells to allow for flight during the warning time is performed by a table look-up on speed class, cell quadrant position, and heading. The table contains cell displacements to calculate the new cell indices from the basic cell index. The construction of the table was performed off-line to provide for fast real-time execution. The table considers the tracking errors.

After all aircraft are entered into the cells, the conflicts between these aircraft are selected. The box size and placement method are such that if an aircraft is the only cell occupant, there can be no conflict. If more than one aircraft occupies a cell, further tests are required to determine the hazard. (These tests are described later).

Terrain avoidance is performed during the cell placement process. Each cell has a minimum altitude for safe flight. Only aircraft with altitude information can be checked for terrain avoidance. Before an aircraft with altitude information is placed into a cell, its reported altitude is checked against the acceptable minimum altitude. If the aircraft is too low, the appropriate information is output.

There are three further filters to be performed before an aircraft pair is output as a conflict. First, the software determines if this aircraft pair was selected as a conflict in the previous 60 seconds. If so, no further processing is done, as this pair has already been processed by display or resolution. The second test is an altitude filter. If both aircraft have altitude information and have reported greater than 500 foot vertical separation, then the conflict is ignored. The final test is a coarse hazard filter. The coarse hazard filter adds the time dimension to the prediction process and determines if the aircraft can violate the safety standard.

Because the aircraft are tracked, the software can determine if a near miss is predicted. It is known that the heading data provided by the tracker is subject to error. Thus, for each speed class and bearing position, there is a maximum error in the velocity components in the two horizontal directions. The coarse hazard filter will add these maximal errors to the predicted velocities to obtain the highest relative closing velocity. Then, a simple miss distance calculation will determine if the aircraft could pass within the minimum safe miss distance.

CONFLICT RESOLUTION

Once a possible conflict has been isolated by the prediction function, it must be further evaluated to determine its relative collision potential, or risk. Ordering the possible conflicts by risk allows the automated system to respond consistently to the priorities of the users in presentation of warnings. The chosen measure of risk is the probability of violating a given miss distance within the warning time provided by the system. This probability is calculated from the geometric configuration of the two aircraft and the uncertainties inherent in their position and velocity data. The logic does not consider the conditional probability that an aircraft will turn from its current course, although future systems should utilize whatever "intent" information is available in the system.

Considering the aircraft's current position, velocity, and acceleration, it is possible to project an ensemble of possible paths which the aircraft could follow. The uncertainty associated with the choice of a path from this ensemble arises from two sources: variances in the current data, and uncertainty about the pilot's intent.

Uncertainty in velocity (in particular, heading) is a major source of spread in the path ensemble. This uncertainty is approximated by the normal distribution of straight paths symmetrically projected about the estimated heading of a non-turning track. Uncertainties in position are accommodated in the miss distance criteria.

Lack of knowledge of the pilot's intent is another source of uncertainty in defining the path ensemble. If an aircraft turns within the projected time period, then the assumption of straight flight can result in a hazard suddenly appearing with less than 30 seconds to possible impact. If all possible paths are included in the ensemble, however, the volume of airspace occupied by the ensemble grows, and data must be available to define the probability distributions of turning paths. The problem of turning aircraft is compounded by the fact that the tracking logic produces greater variances in current estimated heading, as well as time lags in heading prediction, when a turn is in progress. These considerations led to the assumption of a uniform distribution of headings in the direction of turn if a turn was determined to be in progress from the track data. The basis for the assumption was that if an aircraft were turning in the terminal environment, it was equally likely that it would continue turning, or stop turning at any point on the current trajectory (and proceed straight along a tangent to the turn curve). While this assumption is only a modest first approximation to a definition of the full path ensemble, it does reflect the broader distribution resulting from the turn in a realistic manner. Further development of the conflict resolution logic must examine the possibility of developing valid <u>a priori</u> probabilities of turn in the terminal airspace, perhaps as a function of aircraft position, wind patterns, or other variables. Future systems might incorporate such "intent" information from the other ATC functions in the data processor.

The risk probability is calculated by a numerical integration over the ensemble of possible paths of the two aircraft. The area encompassed by the potential paths is divided into a number of equal size segments, and a representative path selected for each segment. Each representative path has a probability associated with it. If the two aircraft paths result in a violation of the miss distance in the warning time, then the joint probability is summed into the risk probability. More explicitly, the risk probability becomes

$$\text{RISK} = \sum_i \sum_j p_i^a \, p_j^b \, \delta(D_{ca})$$

where p_i^a is the probability aircraft A traverses the ith path in the ensemble and $\delta(D_{ca}) = 1$ if the distance of closest approach for paths i & j is less than a critical distance, and 0 otherwise. A more responsive risk criteria function currently under investigation would additionally weigh each contribution of a conflicting path pair according to the time remaining to violate the separation criteria. This criteria becomes

$$\text{RISK} = \sum_i \sum_j p_i^a \, p_j^a \, \delta(D_{ca}^{ij}) \times W(T_{ca}^{ij})$$

where $W(T_{ca}^{ij})$ is a weighting function (0-1) which is a function of T_{ca}^{ij}, the time of closest approach for paths i and j.

