HARDWARE-IN-THE-LOOP TESTING OF KEW FLIGHT PROCESSORS

SPECIAL TECHNICAL REPORT

REPORT NO. STR-0412-90-009

March 16, 1990

GUIDANCE, NAVIGATION AND CONTROL
DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Contract Data Requirements List Item A004

Period Covered: Not Applicable

Type Report: As Required

DISCLAIMER

DISCLAIMER STATEMENT - The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

(1) DISTRIBUTION STATEMENT - Approved for public release; distribution is unlimited.

(2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227 - 7013 October, 1988.

HARDWARE-IN-THE-LOOP TESTING OF KEW FLIGHT PROCESSORS

March 16, 1990

Author

C. O. Alford, R. T. Abler, W. S. Tan, A. H. Register

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Eugene L. Sanders

USASDC

Contract Monitor

Cecil O. Alford

Georgia Tech

Project Director

OUTLINE

1. INTRODUCTION

1.1 Simulation Overview

The Computer Engineering Research Laboratory at the Georgia Institute of Technology, under contract to the United States Army Strategic Defense Command, is developing special purpose parallel computers for hardware–in–the–loop simulation and testing of kinetic energy weapons (KEW) systems and components. Of primary interest is the ability to test guidance, navigation and control (GN&C) algorithms. This paper presents details on a hardware–in–the–loop test of a Honeywell (Sandia) S5, GN&C processor. The simulation emphasizes object processing and assumes information from the seeker sensor has been processed to produce object centroids. The objectives of the simulation were to:

1. Demonstrate hardware–in–the–loop testing with real GN&C technology.
2. Demonstrate real–time simulation capability using ADA software.
3. Assess the performance of the simulation processor relative to the GN&C processor.
4. Demonstrate the effectiveness of the SUN 386i as a user interface (host computer) to the simulation machine.

This report presents the simulation development in Section 2, the simulation model in Section 3, simulation hardware in Section 4, simulation software in Section 6, the GN&C processor in Section 7, and the HWIL test and conclusions in Section 8. Source code for the test is given in the Appendix, Section 9.

2. DEVELOPMENT STAGES

This simulation required the development of hardware and software. These two were developed in stages. The overall simulation is rudimentary since it is the first step in a much more involved program. The objectives were to get all the hardware units running in a real HWIL configuration. Enhancements will be added as this project progresses.

2.1 The simulation

The simulation scenario was a single exo–atmospheric interceptor in the final stages of selecting an incoming ballistic missile, tracking it, adjusting course and destroying it by a kinetic energy impact.

The simulation begins with the interceptor using an infrared focal plane array (FPA) sensor at 100 frames per second to detect possible targets. Initially no targets are visible, so the interceptor proceeds on the current trajectory until a target is visible. When targets become visible for several frames the interceptor begins calculating the trajectory of the targets. At some point, a particular target is selected, and the interceptor adjusts its course in an attempt to collide with the target.

During the simulation, two graphics windows on the host are presented for observation. The first window illustrates the view of the target space as seen by the FPA. The second window is the view of an observer sitting at a some point in space. The observation location can be selected in a configuration file. Information on range to the nearest target, and which frame is currently being displayed on the host are also shown by the host.

The course corrections continue until the KEW interceptor "hits" the target, or passes by. The simulation returns a nearest miss distance. The nearest miss distance may actually be a collision if the miss distance is below a certain adjustable threshold. If the simulation detects a collision, suitable display effects on the graphics output indicate the collision event.

Figure 1. shows the graphics screen on the Sun host computer at an early stage in the simulation. In the *World View* window, the targets are on the left, and the KEW interceptor is on the right. The *FPA View* window shows eight targets seen as point sources by the KEW interceptor. The windows on the upper right were used to start the simulation, and to take a snapshot of the screen. Figure 2. shows the end of the same simulation The course corrections performed by the interceptor are too small to appear as any deviation from a straight line, since a pixel on the display actually covers 200x200 meters. The center to center miss distance as shown in the World View is 0.24 meters, counting as a hit. The "Non Real–Time Simulation: 0.680 Sec Lag!" indicates that the simulation did not run in real time. This was caused by the host, which for this simulation, displayed every frame, slowing the simulation.

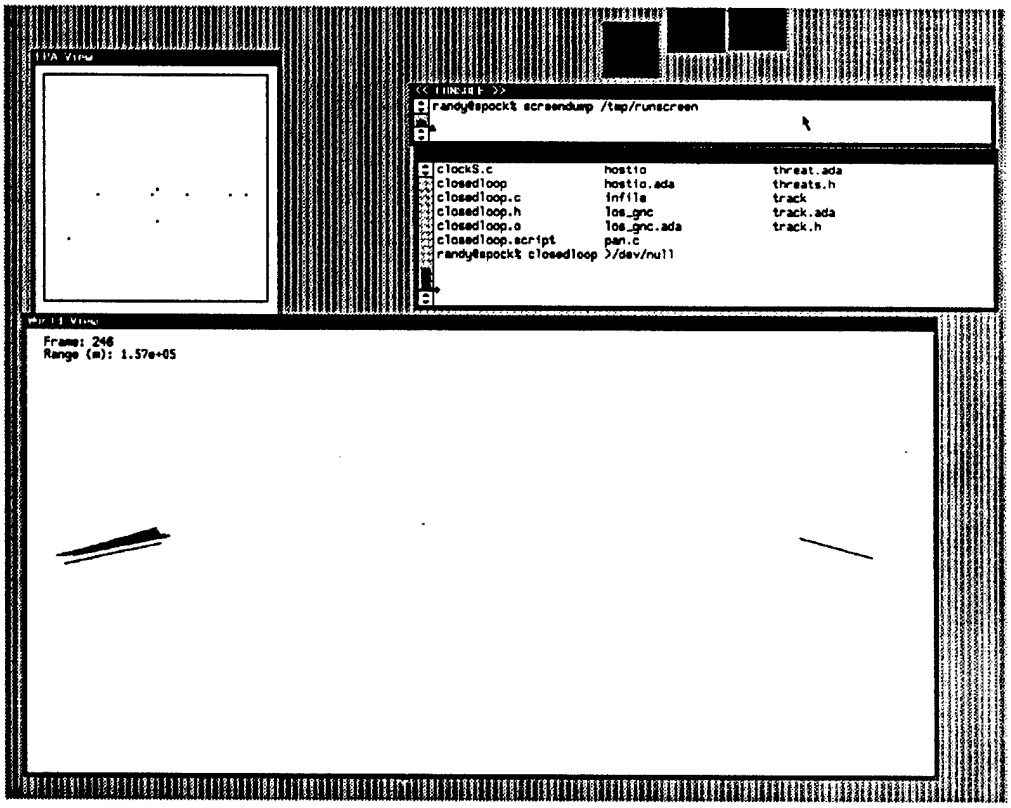

Figure 1. Beginning of Simulation

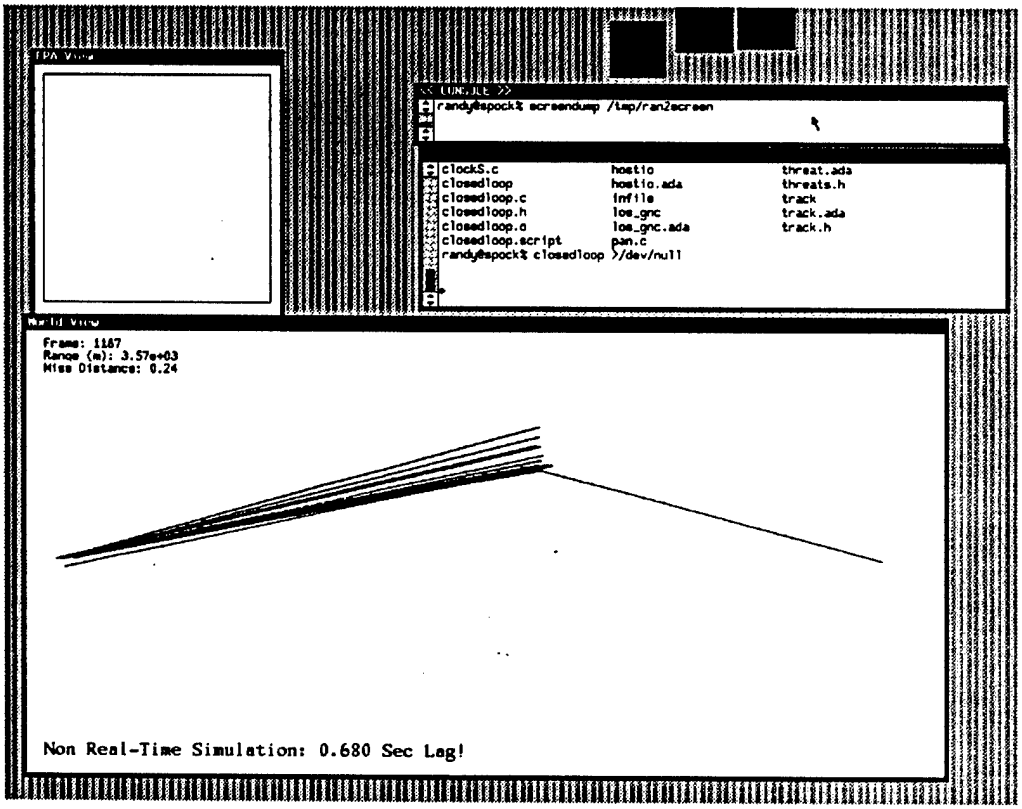

Figure 2. End of Simulation

2.2 Host only

The simulation was initially begun on a Sun 386i. This was convenient since the Sun 386i was an available platform maintained on a network at Georgia Tech. One of the networked Sun 386i's was also the eventual host for the PFP, minimizing the problem of transporting code between this stage and later stages.

C was chosen as the development language, because the Sun is a Unix platform and strongly supports C with debuggers and windowing libraries. The various algorithms that were used had to be collected and converted to C. Some of this code was originally in Occam, the Inmos Transputer's native language. Other pieces had been developed on PCs in Turbo–Pascal. These languages were all converted to separately compiled modules in C. The conversion was done with a plan to eventually run these algorithms on a parallel machine. This required the modules to be developed in such a way that they would easily map to one processor on the target machine.

Since the target architecture does not support shared memory, all shared variables must be passed as messages to other processors that use the data. While writing the modules, the code was developed with carefully constructed global structures. These structures are the elements that eventually must be transmitted over the communication network on the parallel target machine. All other variables were kept local to the module which used them.

2.3 PFP

The target machine is a parallel computer designated as a Parallel Function Processor (PFP) (see Section 4). The C code was modified to run on the Parallel Function Processor (PFP). This required replacing all of the global data structures with the send() and receive() calls used on the PFP. In conjunction with the code development, crossbar code was developed to link the sender to the receiver at the appropriate time. This has to be done carefully as there is no consistency checking to assure that the receiver gets data from the desired source. An incorrect crossbar program will simply pass the wrong data to a processor, or not pass data to it at all. Reading from a processor that is not sending will cause the entire crossbar communication network to deadlock. Debugging tools to handle this situation were developed in conjunction with debugging the code.

The host based simulation modeled only one target due to processing speed limitations. Given the parallel architecture and higher performance of the PFP nodes, multiple targets could be modeled on the PFP by replicating the target generation module, and modifying the crossbar code to deal with multiple targets. The simulation continuously passed data back to the host, but this quickly became the limitation on execution time. The host system is Unix based, and is therefore notoriously inconsistent in how fast it can handle data. To overcome this problem while still sampling as much data as possible, an additional buffer module was added to the simulation which collected data samples every simulation cycle (100 Hz). The host signalled this module when it was ready to receive a data sample. Other data packets were lost. To allow full data collection, the buffer module was modified to run either in a demand mode as before, or to force every N^{th} sample to be passed to the host, delaying the simulation until the host was ready for data.

The code was then converted on a module by module basis to Ada. Each module was re–written in Ada, and its results (with the rest of the system running C based code) were compared to the results of the C version of that module.

2.4 HWIL

The final stage of the simulation was to replace the module simulating the object tracking with an off the shelf flight processor. This processor was connected in place of one of the modules on the

PFP. A compiler could not be obtained for the flight processor due to licensing delays. The solution to this was to send the C code for the module to the vendor for the flight processor. They compiled the code and returned the loadable program. Since the code had to be sent back and forth via overnight mail, turnaround time was approximately one week.

A problem developed with the data ordering in the structure between the PFP and the flight processor. To avoid the long turnaround delay of re–compiling the flight processor code, this was solved by inserting an additional module in the simulation which re–ordered the data for the flight processor.

3. SIMULATION MODELS

3.1 FPA Model

The KEW's Focal Plane Array (FPA) is a fixed position, 128x128 pixel, infrared detector. A full FPA model would contain optical distortion, detector sensitivity, A/D quantization error and other noise sources. For the purpose of this simulation, which was constructed as a demonstration of the PFP's capability, the FPA model is combined with the signal processing to produce a very simple model.

The signal processing section of the KEW's avionics recognizes non–overlapping objects in the FPA's field of view, and returns the pixel column and row of the centroid of the object (x_c,y_c), the pixel column and row of the intensity centroid (x_{ci},y_{ci}), the total size in pixels of the object (N_c), and the total intensity of the object (I_c). These are calculated using the following

$$I_c = \sum_{i=1}^{i=N} I_i \quad , \quad x_c = \frac{\sum_{i=1}^{i=N} x_i}{N} \quad , \quad y_c = \frac{\sum_{i=1}^{i=N} y_i}{N} \quad , \quad x_{ci} = \frac{\sum_{i=1}^{i=N} I_i \cdot x_i}{I_c} \quad , \quad y_{ci} = \frac{\sum_{i=1}^{i=N} I_i \cdot y_i}{I_c} \quad .$$

For this simulation, the position of the target was directly mapped into the pixel cordinates (x_c,y_c). The total number of pixels (N_c) was calculated based on size and range, and the total intensity (I_c) was calculated based on range and target intensity.

3.2 Target Model

The target was modeled as a uniform temperature body, resulting in the area centroid being identical to the intensity centroid. The target model allowed the thermal radiation intensity and the size of the target to be set on a per target basis, allowing multiple size and intensity targets in one simulation.