An aircraft configuration is considered hazardous if the risk probability is greater than a threshold value. According to the Knoxville experiment requirements, certain configurations required calculation of a maneuver which would eliminate the conflict. This maneuver was displayed to the controller for transmission to the pilot.

The aircraft are divided into two types, associated and unassociated. Associated aircraft are under direct positive control of an ATC controller (Instrument Flight Rules). Unassociated aircraft are not under positive control (Visual Flight Rules). For conflicts between one associated and one unassociated aircraft where both aircraft are reporting altitude information, a maneuver is calculated for the associated aircraft. For other types of conflicts, the controller is merely alerted to the existence of the conflict.

Note that when altitude information is available, the risk probability has been calculated on the basis of the three-dimensional position and velocity vectors. If a leveloff command will reduce the conflict probability below the threshold value, the recommended maneuver is "Level Off". However, if that probability is greater than the acceptable safety threshold, a turn maneuver is calculated. Since most of the aircraft in the system do not have altitude reporting transponders, positive climb and dive maneuvers are not generated. If a lateral maneuver is required, the associated aircraft is turned away from the unassociated aircraft until the distance of closest approach is greater than the allowable miss distance. The paths used to calculate this turn are selected from the ensemble of paths of the aircraft on the basis of shortest time to conflict.

The direction of turn may be determined by considering the two aircraft as a physical system and locating the positional centroid of this system at the time of the expected turn. The maneuvering aircraft is turned away from this centroid. Turning the aircraft towards the centroid may, in some cases, produce less severe maneuvers. However, in a system with uncertainties in aircraft position, velocity, and time of maneuver initiation, a turn toward the centroid often increases, not decreases, the risk. The time at which the relative location of the centroid, and therefore the advisable direction of turn, changes is different for any two aircraft paths. Thus the ensemble of paths around each aircraft generates a spectrum of these critical times. If, within the time span of interest, different paths have different advisable turn directions, the maneuver becomes ambiguous. This ambiguity can be discovered by inspecting the edges of the ensemble of paths. If a maneuver is ambiguous in this sense, the resolution algorithm indicates this on the display and does not recommend a resolution.

To decide how far the aircraft should turn, different degrees of turn, at the standard rate, are projected. The distance between the aircraft pair at the end of each trial turn is determined in order to insure that the aircraft do not violate the minimum miss distance while the maneuvering aircraft is turning. The path tangent to the turn circle is then checked against the unassociated aircraft's worst case heading vector (appropriately projected in time) and the distance of closest approach is calculated. If this distance, the closest that the two aircraft will ever get to each other if neither deviates from the given course, is greater than the minimum miss distance, then this is considered to be a feasible maneuver. The two new vectors then have their risk probability calculated. If this probability is less than the threshold, then the maneuver is accepted. If the probability is greater than the threshold, then the aircraft is turned further and checked again. This last step is repeated until a safe maneuver is found. No turn greater than 180° is considered.

Tables are maintained to determine if an aircraft gets into multiple conflicts. If so, consistent maneuver suggestions are calculated. If the multiple conflict occurs from the same side, then the larger of the bearing changes will be sent as the maneuver. If the conflict is from the opposite side, then a flag is set to indicate that no unambiguous maneuver was found.

ADVISORY MESSAGE GENERATION

The newest function added to the simulator is a simple module which prepares messages which could be transmitted to pilots involved in conflicts. A similar function was implemented and tested at Knoxville in 1972.

The messages consist of traffic advisories and service messages. Traffic advisories warn an aircraft of the location and heading of aircraft which could be in conflict. Service messages warn of restricted areas, terrain conflicts, loss of radar contact, etc.

The simulation module merely prepares a message in a form which could be sent to an automatic voice response unit for transmission to the pilot. In general, the time required to send an advisory will exceed the time for one antenna scan. The average message will take two or three scans to transmit. During peak periods, there will be messages waiting in a queue to be sent to the aircraft. The message generator must choose the order in which the messages are transmitted.

The basic structure of the system allows for a flexible priority scheme for message transmission. There are defined classes of messages which are assigned different priorities based on selections made by the controller. In addition, within these classes, priorities are established as appropriate. For example, the risk level provides a good priority measure within the traffic advisory class. Thus, the most hazardous conflicts will receive the first messages. Other messages, such as "all clear of traffic" are given priority based upon time in queue.