3.3 KEW Model

The KEW is modeled as a simple rigid body with an orientation vector and a velocity vector. The orientation vector is normal to the FPA. The velocity vector determines the flight path of the KEW. Two types of compensation are applied to the KEW during the homing phase of the flight. The first type of compensation is to orient the KEW such that the designated target is at the center of the FPA. The direction of rotation is proportional to the location of the target in the FPA. The second type of compensation is the divert maneuver of the KEW. The divert is proportional to the predicted velocity of the target on the FPA.

3.4 Flight Processor

The flight processor ran a tracking filter based on a Kalman filter. This is only a small portion of what an actual flight processor would have to run, but was sufficient to stress the example flight processor we were using for this demonstration.

4. PARALLEL FUNCTION PROCESSOR

4.1 System Compents

A Parallel Function Processor (PFP) is a parallel computing array consisting of the following components (see Figure 3.):

Host Computer with the following peripherals

- Terminal
- Printer
- Removable Hard Disk Drive

One or two Crossbar Switches (16x16 array)

Sequencer (crossbar switch controller)

PPEs (32 processor boards per crossbar)

These components are embedded in a chassis with Multibus I card cages, power supplies and fans. In addition there are display boards to indicate the crossbar switch pattern and the processor status.

The PFP can be expanded as shown in Figure 4., to include 64 PPEs. In Figure 4., the components which have been added are a crossbar switch, sequencer and 32 parallel processing elements. This system is essentially a duplicate of the components in Figure 3., except for the Host Computer.

In Figure 4. there are actually two parallel computing arrays which can operate independently on separate problems, or together on a single problem. When operating together, an Array Interconnect Module is inserted into a PPE port to provide a connection between the two units. More than one Array Interconnect Module can be used, but each connection causes the loss of two PPEs. For higher bandwidth four interconnection modules are used to replace four PPEs, providing two links between the PFP units. The hardware is packaged in three 19" racks except for the host and printer.

4.2 Host Computer

The sequencer, crossbar switch and processors all reside on a common computer bus (Intel Multibus). This bus is controlled by a host computer which is used to download the sequencer, crossbar switch and processors with their necessary microcode or program. Previously an Intel 310 computer (80286 based) was used as the host for all code development.

A SUN Microsystems 386i workstation is currently used as the host computer. The SUN 386i host is connected to the PFP Multibus using a BIT3 card set which converts between

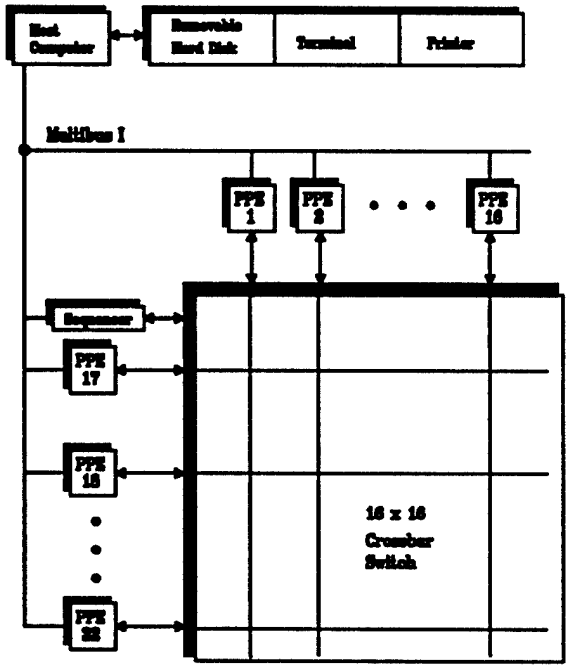

Figure 3. Single Array

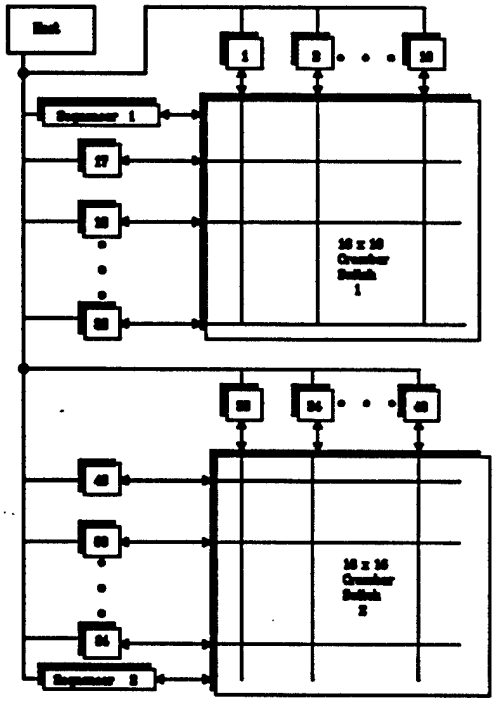

Figure 4. Double Array

the Sun's internal PC–AT style bus and the Intel Multibus.
The SUN host is used to compile, link and load programs to the processing elements, the crossbar sequencer and the crossbar memory. The SUN 386i host provides many features not available on the Intel 310 host such as graphical user interface (block diagram editor), remote login capability, and an integrated parallel programming framework for the development of Ada programs.

4.3 Sequencer

The Sequencer is the master state machine of the computer. The responsibility of the sequencer is to clock the crossbar switch for the appropriate communication configuration and to control the handshake lines of each processor. The microcode of the sequencer, stored in a 96 x 1024 bit memory, defines the sending and receiving processors and the next cycle address where the sequencer and crossbar memory can retrieve information. The next cycle address allows both the sequencer and crossbar to run in a pipelined mode, removing memory read time from execution delay.

Once the sequencer is loaded with the microcode for a problem, it is started by writing a control word to the I/O control port of the sequencer. Immediately the sequencer will load the information for the first cycle of the communication configuration. The sequencer will check the handshake lines of the processors and wait till all the senders and receivers, for that particular cycle, are ready. When this condition is met, the sequencer transmits handshake signals to the active processors to latch data in and clears handshake lines for the next cycle. When the sequencer is started, the crossbar switch is also enabled and the switch configuration for the first cycle is loaded. At each cycle the sequencer uses the next sequencer address to fill the pipeline for the next cycle. The sequencer also transmits the next crossbar address so the crossbar pipeline will also be ready for the next cycle. Each cycle is synchronized by waiting for all active sending and receiving processors to become ready. The sequencer will continue to run until it is reset. For most problems, the sequencer will stop at a certain cycle when the processors finish their tasks and no longer generate handshake signals to the sequencer.

4.4 Crossbar Switch

The crossbar switch makes up the second component of the communication network. It consists of four circuit boards, where each board is a 4x4, 16 bit bi–directional switch. The crossbar switch is physically arranged so that there are horizontal "x" buses and vertical "y" buses. There are sixteen "x" buses connected to sixteen "y" buses. Each circuit board has memory to store control bits. The memory stores 32 x 1024 bits of microcode, which defines the control for each switch on the circuit board. There are three possible states for each switch: "x" to "y" transfer, "y" to "x" transfer and tri–state for no transfer. Each circuit board depends on the sequencer to provide the next crossbar address to retrieve control bit information from the microcode memory. Since the microcode for each circuit board is stored in the same order, the sequencer only provides one next crossbar address to all four circuit boards.

The crossbar switch is enabled by the sequencer when the proper control word is written to the I/O control port of the sequencer. Once enabled, the crossbar switch will load the switch configuration for the first cycle. Since the sequencer is the master state machine, it will clock the crossbar to use the next crossbar address to load information. Using the next crossbar address, the crossbar switch will set the switch configuration before the sequencer begins checking the handshake lines of the processors. The crossbar switch maintains a particular switch setting until the sequencer clocks it or the crossbar switch is reset.

4.5 Parallel Processing Element

A Parallel Processing Element (PPE) must meet two criteria to be used in the PFP. First, the processor must meet the Multibus standard to provide power and an interface for the host processor. Second, the processor must meet the crossbar interface standard (proprietary standard for the computer) to provide handshake signals to the sequencer and a data bus to the crossbar switch.

A custom floating point processor (GT–FPP or FPP) designed at Georgia Tech was used for this simulation. The FPP processor is a custom processor board designed at CERL for use in the PFP. The FPP is a three address machine, using the AMD29325 32 bit floating point ALU, in conjunction with dual port memory, and an AMD 2910 sequencer.

5. SIMULATION INTERFACE

5.1 Host

A Sun 386i was used as a host computer for the PFP. The Sun 386i uses a 25 MHz 80386 CPU with the 80387 floating point co–processor, 12 MB main memory, a 307MB Hard disk, and a 1152x900 pixel color monitor. The host system supports:

1. FPP processor program compilation,
2. crossbar sequencer program compilation,
3. FPP and crossbar sequencer program loading,
4. all program storage, either online (hard disk) or offline (floppy disk or tape),
5. debugging access,
6. network access to the PFP,
7. graphical output display,
8. windowed user interface, and
9. data and simulation result collection.

5.1.1 Host to Processor Element program loading

The processor boards are accessible from the host via a multibus repeater array. The processor selects a specific multibus rack within the PFP. Within a given rack, each processor is assigned a specific 64kB memory range. To select a specific processor, this memory range is mapped into the host memory map. The device drivers on the host are written such that this is handled automatically whenever a specific processor is accessed. Once a processor is mapped into the host memory space, the code is loaded by direct memory access from the host.

For the FPP, the memory range contains the program memory, a control register, and the two FIFOs for communication with the processor. Since the FPP employs separate address and data spaces, and the data memory is not accessible by the host, it is necessary to first load and run a data memory loader. The data memory loader takes data from the host to FPP communication FIFO and stores it in data memory. The actual program code is then loaded to the processor's instruction memory.

The control register allows the host to start and stop the FPP, and to check the status of the host to FPP and FPP to host FIFOs. The status flags identify when the FPP to host FIFO is empty, and when the host to FPP is full. The intent is to allow the host to easily identify if an FPP has data for the host, and to determine if the FPP's FIFO has room for host to FPP data.

5.1.2 Host to Processor Element runtime communications

The host to processor communication mechanism was designed for a simple host that acted only as a loader. With the Sun 386i's 1152x900 color graphics monitor, the host can play an important part in a simulation. This brings up the problem of transferring data from the processors to the host in a system that was designed to use the host strictly for loading programs. Since the host can read as well as write processor memory, a bi–directional communication capability exists. The FPP was built to capitalize on this by using two memory mapped FIFOs to allow buffered messages to be passed between the host and the processor.

No mechanism for passing interrupts between the host and processor exist, requiring a polling mode of data transfer between the two. The host is the master in any transfers, since the PPE does not have

any method of initiating transfers to the host. In sending from a processor to the host, the processor loads data into a processor to host FIFO. The host selects the processor then polls the status register to determine if there is new data present in the FIFO, and if so can read the data from the FIFO. Host to processor communications do not require polling by the host. The host selects the target processor, checks that the FIFO is not full, and writes the data to the host to processor FIFO.

A Unix *device special file* acts as the access point for the two FIFOs on each processor. This allows the standard unix syntax of opening, reading and writing to be used for accessing each processor. The device driver handles all of the FIFO status checking, and uses the standard Unix error handling routines. Programs on the host can then use all of the Unix I/O libraries for accessing the FPPs.

5.1.3 Host Network Interface

The host is connected to the local ethernet. This allows the host to use the full range of network features available on modern Unix implementations. Remote tape drives are used for backing up the data files on the host, remote access to laser printer for printouts, and file sharing via NFS to remote servers. The most important feature of the network access is the ability to work on the PFP from remote hosts. This allowed the simulation to be developed by several people from multiple workstations. This was important since the group supporting the software tools reside in a separate building, yet they could work on the system over the network from their facility. The simulation produced graphical output that was developed under Sun Microsystem's SunView which does not allow networked graphics windows. This required the simulation to be run on the host computer when graphical output was desired. Future use of the X11 window system will allow remote graphical display.

6. SIMULATION SOFTWARE

The parallel architecture of the PFP and the use of custom hardware designs requires custom software tools to program and debug any simulation. The software tools for programming the PFP are being developed at Georgia Tech with the aid of the School of Information and Computer Science.

6.1 Support Software

Support software under development at Georgia Tech consists of compilers and assemblers for the FPP and other PPEs, a crossbar compiler, and miscellaneous debugging tools.

6.1.1 FPP C Compiler

The FPP C compiler is based on the Portable C Compiler. The FPP is a difficult processor to implement C on since it is not capable of manipulating or addressing 8 bit quantities. The FPP supports only two data types, *int* and *float*, and pointers to these types. Note that the type *char* is not supported, and therefore, strings are not supported. The limited memory size of the FPP prohibits large programs from being loaded into the FPP. Communication routines have been added to the C library for the FPP to support communication to the host and to the crossbar.

6.1.2 Ada Compiler

An initial Ada compiler was developed for the FPP processor. The output of the Ada compiler is C code, therefore it is actually an Ada to C translator. The translator mechanism simplifies adding new processors as processing nodes to the PFP, since C compilers are in general more readily available for any processor. The Ada compiler only supports a Pascal like subset of the Ada language. Tasking is not supported by this compiler. The Ada compiler supports a one program per processor model, so a separate program has to be written for each processor, except where identical code can be repeated on multiple processors. The internal compiler is currently being replaced by a commercial Ada to C compiler to avoid the development cost and difficulties associated with writing, testing, and validating an Ada compiler. Communication routines have been added to the Ada library for the FPP to support communication to the host and to the crossbar.

6.1.3 Macro Assembler and Linker

The output of the C compiler is fed to an assembler in order to produce linkable code. This assembler uses a preprocessor to handle macro expansion then creates the microcode instructions for the FPP. A final link stage prepares the code for loading.

6.1.4 Crossbar Compiler

The programming model for a PPE has four channels. These are ;

1. 16 bit input from the host,
2. 16 bit output to the host,
3. 16 bit input from the crossbar, and
4. 16 bit output to the crossbar.

Since the processor sees only a crossbar output channel and a crossbar input channel, and no source or destination addresses, a careful job of coordinating the crossbar data transfers with the processor data is required to prevent data from getting delivered to the wrong destination.

A compiler is used to generate the needed crossbar code. The crossbar code also is used to assign modules to physical processors. At the beginning of the crossbar code, a definition line is used to

declare a name for a module and to designate a physical processor as the code destination. This name can be used throughout the crossbar code, and the host program. The definition is also used by the crossbar compiler to generate a loader program that will load all of the modules to the PFP when the simulation is run.

6.1.5 Tools

Miscellaneous software tools were used to support the simulation. Control programs for the crossbar sequencer allow the sequencer to be single stepped, or stepped slowly, allowing the operator to observe the status of the communication network as the simulation proceeds. The debugging tools for the FPP allow memory to be dumped to a file for low level debugging. Other programs examine the entire PFP and report the status of each processor, performing sanity checks, processor type checking, and processor network consistency verification.

6.2 Simulation Code

The simulation used three modules to generate the actual simulation, and one module to provide the interface to the host. Crossbar code was necessary to run the communication network. The host also had to be programmed to load the appropriate parameters for the simulation, and display the simulation results as the program ran. A block diagram of the program model is shown in figure 5.

6.2.1 Threat Generation Module

The threat module was responsible for creating the trajectory for one target. Multiple targets were created by using multiple threat modules on separate processor nodes. The target followed an exo–atmospheric ballistic trajectory. The threat module computed the current position for a single target. With data received from the rest of the simulation, the threat module then computed its relative position to the interceptor. Using this data, the projection of the target onto the FPA of the interceptor was computed. This projection was then transferred over the crossbar to the threat module.

6.2.2 Tracking Module

This module received the projection of each threat onto the FPA. It ran in two basic stages. Initially the tracking module did not provide any course corrections, but simply built up track files of the target position based on relative motion across the FPA. After a certain number of frames, the tracking algorithm started sending velocity corrections to the guidance module to intercept a selected target.

6.2.3 Guidance Module

The guidance module received the velocity corrections from the tracking module, and implemented these corrections within the maximum thrust constraint of the thrusters. The guidance module maintained the actual interceptor position, and transmitted this over the crossbar to the threat generation module(s) to close the loop for the simulation.

6.2.4 Host Interface Module

The host interface module received data from the threat generation module and the host guidance module for possible display by the host. Since the host processor does not provide sufficient speed to display the data from every simulation cycle, this module provided a throw–away buffer, where data not requested by the host was overwritten by the data from the next cycle of the simulation. This module had an alternate mode which forced the simulation to run slow enough for the host to receive every N^{th} frame, where N was a parameter that was downloaded from the host at the beginning of the simulation. This module attempted to format the data as much as possible to minimize the computation necessary on the host in order to provide maximum display update speed.

6.2.5 Host Programs

The host program provides the host side code for running the simulation and displaying the results. The host program initially calls a routine created by the crossbar compiler to load all of the code onto the PFP's processors. The host must then read a simulation data file and transmit all the initialization data to the appropriate processors. This allows several parameters of the simulation to be changed without recompiling the simulation by just editing this file and re–running the simulation. The host must then provide the graphic display of the data as the simulation runs. When the host is ready for a snapshot of the simulation, it sends a request to the host interface module on the PFP for the data from one simulation cycle. This data is displayed and checked for end of simulation. This continues until the end of the simulation, or the operator aborts the simulation.

6.2.6 Crossbar Code

The crossbar code is used to program the crossbar and crossbar sequencer. This code is responsible for assuring that the data is transferred between the correct modules. Running a different number of threats requires the crossbar code to be recompiled with a new parameter reflecting the new number of threats. Another parameter in this code causes the simulation to use the actual flight processor harware instead of running the track module on a PPE.

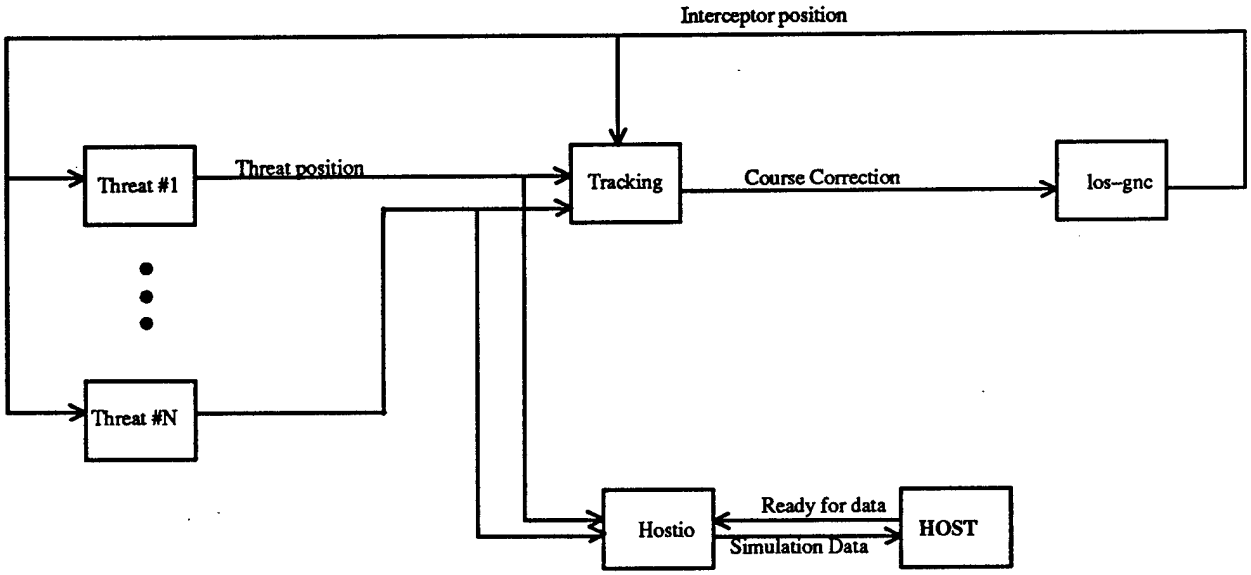

Figure 5. Simplified Program Block Diagram

7. FLIGHT PROCESSOR

This phase of the simulation was designed to demonstrate both the hardware–in–the–loop capabilities of the PFP and to benchmark the FPP hardware and software against a flight qualified GN&C processor. For this phase, a commercially available flight processor was chosen. A GN&C to PFP interface was designed and constructed. Finally, a simulation scenario was designed and coded.

7.1 Hardware

7.1.1 GN&C Processor

The Honeywell (Sandia) S5 was chosen as the flight qualified GN&C processor. Far from a bare–bones computer, the design of the S5 is entirely modular. Modular construction allows the system to be configured based on deployment objectives. Georgia Tech's S5 was configured with the following modules.

1. D PM–H117X Processor Module
2. D SI–H048X System I/O Module
3. D EM–H011X Expansion Memory Module
4. D UM–H039X Utility Module
5. D FF–H010X Mil–Std 1553 Bus Interface Module

For the simulation only the first two modules were used. The processor module is based on a 20MHz Motorola 68020. Included on the module are a Motorola 68881 floating point co–processor and 128kB of ram. The processor module is rated at 800 KFLOPs.
The I/O module contains a wealth of resources. Included on this module are five serial ports, a parallel port, timers, interrupt inputs, discrete inputs and discrete outputs. The parallel port was chosen as the interface port between the S5 and the PFP. The parallel port is essentially a slow extension of the S5's internal busses. Through this port, address, handshake, reset and data are all available for use by an external device. Using this port as the basis for the interface allowed the S5 to read and write data to the PFP interface through a single memory mapped location.

7.1.2 PFP Interface

One of the attractive features of the PFP is the ease with which devices can be added as parallel processing elements. The interface hardware is simply a protocol transformer between a device port and one of the PFP sequencer/crossbar ports. The PFP sequencer/crossbar protocol is defined by the protocol of the Advanced Micro Devices AM2950 (bi–directional I/O port with handshake). It is generally preferable to use the AM2950 when designing a PFP interface. A schematic of the interface is shown in Figure 6. The circuit uses two AM2950s, some buffer/drivers, three connectors and a single 20 pin programmable logic device (PLD). The schematic shows the ease with which any parallel device can be added as a PPE.

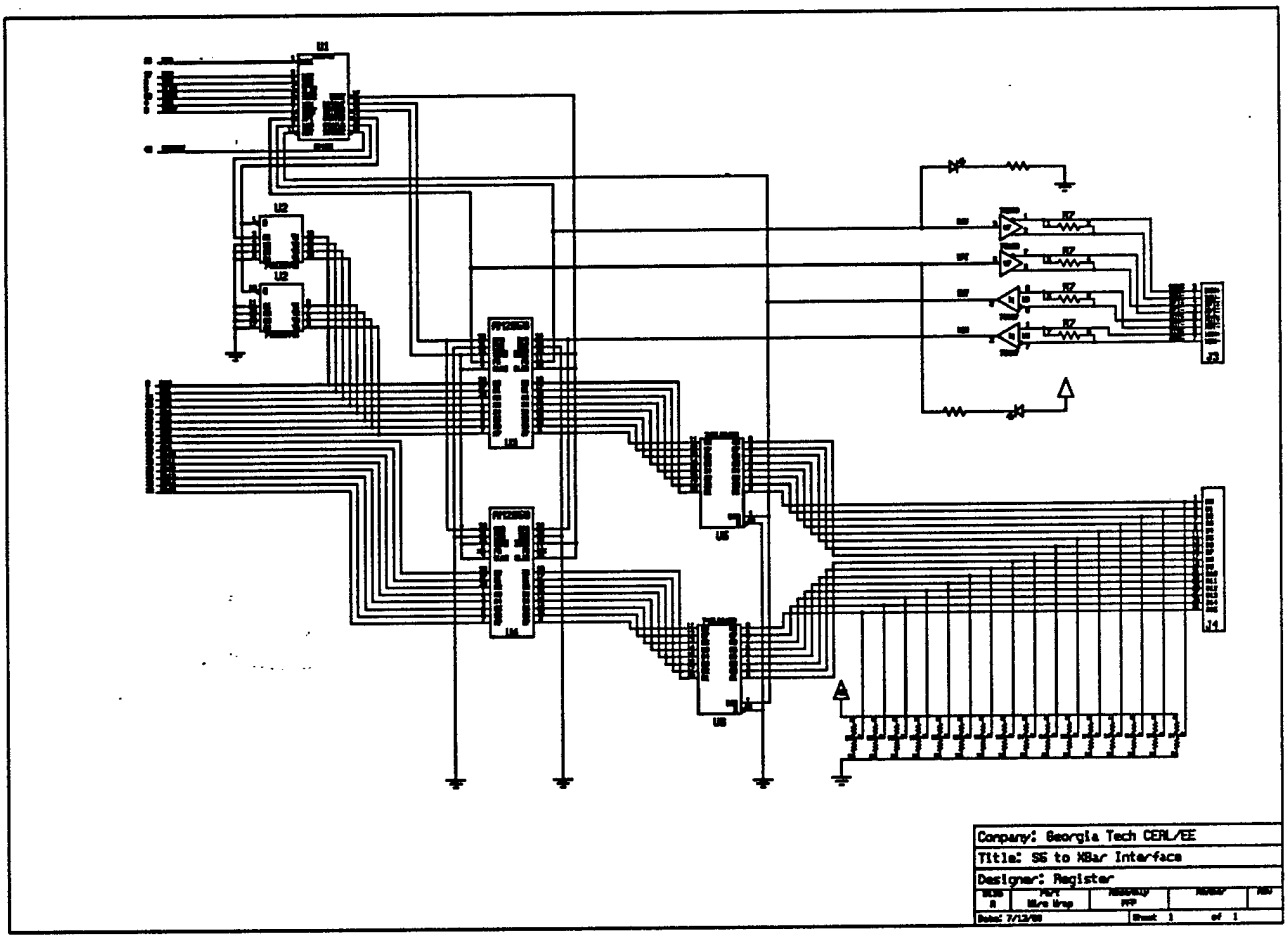

Figure 6. Flight Processor connection board

7.2 Software

Software for the S5 is broken down into three categories. The first is a description of the S5 Gateway control software. The second is a description of the compiler. The third is a description of the simulation program.

7.2.1 S5 Gateway

The laboratory setup for the S5 included an IBM PC–AT running the S5 Gateway software under MS–DOS. The S5 Gateway was provided by Honeywell and controlled all host interaction with the S5. The Gateway software is used to load and run programs. However, its most useful feature turned out to be the debugging environment. The debugging environment for the S5 is powerful. It allows a programmer to set a variety of different break–points and single step through instructions once a break–point is reached. A memory viewer that formats memory in a variety of different formats is also available. Memory can be viewed as (float), (int), (double), (long int), etc. The memory can also be "un–compiled" to show the assembly language instructions associated with a memory pattern. The contents of any memory location can be changed and an entire "snap shot" of the memory can be saved to disk for loading at a later time. The debugging tools were used extensively in the development of the simulation software.

7.2.2 Compiler

The recommended compiler for the S5 is the Oasys/GreenHill C compiler running on a microVAX under VMS. Due to licensing problems within the Georgia Tech legal department, Honeywell compiled the source code used for the simulation. The turnaround time for this was approximately one week. This made the rewrite–compile–run–debug cycle too long for more than a few iterations. Fortunately, the application code could be run and verified on the FPP prior to compilation at Honeywell. Even with this precaution, one bug in the S5 code occcurred. Since time constraints made another compilation cycle impossible, the debugging tools were brought to bear with good results. The bug turned out to be a tricky but interesting one. The problem manifested itself whenever the S5 sent (float) data to the interface. The procedure call to send a (float) is shown below.

```
float send_float(x)
float x;                        /* cast as 32 bit float */
 {
 unsigned short *ptWord;        /* unsigned short = 16 bits so pointer to a 16 bit quantity */
 ptWord = &x;                   /* assign a pointer to x */
 send_word( *(ptWord) );        /* send out first 16 bits of 32 bits */
 send_word( *(ptWord+1) );      /* send out second 16 bits of 32 bits */
 }
```

The data seen at the interface using this code turned out to be the two most significant words of a 64 bit IEEE real. As it turns out, all reals are passed in C as (double). The procedure is then supposed to cast the variable into the proper size based on the variable type defined after the procedure definition. The compiler in this case did not generate the code to change x into a (float). Thus, ptWord was pointing to a (double) instead of a (float). The C work around for this is easy. The corrected code looks like the following.

```
float send_float(x)
 float x;                       /* does not cast as 32 bit float COMPILER ERROR */
 {
 float NewX;                    /* new (float) variable */
 unsigned short *ptWord;        /* unsigned short = 16 bits sp pointer to 16 bit quantity */
 NewX = x;                      /* NewX is now (float)x */
 ptWord = &NewX;                /* assign a pointer to NewX */
 send_word( *(ptWord) );        /* send out first 16 bits of 32 bits */
 send_word( *(ptWord+1) );      /* send out second 16 bits of 32 bits */
 }
```

This C code has been compiled and verified.

The binary work around is a little more involved. The code could not be altered at the procedure level, but had to be altered at every call to the procedure. The change involved reworking the stack so that the (float) variable was pushed on the stack instead of the (double) variable. This change was possible because the instructions occupied the same bit lengths. The debugging tools and the compiler output files made the job of finding and correcting this bug relatively straight forward.

7.3 Programming

The algorithms that were run on the S5 were identical to the algorithms described Section 6. The difference is that the S5 algorithms were coded in C instead of Ada.

8. HWIL TEST RESULTS

8.1 Simulation

Because this simulation was intended to be a demonstration of the PFP as a simulation engine, the simulation avoided using detailed models which would have made the simulation more realistic but would have prevented the simulation from being used as a demonstration. Because of these limitations, the simulation results of the interceptor in themselves are not meaningful for any real problem.

8.2 Test Cases: Host Only

Running the simulation initially on the host was invaluable for porting the initial algorithms and for developing new algorithms. The multi–windowed environment with source level debuggers allowed the code to be edited, compiled, and debugged faster than could be done on the PFP.

Although running on the host was faster for the edit, compile and debug stages, the host was soon found to be too slow when the simulation actually had to be run. The simulation was run on the host with only one target. The host could only run the program at a 3 Hz frame rate, much slower than the 100 Hz real–time simulation.

8.3 Test Cases: PFP

When ported to the PFP, the code was already operational, so the major task was to develop the code for the communication network. The PFP provided the fastest environment for running the simulation. The PFP simulation used a timer algorithm to prevent the simulation from running faster than real–time speed of 100 Hz. The simulation would run real–time up to the maximum possible number of threats (26). In order to test the speed of the simulation, the simulation frame rate was increased to 160 Hz with 26 threats before real–time performance was lost. Analysis of the simulation revealed that the bottleneck at this speed was the time to transfer data to the host.

The buffering program would only send a data frame to the host when the host requested it, but once a data frame was requested, it had to be completely transmitted before the simulation could proceed. The time to send one frame to the host exceeded the cycle time when the simulation was run at over 160 Hz. At 100 Hz the PFP would probably be able to track 40 to 50 targets. Since the tracking algorithm is not linear in computational requirements with respect to the number of threat objects, it is difficult to estimate the exact cutoff.

8.4 Test Cases: S–5/PFP

With the simulation running on the PFP and the Sandia S–5 executing the tracking algorithm, the maximum number of threats that could be handled in real time was six threats. Attempting to run with more than six threats dropped the simulation out of real–time. This gives a real performance measure of this processor for use as a KEW flight processor, which is critical in today's atmosphere of inflated MFLOPS ratings.

9. APPENDIX

9.1 Crossbar code

```
#include "threats.h"

#ifndef    threats
#error     "#define threats <number>" missing from "threats.h" file
#endif

#define THREAT(N)  Threat##N is threat on p##N

#if USE_S5 == TRUE
     [Use_S5 == True]
#    define S5 y14
#endif

ClockS    is clockS   on p26
ClockR    is clockR   on p27

#if USE_S5 == TRUE
     [Use_S5 == True]
     Track      is s5_control     on p28
#else
     [Use_S5 == False]
     Track      is track     on p28
#endif

Los_gnc   is los_gnc on p29
HostIO    is hostio   on p31

     THREAT(0)

#    define      Threats_1 Threat0
#    define      Threats Threats_1

#if   threats > 1
     THREAT(1)
#    define      Threats_2 Threats_1 Threat1
#    undef       Threats
#    define      Threats Threats_2
#endif
#if   threats > 2
     THREAT(2)
#    define      Threats_3 Threats_2 Threat2
#    undef       Threats
#    define      Threats Threats_3
#endif
#if   threats > 3
     THREAT(3)
#    define      Threats_4 Threats_3 Threat3
#    undef       Threats
#    define      Threats Threats_4
#endif
#if   threats > 4
     THREAT(4)
#    define      Threats_5 Threats_4 Threat4
#    undef       Threats
#    define      Threats Threats_5
#endif
#if   threats > 5
     THREAT(5)
#    define      Threats_6 Threats_5 Threat5
#    undef       Threats
#    define      Threats Threats_6
#endif
#if   threats > 6
     THREAT(6)
#    define      Threats_7 Threats_6 Threat6
#    undef       Threats
#    define      Threats Threats_7
#endif
```

```
#if   threats > 7
      THREAT(7)
#     define      Threats_8 Threats_7 Threat7
#     undef       Threats
#     define      Threats Threats_8
#endif
#if   threats > 8
      THREAT(8)
#     define      Threats_9 Threats_8 Threat8
#     undef       Threats
#     define      Threats Threats_9
#endif
#if   threats > 9
      THREAT(9)
#     define      Threats_10 Threats_9 Threat9
#     undef       Threats
#     define      Threats Threats_10
#endif
#if   threats > 10
      THREAT(10)
#     define      Threats_11 Threats_10 Threat10
#     undef       Threats
#     define      Threats Threats_10
#endif
#if   threats > 11
      THREAT(11)
#     define      Threats_12 Threats_11 Threat11
#     undef       Threats
#     define      Threats Threats_12
#endif
#if   threats > 12
      THREAT(12)
#     define      Threats_13 Threats_12 Threat12
#     undef       Threats
#     define      Threats Threats_13
#endif
#if   threats > 13
      THREAT(13)
#     define      Threats_14 Threats_13 Threat13
#     undef       Threats
#     define      Threats Threats_14
#endif
#if   threats > 14
      THREAT(14)
#     define      Threats_15 Threats_14 Threat14
#     undef       Threats
#     define      Threats Threats_15
#endif
#if   threats > 15
      THREAT(15)
#     define      Threats_16 Threats_15 Threat15
#     undef       Threats
#     define      Threats Threats_16
#endif
#if   threats > 16
      THREAT(16)
#     define      Threats_17 Threats_16 Threat16
#     undef       Threats
#     define      Threats Threats_17
#endif
#if   threats > 17
      THREAT(17)
#     define      Threats_18 Threats_17 Threat17
#     undef       Threats
#     define      Threats Threats_18
#endif
#if    threats > 18
      THREAT(18)
#     define  Threats_19 Threats_18 Threat18
#     undef   Threats
#     define  Threats Threats_19
#endif
#if    threats > 19
```

```
    THREAT(19)
#   define Threats_20 Threats_19 Threat19
#   undef  Threats
#   define Threats Threats_20
#endif
#if  threats > 20
    THREAT(20)
#   define Threats_21 Threats_20 Threat20
#   undef  Threats
#   define Threats Threats_21
#endif
#if  threats > 21
    THREAT(21)
#   define Threats_22 Threats_21 Threat21
#   undef  Threats
#   define Threats Threats_22
#endif
#if  threats > 22
    THREAT(22)
#   define Threats_23 Threats_22 Threat22
#   undef  Threats
#   define Threats Threats_23
#endif
#if  threats > 23
    THREAT(23)
#   define Threats_24 Threats_23 Threat23
#   undef  Threats
#   define Threats Threats_24
#endif
#if  threats > 24
    THREAT(24)
#   define Threats_25 Threats_24 Threat24
#   undef  Threats
#   define Threats Threats_25
#endif
#if  threats > 25
    THREAT(25)
#   define Threats_26 Threats_25 Threat25
#   undef  Threats
#   define Threats Threats_26
#endif
#if  threats > 26
#error  P26 is currently used by an Clock Sender!
#endif

#if USE_S5 == TRUE
[Use_S5 == True]

#define delta_phi \
    cycle Los_gnc.2     --> Track Threats;  [delta_phi as float] \
    cycle Track.2       --> S5;             [delta_phi as float]

#define delta_theta \
    cycle Los_gnc.2     --> Track Threats;  [delta_theta as float] \
    cycle Track.2       --> S5;             [delta_theta as float]

#define pixel_x(N) \
    cycle Threat##N     --> Track HostIO;   [pixel_x as short] \
    cycle Track         --> S5;             [iAcoorx as float] \
    cycle Track         --> S5;             [iAcoorx as float]

#define pixel_y(N) \
    cycle Threat##N     --> Track HostIO;   [pixel_y as short] \
    cycle Track.2       --> S5;             [iAcoory as float]

#define S5_iIcoorx \
    cycle Track.2       --> S5;             [iIcoorx as float]

#define S5_iIcoory \
    cycle Track.2       --> S5;             [iIcoory as float]

#define cArea(N) \
    cycle Threat##N     --> Track HostIO;   [cArea as short] \
    cycle Track         --> S5;             [cArea as short]
```

```
#define cIntensity(N) \
     cycle Threat##N.2    —> Track HostIO;   [Intensity as int] \
     cycle Track          —> S5;             [Intensity as short]

#define cRange(N) \
     cycle Threat##N.2    —> Track HostIO;   [cRange as float] \
     cycle Track          —> S5;             [cRange as short]

#define x_dot \
     cycle S5.2      —> Track;          [x_dot as float] \
     cycle Track.2        —> Los_gnc;   [x_dot as float]

#define y_dot \
     cycle S5.2      —> Track;          [y_dot as float] \
     cycle Track.2        —> Los_gnc;   [y_dot as float]

#define phi_dot \
     cycle S5.2      —> Track;          [phi_dot as float] \
     cycle Track.2        —> Los_gnc;   [phi_dot as float]

#define theta_dot \
     cycle S5.2      —> Track;          [theta_dot as float] \
     cycle Track.2        —> Los_gnc;   [theta_dot as float]

#define init_s5 \
     cycle Track.2        —> S5;   [threat_intensity ]     \
     cycle Track.2        —> S5;   [dt ] \
     cycle Track.2        —> S5;   [TargetSelectionTime ]     \
     cycle Track          —> S5;   [Num_obj ]      \
     cycle Track.2        —> S5;   [FinalAcquisitionRange ]  \
     cycle Track          —> S5;   [TrackingRange ]

#define    Threat(N) \
           pixel_x(N)\
           pixel_y(N)\
           S5_iIcoorx\
           S5_iIcoory\
           cArea(N)  \
           cIntensity(N)    \
           cRange(N)\
           threat_coor(N)[0] \
           threat_coor(N)[1] \
           threat_coor(N)[2] \
           miss_distance(N)

[End of if Use_S5 == True]

#else [if Use_S5 == True]

[Use_S5 == False]
#define delta_phi \
     cycle Los_gnc.2      —> Track Threats;   [delta_phi as float]

#define delta_theta \
     cycle Los_gnc.2      —> Track Threats;   [delta_theta as float]

#define pixel_x(N) \
     cycle Threat##N      —> Track HostIO;   [pixel_x as short]

#define pixel_y(N) \
     cycle Threat##N      —> Track HostIO;   [pixel_y as short]

#define cArea(N) \
     cycle Threat##N      —> Track HostIO;   [cArea as short]

#define cIntensity(N) \
     cycle Threat##N.2    —> Track HostIO;   [Intensity as int]

#define cRange(N) \
     cycle Threat##N.2    —> Track HostIO;   [cRange as float]

#define x_dot \
     cycle Track.2        —> Los_gnc;   [x_dot as float]
```

```
#define y_dot \
     cycle Track.2          —> Los_gnc;  [y_dot as float]

#define phi_dot \
     cycle Track.2          —> Los_gnc;  [phi_dot as float]

#define theta_dot \
     cycle Track.2          —> Los_gnc;  [theta_dot as float]

#define    Threat(N) \
           pixel_x(N)\
           pixel_y(N)\
           cArea(N) \
           cIntensity(N)   \
           cRange(N)\
           threat_coor(N)[0] \
           threat_coor(N)[1] \
           threat_coor(N)[2] \
           miss_distance(N)

[end of if Use_S5 == True else clause]
#endif     [if Use_S5 == True]

#define delta_x \
     cycle Los_gnc.2        —> Threats;  [delta_x as float]

#define delta_y \
     cycle Los_gnc.2        —> Threats;  [delta_y as float]

#define missle_coor \
     cycle Threat0.2        —> HostIO;        [float]

#define threat_coor(N) \
     cycle Threat##N.2      —> HostIO;        [float]

#define miss_distance(N) \
     cycle Threat##N.2      —> HostIO;        [float]

#define lockstep\
     cycle ClockS           —> ClockR;        [dummy short]

#define Basic_Sends \
     delta_theta\
     delta_phi  \
     delta_x    \
     delta_y

[ ************************************************************************** ]

#if USE_S5 == TRUE
     [Use_S5 == True]
     [Initialize s5]
     init_s5
#endif

loop

[lock the step]
     lockstep
[basic_sends]
     Basic_Sends

[Threat 0]
           pixel_x(0)
           pixel_y(0)
#    if USE_S5 == TRUE
           [Use_S5 == True]
           S5_iIcoorx
           S5_iIcoory
#    endif
           cArea(0)
           cIntensity(0)
```

```
        cRange(0)
    [
    Missile Coordinate[n] is sent by Threat0 only
    Cycles shared with TrackObj sending to Los_gnc
    ]
        missle_coor[0]

        missle_coor[1]

        missle_coor[2]

        threat_coor(0)[0]

        threat_coor(0)[1]
        threat_coor(0)[2]
        miss_distance(0)

#if   threats > 1
    Threat(1)
#endif
#if   threats > 2
    Threat(2)
#endif
#if   threats > 3
    Threat(3)
#endif
#if   threats > 4
    Threat(4)
#endif
#if   threats > 5
    Threat(5)
#endif
#if   threats > 6
    Threat(6)
#endif
#if   threats > 7
    Threat(7)
#endif
#if   threats > 8
    Threat(8)
#endif
#if   threats > 9
    Threat(9)
#endif
#if   threats > 10
    Threat(10)
#endif
#if   threats > 11
    Threat(11)
#endif
#if   threats > 12
    Threat(12)
#endif
#if   threats > 13
    Threat(13)
#endif
#if   threats > 14
    Threat(14)
#endif
#if   threats > 15
    Threat(15)
#endif
#if   threats > 16
    Threat(16)
#endif
#if   threats > 17
    Threat(17)
#endif
#if   threats > 18
    Threat(18)
#endif
#if    threats > 19
    Threat(19)
```

```
#endif
#if    threats > 20
       Threat(20)
#endif
#if    threats > 21
       Threat(21)
#endif
#if    threats > 22
       Threat(22)
#endif
#if    threats > 23
       Threat(23)
#endif
#if    threats > 24
       Threat(24)
#endif
#if    threats > 25
       Threat(25)
#endif
#if    threats > 26
       Threat(26)
#endif
#if    threats > 27
       Threat(27)
#endif
#if    threats > 28
       Threat(28)
#endif
       x_dot
       y_dot
       phi_dot
       theta_dot
```

9.2 C Code

9.2.1 Main Host Program *closedloop.c*

```
#include "track.h"

/*
       Take earth's center as x,y,z=(0,0,0) for earth relative cordinates
       fpa coordinates is a cartesian coordinate system with
       the orgin on the interceptor. The z axis of this coordinate
       system extends through the center of the fpa out into
       the field of view, normal to the fpa.
       The x axis is runs through the extended plane of the fpa horizontally,
       and the y axis vertically. In order for this to be a right handed
       cordinate system, x is positive toward the right, but y is positive
       downward.
*/

extern void Initialize_track_object();
extern void track_object();
extern float host_io();
extern struct hostioType hostio;
extern  struct  ControlType Control;

FILE *fp;  /* input file */
float time_step;        /* Time increment per simulation step */

struct CentroidDataType CentroidData[100];
struct initializeType initialize;
struct ControlType Control;
int CentroidDataPtr,hostioPtr;

int run_length, cnt;

main(argc,argv)
int argc;
char **argv;
{
       float t, prev_miss_distance,miss;
       char line[100];
```

```
    if(argc>1) initialize_main_loop(argv[1]);
    else  initialize_main_loop("infile");
    initialize_threat_object();
    initialize_track_object();
    initialize_hostio();
    initialize_los_gnc();

    prev_miss_distance = 2e30;
    miss = 1e30;
    while (hostio.miss_distance<0) {

        CentroidDataPtr = 0;  /* Reset Centroid Data pointer */
        hostioPtr = 0;
        threat_object();        /* Generate a Threat Centroid */
        CentroidDataPtr = 0;  /* Reset Centroid Data pointer */
        hostioPtr = 0;
        track_object();             /* Track object & Select Target */
        los_gnc();

        t += time_step;             /* Move time forward */
        host_io();

    }
    gets(line);       /* Hold until CR pressed on Keyboard */
}

initialize_main_loop(infile)
char *infile;
{
    FILE *ifp;
    int a;

    if (NULL == (ifp=fopen(infile,"r"))) {
        fprintf(stderr,"Could not open input file %s\n",infile);
        exit(1);
    }

    a = fscanf(ifp," Threat Coordinates: %f %f %f",
            &(initialize.threat_coor[0]) ,
            &(initialize.threat_coor[1]) ,
            &(initialize.threat_coor[2]) );
    a += fscanf(ifp," Threat Velocity: %f %f %f",
            &(initialize.threat_velocity[0]) ,
            &(initialize.threat_velocity[1]) ,
            &(initialize.threat_velocity[2]) );
    a += fscanf(ifp," Threat Size, Intensity: %f %f",
            &(initialize.threat_size) ,
            &(initialize.threat_intensity) );
    a += fscanf(ifp," Missile Orientation: %f %f %f %f %f %f %f %f %f",
            &(initialize.missile_aim[0][0]),
            &(initialize.missile_aim[0][1]),
            &(initialize.missile_aim[0][2]),
            &(initialize.missile_aim[1][0]),
            &(initialize.missile_aim[1][1]),
            &(initialize.missile_aim[1][2]),
            &(initialize.missile_aim[2][0]),
            &(initialize.missile_aim[2][1]),
            &(initialize.missile_aim[2][2]) );
    a += fscanf(ifp," Missile Coordinates: %f %f %f",
            &(initialize.missile_coor[0]) ,
            &(initialize.missile_coor[1]) ,
            &(initialize.missile_coor[2]) );
    a += fscanf(ifp," Missile Velocity: %f %f %f",
            &(initialize.missile_velocity[0]),
            &(initialize.missile_velocity[1]),
            &(initialize.missile_velocity[2]) );
    a += fscanf(ifp," Miss Distance: %f",&(initialize.MissDistance));
    a += fscanf(ifp," Time Step: %f",&time_step);
    a += fscanf(ifp," Observation Orientation: %f %f %f %f %f %f %f %f %f",
            &(initialize.panview_orient[0][0]),
            &(initialize.panview_orient[0][1]),
            &(initialize.panview_orient[0][2]),
            &(initialize.panview_orient[1][0]),
```

```
                &(initialize.panview_orient[1][1]),
                &(initialize.panview_orient[1][2]),
                &(initialize.panview_orient[2][0]),
                &(initialize.panview_orient[2][1]),
                &(initialize.panview_orient[2][2]) );
        a += fscanf(ifp," Observation Point: %f %f %f",
                &(initialize.observation_point[0]) ,
                &(initialize.observation_point[1]) ,
                &(initialize.observation_point[2]) );
        a += fscanf(ifp," Panview FOV: %f %f",
                &(initialize.panview_view),
                &(initialize.panview_range) );
        a += fscanf(ifp," Angular, Linear Nav Constants: %f %f",
                &(initialize.ang_nav_const),
                &(initialize.lin_nav_const));
        a += fscanf(ifp," Thruster_interval: %f",
                &(initialize.thruster_interval));
        a += fscanf(ifp," Display Interval: %f",
                &(initialize.display_interval) );
        a += fscanf(ifp," Target Selection Time: %f",
                &(initialize.target_select_time) );
        a += fscanf(ifp," Maximum Divert: %f",
                &(initialize.max_divert) );
        a += fscanf(ifp," Number of Objects: %d",
                &(initialize.Num_obj) );
        a += fscanf(ifp," Final Acquisition Range: %f",
                &(initialize.FinalAcquisitionRange) );
        a += fscanf(ifp," Tracking Range: %f",
                &(initialize.TrackingRange) );


        initialize.time_step = time_step;

        if ( a != 48 ) {
                fprintf(stderr," Error Parsing Input \n");
                exit(1);
        }
}
```

9.2.2 Host FPA window Program *fpa.c*

```
#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <stdio.h>
#include "track.h"

static Notify_value   my_destroyer();
extern      Notify_error     notify_dispatch();

struct FpaDataType FpaData[32];

static Frame      frame;
static Canvas     canvas;
static Pixwin *pw;
char line[1000],command,name[CMS_NAMESIZE];
int x,x_pix,y_pix,color;
static unsigned char red[64] = {
        0,255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255

},
green[64]= {
        0,255, 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80,
        88, 96, 104, 112, 120, 128, 136, 144, 152, 160,
        168, 176, 184, 192, 200, 208, 216, 224, 232, 240,
        248, 255, 255, 255, 255, 255, 255, 255, 255, 255,
```

```
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255
};
blue[64]={
        0,255,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80,
        88, 96, 104, 112, 120, 128, 136, 144, 152, 160,
        168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248, 255
};

struct pr_pos plist[4];

fpa_mainloop()
{
        int     FpaPtr = 0,type,radius;
        register int i,j;
        notify_dispatch();

        pw_batch_on(pw);
        pw_writebackground(pw,1,1,255,255,PIX_SRC);
        while( FpaData[FpaPtr].intensity != 0 ) {

                x_pix = FpaData[FpaPtr].fd_pixel_x;
                y_pix = FpaData[FpaPtr].fd_pixel_y;
                radius = FpaData[FpaPtr].radius;
                color = FpaData[FpaPtr].intensity;

                x_pix += 1;
                y_pix += 1;
                x_pix *= 2;
                y_pix *= 2;
        if(radius>=1) {
          for(i=( -radius + x_pix);i<=(radius+x_pix);i++)
                  pw_vector(pw,i,radius+y_pix,i,-radius+y_pix,PIX_SRC,color);
        } else if( radius < 0 ) {
                pw_vector(pw,x_pix+4,y_pix,x_pix-4,y_pix,PIX_SRC,63);
                pw_vector(pw,x_pix,y_pix+4,x_pix,y_pix-4,PIX_SRC,63);
        } else
                pw_polypoint(pw,x_pix,y_pix,4,plist,PIX_SRC|PIX_COLOR(color));
        FpaPtr++;

        }
        pw_batch_off(pw);
        notify_dispatch();


}

/* An attempt at peaceful exits */
static Notify_value
my_destroyer(frame, status)
        Frame           frame;
        Destroy_status  status;
{
        if (status != DESTROY_CHECKING) {
                notify_stop();
        }
        return(notify_next_destroy_func(frame,status));
}


fpa_initialize()
{

        /* Initialize plist */

        plist[0].x = 0;
        plist[0].y = 0;
        plist[1].x = 1;
        plist[1].y = 0;
        plist[2].x = 0;
        plist[2].y = 1;
```

```
    plist[3].x = 1;
    plist[3].y = 1;

    /* Create frame and canvas */
    frame = window_create(NULL, FRAME,
        WIN_HEIGHT, 257+20+28,
        WIN_WIDTH, 257+20+12,
        WIN_X,      FPA_WIN_X,
        WIN_Y,      FPA_WIN_Y,
        FRAME_LABEL,    "FPA View",
        FRAME_NO_CONFIRM, TRUE,
        0);
    canvas = window_create(frame, CANVAS,
        CANVAS_RETAINED,    TRUE,
        CANVAS_HEIGHT, 258,
        CANVAS_WIDTH, 258,
        CANVAS_AUTO_SHRINK,    FALSE,
        CANVAS_AUTO_EXPAND,    FALSE,
        CANVAS_MARGIN, 10,
        0);

    /* get the canvas pixwin to draw into */
    pw = canvas_pixwin(canvas);

    /* Handle exits peacefully */
    notify_interpose_destroy_func(frame,my_destroyer);

    /* set up colormap */
    strcpy(name,"fpa");
    pw_setcmsname(pw,name);


    pw_putcolormap(pw,0,64,red,green,blue);

    pw_vector(pw,0,0,257,0,PIX_SRC,1);
    pw_vector(pw,257,0,257,257,PIX_SRC,1);
    pw_vector(pw,257,257,0,257,PIX_SRC,1);
    pw_vector(pw,0,257,0,0,PIX_SRC,1);
    window_set(frame,WIN_SHOW,TRUE,0);
}
```

9.2.3 Host Panoramic View *pan.c*

```
#include "track.h"
#include <suntool/sunview.h>
#include <suntool/canvas.h>

static Notify_value   my_destroyer();
extern    Notify_error    notify_dispatch();

struct PanDataType PanData[32];

static Frame    frame;
static Canvas   canvas;
static Pixwin *pw;
char line[1000],
        command,
        name[CMS_NAMESIZE];
int   x,
        x_pix,
        y_pix,color;
unsigned char red[4] = {
    0,255,0,255
},
green[4]= {
    0,255,255,0
},
blue[4]={
    0,255,0,0
};

pan_initialize()
{
    /* Create frame and canvas */
```

```
    int i;
    for(i=0;i<=32;i++) PanData[i].miss_distance = -1.0;

    frame = window_create(NULL, FRAME,
        WIN_WIDTH, 1024+28,
        WIN_HEIGHT, 512+12,
        WIN_X, PAN_WIN_X,
        WIN_Y, PAN_WIN_Y,
        FRAME_LABEL, "World View",
        FRAME_NO_CONFIRM, TRUE,
        0);
    canvas = window_create(frame, CANVAS,
        CANVAS_RETAINED, TRUE,
        CANVAS_HEIGHT, 512,
        CANVAS_WIDTH, 1024,
        CANVAS_AUTO_SHRINK, FALSE,
        CANVAS_AUTO_EXPAND, FALSE,
        0);

    /* get the canvas pixwin to draw into */
    pw = canvas_pixwin(canvas);

    /* Handle exits peacefully */
    notify_interpose_destroy_func(frame,my_destroyer);

    /* set up colormap */
    strcpy(name,"pan");
    pw_setcmsname(pw,name);
    pw_putcolormap(pw,0,4,red,green,blue);
    window_set(frame,WIN_SHOW,TRUE,0);
}

pan_mainloop()
{
    int panptr = 0;
    float miss_dist;
    char frm[30];

    notify_dispatch();

    pw_batch_on(pw);
    while( PanData[panptr].type != 0 ) {
        pw_put( pw,
                PanData[panptr].pd_pixel_x,
                PanData[panptr].pd_pixel_y,
                PanData[panptr].type);
        miss_dist = PanData[panptr].miss_distance;
        if(miss_dist>0) {
            sprintf(frm,"Miss Distance: %.2f ",miss_dist);
            pw_text(pw,20,46,PIX_SRC|PIX_COLOR(1),0,frm);
        }
        panptr++;
    }
    sprintf(frm,"Frame: %d ",PanData[0].frame);
    pw_text(pw,20,18,PIX_SRC|PIX_COLOR(1),0,frm);
    sprintf(frm,"Range (m): %8.2e ",PanData[0].pd_range);
    pw_text(pw,20,32,PIX_SRC|PIX_COLOR(1),0,frm);

    miss_dist = PanData[panptr].miss_distance;
    if(miss_dist>0) {
        sprintf(frm,"Miss Distance: %.2f ",miss_dist);
        pw_text(pw,20,46,PIX_SRC,0,frm);
    }
    pw_batch_off(pw);
    notify_dispatch();

}

/* An attempt at peaceful exits */
static Notify_value
my_destroyer(frame, status)
    Frame           frame;
    Destroy_status  status;
```

```
{
    if (status != DESTROY_CHECKING) {
        notify_stop();
    }
    return(notify_next_destroy_func(frame,status));
}
```

9.2.4 PFP to HOST interface *hostio.c*

```
#include "track.h"

/*  This process is the host buffer and conversion routine which
    prepares things for display on the 'hosts' display windows */

static FILE *fpa,*pan;
struct hostioType hostio[32];
static float observation[3],fov,pv_orient[3][3];
extern struct initializeType initialize;

extern struct FpaDataType FpaData[32];
extern struct PanDataType PanData[32];

float prev_distance,
        hio_xm[3],
        hio_xo[3],
        hio_xmr[3],
        hio_xor[3];

int   radius,
        pixel_x,
        pixel_y,
        total_int,
        frame_display_int,
        frame_cnt,
        i,
        j,
        pv_KEW_x,
        pv_KEW_y,
        pv_threat_x,
        pv_threat_y;
static float  Num_obj;

float host_io()
{
    int i,j;
    int fpaDptr=0,PanDptr=0;
    float distance;

    for (j=0;j<Num_obj;j++) {
        pixel_x = hostio[j].pixel_x;
        pixel_y = hostio[j].pixel_y;
        radius  = hostio[j].radius;
        total_int = hostio[j].total_intensity;

        /* Output the pertinent information to the fpa view */
        if ( pixel_x != -1  ) {
            FpaData[fpaDptr].fd_pixel_x = pixel_x;
            FpaData[fpaDptr].fd_pixel_y = 128-pixel_y;
            FpaData[fpaDptr].radius = radius;
            FpaData[fpaDptr].intensity = total_int;
            fpaDptr++;
        }
        FpaData[fpaDptr].intensity = 0;

        for(i=0;i<=2;i++) {
            hio_xo[i] = hostio[j].threat_coord[i];
            hio_xo[i] = hio_xo[i] - observation[i];
        }

        hio_solve(pv_orient,hio_xor,hio_xo);
        pv_threat_x = (int)( 511.5 + 1024 *atan((double)(hio_xor[0]/(-hio_xor[2])))
```

```
                        / fov );
                pv_threat_y = (int)( 255.5 + 1024 *atan((double)(hio_xor[1]/(-hio_xor[2])))
                        / fov );

                /* Output the pertinent information to the pan view for the threat */
                if( (pv_threat_x >= 0 ) && (pv_threat_x < 1024 ) &&
                        (pv_threat_y >=0  ) && (pv_threat_y < 512 ) &&
                        ( hio_xor[2]<=0)  ) {
                    PanData[PanDptr].pd_pixel_x = pv_threat_x;
                    PanData[PanDptr].pd_pixel_y = 512 - pv_threat_y;
                    PanData[PanDptr].type = 2;
                    PanData[PanDptr].pd_range = hostio[j].range;
                    PanData[PanDptr].miss_distance = hostio[j].miss_distance;
                    PanDptr++;
                }
            }
            for(i=0;i<=2;i++) {
                hio_xm[i] = hostio[0].missile_coord[i];
                hio_xm[i] = hio_xm[i] - observation[i];
            }

            hio_solve(pv_orient,hio_xmr,hio_xm);

            pv_KEW_x = (int)( 511.5 + 1024 *atan((double)(hio_xmr[0]/(-hio_xmr[2])))
                        / fov );
            pv_KEW_y = (int)( 255.5 + 1024 *atan((double)(hio_xmr[1]/(-hio_xmr[2])))
                        / fov );

            /* Output the pertinent information to the pan view for the KEW */
            if( (pv_KEW_x >= 0 ) && (pv_KEW_x < 1024 ) &&
                    (pv_KEW_y >=0  ) && (pv_KEW_y < 512 ) &&
                    ( hio_xmr[2]<=0)  ) {
                PanData[PanDptr].pd_pixel_x = pv_KEW_x;
                PanData[PanDptr].pd_pixel_y = 512 - pv_KEW_y;
                PanData[PanDptr].type = 1;
                PanDptr++;
            }

        PanData[0].frame++;

        PanData[PanDptr].type = 0;
        pan_mainloop();

        fpa_mainloop();

}

initialize_hostio()
{
        int i,j;

        fpa_initialize();
        pan_initialize();

        fov = tan( (double) ( initialize.panview_view / initialize.panview_range) );

        for(i=0;i<=2;i++) {
                observation[i] = initialize.observation_point[i];
                for(j=0;j<=2;j++)
                        pv_orient[i][j] = initialize.panview_orient[i][j];
        }
        Num_obj = initialize.Num_obj;

}


/*  User Cramers rule to solve , assumes det of matrix = 1*/
hio_solve(A,x,b)
float A[3][3],b[3],x[3];
```

```
{

  x[0] = (
        b[0] * ( A[1][1]*A[2][2] – A[2][1]*A[1][2] )
      – A[0][1] * ( b[1]  *A[2][2] – A[1][2]*b[2]  )
      + A[0][2] * ( b[1]  *A[2][1] – A[1][1]*b[2]  )
       );

  x[1] = (
       A[0][0] * ( b[1]  *A[2][2] – b[2]  *A[1][2] )
      – b[0]   * ( A[1][0]*A[2][2] – A[1][2]*A[2][0] )
      + A[0][2] * ( A[1][0]*  b[2] – b[1]  *A[2][0] )
       );


  x[2] = (
       A[0][0] * ( A[1][1]*b[2]  – A[2][1]*b[1] )
      – A[0][1] * ( A[1][0]*b[2]  – b[1]  *A[2][0] )
      + b[0]   * ( A[1][0]*A[2][1] – A[1][1]*A[2][0] )
       );

}
```

9.2.5 Target Generation Program *threat.c*

```
#include "track.h"
#include <math.h>
#include <stdio.h>

/*
      Take earth's center as x,y,z=(0,0,0) for earth relative cordinates

      fpa coordinates is a cartesian coordinate system with
      the orgin on the interceptor. The z axis of this coordinate
      system extends through the center of the fpa out into
      the field of view, normal to the fpa.
      The x axis is runs through the extended plane of the fpa horizontally,
      and the y axis vertically. In order for this to be a right handed
      cordinate system, x is positive toward the right, but y is positive
      downward.
*/

extern void track_object();
extern struct CentroidDataType CentroidData[100];
extern struct ControlType Control;
extern int CentroidDataPtr,hostioPtr;
extern struct initializeType initialize;
extern struct hostioType hostio[32];

static float aim[3][3], /* orientation of interceptor in radians */
                        xm[3],          /* X of KEW missile World coord */
                        vm[3],          /* Vel of KEW missile World coord */
                        mr[3],          /* location of this object in missile ref frame*/
                        distance,
                        previous_distance,
                        intense,    /* object brightness */
                        xo[3],      /* X of target */
                        vo[3],          /* velocity of target */
                        size,       /* diameter of the target */
                        time_step, /* Time step to take */
                        velocity,
                        t;              /* time */

static FILE *infp,*outfp;

static int   pixel_x,
             pixel_y,
```

```
                radius,
                total_intensity;
float compute_miss_distance();

threat_object()
{
    static float thetax,thetay;    /* interceptor rotation in radians  */

    static int   i,j;

    thetax = Control.delta_theta;
    thetay = Control.delta_phi;

    fprintf(infp,"thetax = %f\n",thetax);
    fprintf(infp,"thetay = %f\n",thetay);
    fflush(infp);

    t += time_step;

    ballistic(xo,vo,t);  /* computer ballistic path of this object */
    compute_KEW_position();
    rotate(aim,thetax,thetay);    /* Update Missile orientation */
    rel_vector(xo,xm,aim,mr);


    fpa_map();

    hostio[hostioPtr].pixel_x = pixel_x;
    hostio[hostioPtr].pixel_y = pixel_y;
    hostio[hostioPtr].radius = radius;
    for(i=0;i<=2;i++) {
        hostio[hostioPtr].missile_coord[i] = xm[i];
        hostio[hostioPtr].threat_coord[i] = xo[i];

    }
    hostio[hostioPtr].total_intensity = total_intensity;

    distance = sqrt( (double) (mr[0]*mr[0] + mr[1]*mr[1] + mr[2]*mr[2]));

    CentroidData[CentroidDataPtr].iAcoorX
        = CentroidData[0].iIcoorX = pixel_x;
    CentroidData[CentroidDataPtr].iAcoorY =
        CentroidData[0].iIcoorY = pixel_y;
    CentroidData[CentroidDataPtr].cArea = M_PI *radius*radius;
    if (CentroidData[CentroidDataPtr].cArea < 1) CentroidData[CentroidDataPtr].cArea = 1.0;
    CentroidData[CentroidDataPtr].cIntensity =
        total_intensity * (( 1 > CentroidData[0].cArea ) ? 1 : CentroidData[0].cArea);
    CentroidData[CentroidDataPtr].range = distance;

    CentroidData[++CentroidDataPtr].cArea=0;

    if(distance > previous_distance)
        hostio[hostioPtr].miss_distance = compute_miss_distance();
    previous_distance = distance;
    hostio[hostioPtr++].range = distance;

    fprintf(outfp,"pixel_x = %d\n",pixel_x);
    fprintf(outfp,"pixel_y = %d\n",pixel_y);
    fprintf(outfp,"radius = %d\n",radius);

    fprintf(outfp,"xm[0] = %f \n",xm[0]);
    fprintf(outfp,"xm[1] = %f \n",xm[1]);
    fprintf(outfp,"xm[2] = %f \n",xm[2]);

    fprintf(outfp,"xo[0] = %f \n",xo[0]);
    fprintf(outfp,"xo[1] = %f \n",xo[1]);
    fprintf(outfp,"xo[2] = %f \n",xo[2]);
    fprintf(outfp,"range = %f \n",distance);
    fprintf(outfp,"intensity = %d \n",CentroidData[0].cIntensity);
    fflush(outfp);

}
```

```
compute_KEW_position()
{

      float r,r3,gx,gy,gz,ar[2],am[3],z;

      z = xm[2] + Z_OFFSET;
      r = sqrt( (double) ( xm[0]*xm[0] + xm[1]*xm[1] + z*z) );
      r3 = (GRAV * EarthMass) / ( r * r * r );

      gx = -xm[0] * r3;
      gy = -xm[1] * r3;
      gz = -z * r3;

      fprintf(infp,"delta_x = %f\n",Control.delta_x);
      fprintf(infp,"delta_y = %f\n",Control.delta_y);
      fflush(infp);


      ar[0] = Control.delta_x * time_step;
      ar[1] = Control.delta_y * time_step;

      am[0] = aim[0][0] * ar[0] + aim[0][1] * ar[1];
      am[1] = aim[1][0] * ar[0] + aim[1][1] * ar[1];
      am[2] = aim[2][0] * ar[0] + aim[2][1] * ar[1];


      vm[0] += (am[0]+ gx) * time_step;
      vm[1] += (am[1]+ gy) * time_step;
      vm[2] += (am[2]+ gz) * time_step;

      xm[0] += vm[0] * time_step;
      xm[1] += vm[1] * time_step;
      xm[2] += vm[2] * time_step;
}



/* r = axb */
cross_prod(r,a,b)
float r[3],a[3],b[3];
{
      r[0] = a[1]*b[2] - a[2]*b[1];
      r[1] = a[2]*b[0] - a[0]*b[2];
      r[2] = a[0]*b[1] - a[1]*b[0];
}

/* Given thetax, thetay, calculate new position Orientation matrix A */
rotate(A,thetax,thetay)
float A[3][3],thetax,thetay;
{
      float p[3],q[3],r[3],pmag,qmag,rmag;
      int i;

      /* Calculate rotation about x axis   */
      for (i=0;i<3;i++) {
            p[i] = A[i][0] + A[i][2]*thetay;
            q[i] = A[i][1] + A[i][2]*thetax;
      }
      /* force orthogonal coordinate system */
      cross_prod(r,p,q);            /* r=pxq */
      cross_prod(q,r,p);            /* q=rxp */
      /* Force vectors to be unit length ! */
      pmag = sqrt( (double) ( p[0]*p[0] + p[1]*p[1] + p[2]*p[2]) );
      qmag = sqrt( (double) ( q[0]*q[0] + q[1]*q[1] + q[2]*q[2]) );
      rmag = sqrt( (double) ( r[0]*r[0] + r[1]*r[1] + r[2]*r[2]) );

      for(i=0;i<3;i++) {
                        A[i][0] = p[i]/pmag;   /*      p0 q0 r0   */
                        A[i][1] = q[i]/qmag;   /*  A = p1 q1 r1   */
                        A[i][2] = r[i]/rmag;   /*      p2 q2 r2   */
      }
```

```
}


/*  User Cramers rule to solve , assumes det of matrix = 1*/
solve(A,x,b)
float A[3][3],b[3],x[3];
{

      x[0] = (
              b[0] * ( A[1][1]*A[2][2] – A[2][1]*A[1][2] )
            – A[0][1] * ( b[1]   *A[2][2] – A[1][2]*b[2]   )
            + A[0][2] * ( b[1]   *A[2][1] – A[1][1]*b[2]   )
                 );

      x[1] = (
           A[0][0] * ( b[1]   *A[2][2] – b[2]   *A[1][2] )
            – b[0]   * ( A[1][0]*A[2][2] – A[1][2]*A[2][0] )
            + A[0][2] * ( A[1][0]*   b[2] – b[1]   *A[2][0] )
                 );


      x[2] = (
           A[0][0] * ( A[1][1]*b[2]   – A[2][1]*b[1] )
            – A[0][1] * ( A[1][0]*b[2]   – b[1]   *A[2][0] )
            + b[0]   * ( A[1][0]*A[2][1] – A[1][1]*A[2][0] )
                 );

}

/* compute current position and velocity */
ballistic(x,v,tnow)
float *x,*v,tnow;
{

      float r,r3,gx,gy,gz,z;
      z = x[2] + Z_OFFSET;
      r = sqrt((double) ( x[0]*x[0] + x[1]*x[1] + z*z ));
      r3 = (GRAV * EarthMass) / ( r * r * r );

      gx = –x[0] * r3;
      gy = –x[1] * r3;
      gz = –z * r3;

      v[0] += gx * time_step;
      v[1] += gy * time_step;
      v[2] += gz * time_step;

      x[0] += v[0] * time_step;
      x[1] += v[1] * time_step;
      x[2] += v[2] * time_step;
}


/* Catch Math Errors , this is done to prevent atan2 from
complaining about 0 */
int matherr(exc)
struct exception *exc;
{
   return(1);
}

/*
initialize these things
initialize_threat_object(aim, intense, xo, vo, size, time_step, t );
*/

initialize_threat_object()
{

      int i,j;
```

```
    if(NULL==(infp=fopen("threat_obj.input.dump","w"))) {
        fprintf(stderr,"Could not open threat_obj.input.dump\n");
        exit(1);
    }
    if(NULL==(outfp=fopen("threat_obj.output.dump","w"))) {
        fprintf(stderr,"Could not open threat_obj.output.dump\n");
        exit(1);
    }

    for(i=0;i<=2;i++) {
        for(j=0;j<=2;j++)      aim[i][j] = initialize.missile_aim[i][j];
        xo[i] = initialize.threat_coor[i];
        vo[i] = initialize.threat_velocity[i];
        xm[i] = initialize.missile_coor[i];
        vm[i] = initialize.missile_velocity[i];
    }
  size     = initialize.threat_size;
    intense   = initialize.threat_intensity;
  time_step = initialize.time_step;
    t = 0;

    for(i=0;i<=2;i++)
        for(j=0;j<=2;j++)
            fprintf(infp,"aim[%d][%d] = %f\n",i,j,aim[i][j]);
    for(i=0;i<=2;i++) fprintf(infp,"xo[%d] %f\n",i,xo[i]);
    for(i=0;i<=2;i++) fprintf(infp,"vo[%d] %f\n",i,vo[i]);
    for(i=0;i<=2;i++) fprintf(infp,"xm[%d] %f\n",i,xm[i]);
    for(i=0;i<=2;i++) fprintf(infp,"vm[%d] %f\n",i,vm[i]);
    fprintf(infp,"size = %f\n",size);
    fprintf(infp,"intensity = %f\n",intense);
    fprintf(infp,"time_step = %f\n\n",time_step);

    hostio[0].miss_distance = -1.0;
    previous_distance = 2e30;
    distance = 1e30;

}



fpa_map()
{
        int i,j;

        /* Calculate which pixels are actually turned on
            see note at top for weird stuff in y on missile
            relative coordinate system */
        if(mr[2]<0) {

            pixel_x = (  (ArraySize/2) -0.5+ PXEL_PER_RAD * atan((double) (mr[0]/(-mr[2]) )) );
            pixel_y = ( (ArraySize/2) -0.5 + PXEL_PER_RAD * atan((double) (mr[1]/(-mr[2]) )) );
        }
        if ((mr[2]>0) ||(pixel_x>(ArraySize-1))
                || (pixel_x<0) || (pixel_y>(ArraySize-1)) || (pixel_y<0)) {
            pixel_x = -1;
            pixel_y = -1;
        }


        /*  Here is where we figure out how big it is, and how bright.
              Intensity goes as inverse r squared */
        radius = (int) (atan((double) (size/mr[2]) ) * PXEL_PER_RAD );
        if (radius <0) radius = - radius;
        total_intensity =  ( intense / (mr[2] * mr[2]));
        if (radius>1) total_intensity = total_intensity / (radius*radius);
        if (total_intensity < 2 ) total_intensity = 1;
        if (total_intensity > 63 ) total_intensity = 63;

}


/* finds the relative vector  from xb to xa, then transforms
  by the inverse of the rotation matrix rot.
```

```
  Result is put in res, note that all of these
  must be declared! */
rel_vector(xa,xb,rot,res)
float xa[3],xb[3],rot[3][3],res[3];
{
  float d[3];

  d[0] = (xa[0]-xb[0]);
  d[1] = (xa[1]-xb[1]);
  d[2] = (xa[2]-xb[2]);

  solve(rot,res,d);
}


transpose(A,B)
float A[3][3],B[3][3];
{

    int i,j;
    for(i=0;i<=2;i++)
        for(j=0;j<=2;j++)
            B[j][i] = A[i][j];
}

float compute_miss_distance()
{
  float t,dt = .00001,dx[3],dist=1e30,prev_dist=2e30;
    int i;

  while(prev_dist > dist ) {
    prev_dist = dist;
    for(i=0;i<=2;i++) {
      xm[i] = xm[i] - vm[i] * dt;
      xo[i] = xo[i] - vo[i] * dt;
    }

    for(i=0;i<=2;i++) {
      dx[i] = xm[i] - xo[i];
      dx[i] = dx[i] * dx[i];
    }
    dist = sqrt( (double) ( dx[0] + dx[1] + dx[2] ) );
  }
  return(prev_dist);
}
```

9.2.6 Tracking Algorithm *track.c*

```
#include "track.h"

#define MaxTracks    32
#define MaxTracksP1   33
#define MaxMiss      4
#define NumCorWeight 0.1     /* used to determine target selection */
#define AreaWeight 0.4
#define IntensityWeight 0.5
#define    RAD_PER_PXEL 75.0e-6
#define TRUE -1
#define FALSE 0

extern    int   CentroidDataPtr;
extern    struct      CentroidDataType CentroidData[];
extern    structControlType Control;
extern    structinitializeType initialize;

/* program object_tracking;
{ - receives cluster data from clustering chip.
 - predicts the movement of each cluster in tfEstimateVx and tfEstimateVy
 - receive updated orientation of the FPA in delta_theta and delta_phi }
*/

struct TCtype {
          int          tcTrackNum,
```

```
                tcPriority;
        float tcRPriority,
                tcPositionX,
                tcPositionY,
                tcWeightX,
                tcWeightY,
                tcMinDistance,
                tcNumCor,
                tcCenID,
                tcCenAcoorX,
                tcCenAcoorY,
                tcCenIcoorX,
                tcCenIcoorY,
                tcRange;
        int     tcCenArea;
        long tcCenIntensity;
};

struct TFtype {
    int     tfTrackID,
            tfNumCor,
            tfNumMiss;
    float tfEstimateX,
            tfEstimateVX,
            tfXP11,
            tfXP12,
            tfXP22,
            tfEstimateY ,
            tfEstimateVY,
            tfYP11,
            tfYP12,
            tfYP22,
            tfArea,
            tfCenAcoorX,
            tfCenAcoorY,
            tfCenIcoorX,
            tfCenIcoorY,
            tfRange;
    long    tfIntensity;
    int     tfCSO;
};

struct Centroidtype {
    int   cID;
    float cAcoorX,
            cAcoorY,
            cIcoorX,
            cIcoorY,
            cRange;
    int   cArea;
    long  cIntensity;
};

static float time;
static struct    TCtype          TC[MaxTracksP1];
static struct TFtype          TF[MaxTracks];
static struct Centroidtype    centroid;

static int       LastTrack, LastNew,
                    TrackID, CentroidID,
                    ptr, lowptr;
static float dx, dy, dist2, priority, lowpriority, dt, dt2, dtsqr;
static float x_dot,y_dot;
static int       NotMatchCentroid, LastCentroid, iterations;
static float RadPix2, PixelsPerRadian, ProcessNoise, MeasurementNoise;
static float initialP11, initialP12, initialP22, PixelOffset;
static float TargetIntensity ;
static int   TargetTrack , frame , error, TargetSelected ;
static float     delta_theta, delta_phi, theta_dot, phi_dot, threat_intensity;
static float  range;
static floatMaxDist;
static floatprevious_x,
        previous_y;
```

```
static float Num_obj;
static int  FinalAcquisition; /* On Ada this is supposed to be boolean type */
static float TargetSelectionTime;
static float TrackingRange; /* do not apply angular correction if inside this range. ( in radians ) */
static float FinalAcquisitionRange;

#ifdef DUMP
FILE *infp,*outfp;
#endif

track_object()
{
    float xdot_range_comp,
          ydot_range_comp,
          a;
        delta_theta = Control.delta_theta;
        delta_phi  = Control.delta_phi;

#ifdef DUMP
        fprintf(infp,"delta_theta: %f\n",delta_theta);
        fprintf(infp,"delta_phi: %f\n",delta_phi);
#endif

        range = TF[TargetTrack].tfRange;
        if (range > 80e3) MaxDist = 6;
        else MaxDist = (80e3/range)*6;

        PerformTracking();

        frame = frame + 1;
        time = time + dt;
        if (time>= TargetSelectionTime) {
            if (!TargetSelected) {
                SelectTarget();
                phi_dot  = TF[TargetTrack].tfEstimateX - 0.5*RAD_PER_PXEL;
                theta_dot = TF[TargetTrack].tfEstimateY - 0.5*RAD_PER_PXEL;
                x_dot = range*tan((double) TF[TargetTrack].tfEstimateVX)/dt;
                y_dot = range*tan((double) TF[TargetTrack].tfEstimateVY)/dt;
            }
            else {
                if ((range < FinalAcquisitionRange) && ((TF[TargetTrack].tfNumMiss > 0) || FinalAcquisition)) {
                    FinalAcquisition = TRUE;
                    FindTarget();
                    range    = TF[TargetTrack].tfRange;
                    x_dot = range*tan((double) TF[TargetTrack].tfEstimateX-0.5*RAD_PER_PXEL)/dt;
                    y_dot = range*tan((double) TF[TargetTrack].tfEstimateY-0.5*RAD_PER_PXEL)/dt;
                    phi_dot  = TF[TargetTrack].tfEstimateX - 0.5*RAD_PER_PXEL;
                    theta_dot = TF[TargetTrack].tfEstimateY - 0.5*RAD_PER_PXEL;
                }
                else
                {
                    if (TF[TargetTrack].tfNumMiss > 0) {
                        FindTarget();
                        range    = TF[TargetTrack].tfRange;
                    }
                    if ((TF[TargetTrack].tfEstimateX>TrackingRange) || (TF[TargetTrack].tfEstimateX<-TrackingRange)) {
                        phi_dot  = TF[TargetTrack].tfEstimateX - 0.5*RAD_PER_PXEL;
                    }
                    else {
                        phi_dot = 0;
                    }
                    if ((TF[TargetTrack].tfEstimateY>TrackingRange) || (TF[TargetTrack].tfEstimateY<-TrackingRange)) {
                        theta_dot = TF[TargetTrack].tfEstimateY - 0.5*RAD_PER_PXEL;
                    }
                    else {
                        theta_dot = 0;
                    }
                    x_dot = range*tan((double) TF[TargetTrack].tfEstimateVX)/dt;
                    y_dot = range*tan((double) TF[TargetTrack].tfEstimateVY)/dt;
                }
                previous_x = TF[TargetTrack].tfEstimateX;
                previous_y = TF[TargetTrack].tfEstimateY;
            }
```

```
        }
        Control.x_dot = x_dot;
        Control.y_dot = y_dot;
        Control.phi_dot = phi_dot;
        Control.theta_dot = theta_dot;

#ifdef DUMP
        fprintf(outfp,"X_dot: %f\n",x_dot);
        fprintf(outfp,"Y_dot: %f\n",y_dot);
        fprintf(outfp,"Phi_dot: %f\n",phi_dot);
        fprintf(outfp,"Theta_dot: %f\n",theta_dot);
#endif

}

/*=============================================================*/

KalmanTimeUpdate(position,velocity,P11,P12,P22)
float *position,*P11,*P12,*P22,*velocity;
{
 *position =  *position + (dt * *velocity);
 *P11          = *P11 + ((dt2 * *P12) + (dtsqr * *P22));
 *P12          = *P12 + (dt * *P22);
 *P22          = *P22 + ProcessNoise;
}

/*=============================================================
procedure KalmanMeasurementUpdate ( MeasuredPosition : real;
        var position, velocity, P11, P12, P22 : real ); */

KalmanMeasurementUpdate(MeasuredPosition,position,velocity,P11,P12,P22)
float *MeasuredPosition,*position,*velocity,*P11,*P12,*P22;
{
    float temp, K1, K2, StateError;

    StateError = *MeasuredPosition – *position;
    temp = 1.0 / (*P11 + MeasurementNoise);
    K1 = temp * *P11;
    K2 = temp * *P12;
    temp = *P12;
    *P11 = *P11 * (1.0 – K1) ;
    *P12 = *P12 * (1.0 – K1) ;
    *P22 = *P22  – (K2 * temp);
    *position = *position + (K1 * StateError);
    *velocity = *velocity + (K2 * StateError);
}

/*=============================================================
procedure Initialize; */

initialize_track_object()
{
    int i;

#ifdef DUMP
    if (NULL ==(infp=(fopen("Track_obj.input.dump","w")))) {
        fprintf(stderr,"Couldn't open Track_obj.input.dump\n");
        exit(1);
    }
    if (NULL ==(outfp=(fopen("Track_obj.output.dump","w")))) {
        fprintf(stderr,"couldn't open Track_obj.output.dump\n");
        exit(1);
    }
#endif

    /* Get Time stamp from master initialization */
    dt = initialize.time_step;
    dt2 = 2.0 * dt;
    dtsqr = dt * dt;

    FinalAcquisition = FALSE;
```

```
    RadPix2       = RAD_PER_PXEL * RAD_PER_PXEL;
    PixelsPerRadian = 1.0 / RAD_PER_PXEL;
    ProcessNoise   = 2.5 * RadPix2;
    MeasurementNoise = 1.0 * RadPix2;

    initialP11    = 2.0 * RadPix2;
    initialP12    = 0.0 * RadPix2;
    initialP22    = 100.0 * RadPix2;

    PixelOffset = ArraySize / 2.0 - 0.5;

    LastTrack = 0;
    LastNew = 0;
    for (i=0;i<=MaxTracks;i++) {
        TC[i].tcTrackNum = i;
        TC[i].tcPriority = 0;
    }
    TrackID = 0;
    TargetSelected = FALSE;
    time = 0;
    x_dot = 0;
    y_dot = 0;
    theta_dot=0;
    phi_dot=0;

    frame = 0;
    error = FALSE;
    threat_intensity = initialize.threat_intensity;
    TargetSelectionTime = initialize.target_select_time;
    Num_obj  = initialize.Num_obj;
    TrackingRange = initialize.TrackingRange*RAD_PER_PXEL;
    FinalAcquisitionRange = initialize.FinalAcquisitionRange;

#ifdef DUMP
    fprintf(infp,"threat_intensity: %f \n",threat_intensity);
    fprintf(infp,"time_step: %f \n",dt);
    fprintf(infp,"TargetSelectionTime: %f \n",TargetSelectionTime);
    fprintf(infp,"Num_obj: %d \n",Num_obj);
    fprintf(infp,"FinalAcquistionRange: %f\n",FinalAcquisitionRange);
    fprintf(infp,"TrackingRange: %d \n\n", initialize.TrackingRange);
#endif
}

/*=============================================================
Procedure Compensate_FPA_Rotation; */

Compensate_FPA_Rotation()
{
    int i;
    for(i=0;i<=(LastTrack - 1);i++) {
   TF[TC[i].tcTrackNum].tfEstimateX -= delta_phi;
   TF[TC[i].tcTrackNum].tfEstimateY -= delta_theta;
  }
}

/*=============================================================
procedure UpdateTracks; */

UpdateTracks()
{
    int i;

    for(i=0;i<=(LastTrack - 1);i++) {
      /* check for multiple centroid associations */
      if ((TC[i].tcNumCor > 0) && (TC[i].tcCenID >= 0) ) {
                ptr = i + 1;
                while (ptr < LastTrack) {
                    if ((TC[i].tcCenID  == TC[ ptr ].tcCenID) )
                        if ((TC[i].tcMinDistance > TC[ ptr ].tcMinDistance) )
                            TC[i].tcCenID = -1;
                        else
                            TC[ ptr ].tcCenID = -1;
                    ptr = ptr + 1;
```

```
                }
        }
        if ((TC[i].tcNumCor == 0) ) /* no correlation with track file */
                if ((TF[TC[i].tcTrackNum].tfNumMiss >= MaxMiss) )
                        TC[i].tcPriority = 0;
                else {
                        TF[TC[i].tcTrackNum].tfNumMiss++;
                        TC[i].tcPriority = TF[TC[i].tcTrackNum].tfNumCor
                                        - TF[TC[i].tcTrackNum].tfNumMiss;
                }
        else if ( (TC[i].tcCenID == (-1)) )
                /* correlation but another track file is closer */
                {
                        TF[TC[i].tcTrackNum].tfNumMiss = 0;
                        TC[i].tcPriority = TF[TC[i].tcTrackNum].tfNumCor;
                        KalmanMeasurementUpdate( &(TC[i].tcPositionX),
                                                &(TF[TC[i].tcTrackNum].tfEstimateX),
                                                &(TF[TC[i].tcTrackNum].tfEstimateVX),
                                                &(TF[TC[i].tcTrackNum].tfXP11),
                                                &(TF[TC[i].tcTrackNum].tfXP12),
                                                &(TF[TC[i].tcTrackNum].tfXP22) );
                        KalmanMeasurementUpdate( &(TC[i].tcPositionY),
                                                &(TF[TC[i].tcTrackNum].tfEstimateY),
                                                &(TF[TC[i].tcTrackNum].tfEstimateVY),
                                                &(TF[TC[i].tcTrackNum].tfYP11),
                                                &(TF[TC[i].tcTrackNum].tfYP12),
                                                &(TF[TC[i].tcTrackNum].tfYP22) );
                } else {    /* update from correlated centroid */
                        TF[TC[i].tcTrackNum].tfNumCor++;
                        TF[TC[i].tcTrackNum].tfNumMiss = 0;
                        TC[i].tcPriority = TF[TC[i].tcTrackNum].tfNumCor;
                        TF[TC[i].tcTrackNum].tfIntensity = TC[i].tcCenIntensity;
                        TF[TC[i].tcTrackNum].tfArea = TC[i].tcCenArea;
                        TF[TC[i].tcTrackNum].tfRange = TC[i].tcRange;
                        TF[TC[i].tcTrackNum].tfCenAcoorX = TC[i].tcCenAcoorX;
                        TF[TC[i].tcTrackNum].tfCenAcoorY = TC[i].tcCenAcoorY;
                        TF[TC[i].tcTrackNum].tfCenIcoorX = TC[i].tcCenIcoorX;
                        TF[TC[i].tcTrackNum].tfCenIcoorY = TC[i].tcCenIcoorY;
                        KalmanMeasurementUpdate( &(TC[i].tcCenAcoorX),
                                                &(TF[TC[i].tcTrackNum].tfEstimateX),
                                                &(TF[TC[i].tcTrackNum].tfEstimateVX),
                                                &(TF[TC[i].tcTrackNum].tfXP11),
                                                &(TF[TC[i].tcTrackNum].tfXP12),
                                                &(TF[TC[i].tcTrackNum].tfXP22) );
                        KalmanMeasurementUpdate( &(TC[i].tcCenAcoorY),
                                                &(TF[TC[i].tcTrackNum].tfEstimateY),
                                                &(TF[TC[i].tcTrackNum].tfEstimateVY),
                                                &(TF[TC[i].tcTrackNum].tfYP11),
                                                &(TF[TC[i].tcTrackNum].tfYP12),
                                                &(TF[TC[i].tcTrackNum].tfYP22) );
                }
        }
}

/*=============================================================
procedure InitializeNewTracks; */

InitializeNewTracks()
{

        int i;
        for (i=LastTrack;i<=(LastNew - 1);i++) {
                TF[TC[i].tcTrackNum].tfTrackID = TrackID;
                TrackID++;
                TF[TC[i].tcTrackNum].tfNumCor = 1;
                TF[TC[i].tcTrackNum].tfNumMiss = 0;
                TC[i].tcPriority = 1;
                TF[TC[i].tcTrackNum].tfIntensity = TC[i].tcCenIntensity;
                TF[TC[i].tcTrackNum].tfArea = TC[i].tcCenArea;
                TF[TC[i].tcTrackNum].tfRange = TC[i].tcRange;
                TF[TC[i].tcTrackNum].tfEstimateX = TC[i].tcCenAcoorX;
                TF[TC[i].tcTrackNum].tfEstimateVX = 0.0;
                TF[TC[i].tcTrackNum].tfXP11 = initialP11;
                TF[TC[i].tcTrackNum].tfXP12 = initialP12;
```

```
            TF[TC[i].tcTrackNum].tfXP22 = initialP22;
            TF[TC[i].tcTrackNum].tfEstimateY = TC[i].tcCenAcoorY;
            TF[TC[i].tcTrackNum].tfEstimateVY = 0.0;
            TF[TC[i].tcTrackNum].tfYP11 = initialP11;
            TF[TC[i].tcTrackNum].tfYP12 = initialP12;
            TF[TC[i].tcTrackNum].tfYP22 = initialP22;
            TF[TC[i].tcTrackNum].tfCenAcoorX = TC[i].tcCenAcoorX;
            TF[TC[i].tcTrackNum].tfCenAcoorY = TC[i].tcCenAcoorY;
            TF[TC[i].tcTrackNum].tfCenIcoorX = TC[i].tcCenIcoorX;
            TF[TC[i].tcTrackNum].tfCenIcoorY = TC[i].tcCenIcoorY;
      }
}

/*=============================================================
procedure RankTracks; */

RankTracks()
{
      int i,j,temp1,temp2;

      if ((LastNew > 1))          /*    sort on priority */
            for (i=0;i<=LastNew-2 ;i++) {
                  ptr = i;
                  for (j=(i+1);j<=(LastNew-1);j++)
                        if (TC[i].tcPriority < TC[j].tcPriority)
                              ptr = j;
                  if (ptr != i) {
                        temp1 = TC[ptr].tcTrackNum;
                        temp2 = TC[ptr].tcPriority;
                        TC[ptr].tcTrackNum = TC[i].tcTrackNum;
                        TC[ptr].tcPriority = TC[i].tcPriority;
                        TC[i].tcTrackNum = temp1;
                        TC[i].tcPriority = temp2;
                  }
            }
      LastTrack = 0;
      while (TC[LastTrack].tcPriority > 0)  LastTrack = LastTrack + 1;
      LastNew = LastTrack;
}

/*=============================================================
procedure SetUpCorrelation; */

SetUpCorrelation()
{
      int i;
      for (i=0;i<=(LastTrack - 1);i++) {
            KalmanTimeUpdate( &TF[TC[i].tcTrackNum].tfEstimateX,
                              &TF[TC[i].tcTrackNum].tfEstimateVX,
                              &TF[TC[i].tcTrackNum].tfXP11,
                              &TF[TC[i].tcTrackNum].tfXP12,
                              &TF[TC[i].tcTrackNum].tfXP22 );
            KalmanTimeUpdate( &TF[TC[i].tcTrackNum].tfEstimateY,
                              &TF[TC[i].tcTrackNum].tfEstimateVY,
                              &TF[TC[i].tcTrackNum].tfYP11,
                              &TF[TC[i].tcTrackNum].tfYP12,
                              &TF[TC[i].tcTrackNum].tfYP22 );
            TC[i].tcPositionX      = TF[TC[i].tcTrackNum].tfEstimateX;
            TC[i].tcPositionY      = TF[TC[i].tcTrackNum].tfEstimateY;
            TC[i].tcWeightX              = 1.0 / (TF[TC[i].tcTrackNum].tfXP11);
            TC[i].tcWeightY              = 1.0 / (TF[TC[i].tcTrackNum].tfYP11);
            TC[i].tcMinDistance  = MaxDist;
            TC[i].tcNumCor       = 0;
      }
}

/*=============================================================
procedure InputCentroid; */

InputCentroid()
{
      float  iAcoorX, iAcoorY, iIcoorX, iIcoorY ;
```

```
        iAcoorX             = CentroidData[CentroidDataPtr].iAcoorX;
        iAcoorY             = CentroidData[CentroidDataPtr].iAcoorY;
        iIcoorX             = CentroidData[CentroidDataPtr].iIcoorX;
        iIcoorY             = CentroidData[CentroidDataPtr].iIcoorY;
        centroid.cArea   = CentroidData[CentroidDataPtr].cArea;
        centroid.cIntensity = CentroidData[CentroidDataPtr].cIntensity;
        centroid.cRange = CentroidData[CentroidDataPtr++].range;

#ifdef DUMP
        fprintf(infp,"iIcoorX: %f\n",iIcoorX);
        fprintf(infp,"iIcoorY: %f\n",iIcoorY);
        fprintf(infp,"cArea: %d\n",centroid.cArea);
        fprintf(infp,"cIntensity: %d\n",centroid.cIntensity);
        fprintf(infp,"cRange: %f\n\n",centroid.cRange);
#endif

        centroid.cID = CentroidID;
        CentroidID++;
        centroid.cAcoorX = (iAcoorX – PixelOffset) * RAD_PER_PXEL;
        centroid.cAcoorY = (iAcoorY – PixelOffset) * RAD_PER_PXEL;
        centroid.cIcoorX = (iIcoorX – PixelOffset) * RAD_PER_PXEL;
        centroid.cIcoorY = (iIcoorY – PixelOffset) * RAD_PER_PXEL;
}

/*=============================================================
procedure PerformTracking; */

PerformTracking()
{   int   centroid_cntr=0;
    Compensate_FPA_Rotation();
    SetUpCorrelation();
    LastCentroid = 0;
    CentroidID = 0;
    while (centroid_cntr<Num_obj) {
        centroid_cntr++;
        InputCentroid();
        /*   compare against tracks */
        NotMatchCentroid = TRUE;
        ptr = 0;
        while (ptr < LastTrack) {
            dx = centroid.cAcoorX – TC[ptr].tcPositionX;
            dy = centroid.cAcoorY – TC[ptr].tcPositionY;
            dist2 = TC[ptr].tcWeightX*(dx*dx) + TC[ptr].tcWeightY*(dy*dy);
            if (dist2 < MaxDist) {      /* correlated with this track file */
                TC[ptr].tcNumCor = TC[ptr].tcNumCor + 1;
                NotMatchCentroid = FALSE;
                if (dist2 < TC[ptr].tcMinDistance) {
                    TC[ptr].tcMinDistance     = dist2;
                    TC[ptr].tcCenID                 = centroid.cID;
                    TC[ptr].tcCenAcoorX       = centroid.cAcoorX;
                    TC[ptr].tcCenAcoorY       = centroid.cAcoorY;
                    TC[ptr].tcCenIcoorX       = centroid.cIcoorX;
                    TC[ptr].tcCenIcoorY       = centroid.cIcoorY;
                    TC[ptr].tcCenArea         = centroid.cArea;
                    TC[ptr].tcRange           = centroid.cRange;
                    TC[ptr].tcCenIntensity = centroid.cIntensity;
                }
            }
            ptr++;
        }
        /*   compare against uncorrelated centroids */
        if (NotMatchCentroid && (ptr < MaxTracks)) {
            priority = centroid.cAcoorX*centroid.cAcoorX
                        + centroid.cAcoorY*centroid.cAcoorY;
            if (LastNew < MaxTracks) {     /*   room to start new track */
                TC[LastNew].tcRPriority         = priority;
                TC[LastNew].tcCenID                 = centroid.cID;
                TC[LastNew].tcCenAcoorX           = centroid.cAcoorX;
                TC[LastNew].tcCenAcoorY           = centroid.cAcoorY;
                TC[LastNew].tcCenIcoorX       = centroid.cIcoorX;
                TC[LastNew].tcCenIcoorY       = centroid.cIcoorY;
                TC[LastNew].tcCenArea     = centroid.cArea;
                TC[LastNew].tcRange       = centroid.cRange;
                TC[LastNew].tcCenIntensity  = centroid.cIntensity;
```

```
                    LastNew++;
                } else {    /*    check other new track for priority */
                    lowpriority = -1.0;
                    lowptr = LastNew;
                    while (ptr < MaxTracks) {
                        if (lowpriority < TC[ptr].tcPriority) {
                            lowptr = ptr;
                            lowpriority = TC[ptr].tcPriority;
                        }
                        ptr = ptr + 1;
                    }
                    if (priority < lowpriority) {
                        TC[lowptr].tcRPriority      = priority;
                        TC[lowptr].tcCenID          = centroid.cID;
                        TC[lowptr].tcCenAcoorX      = centroid.cAcoorX;
                        TC[lowptr].tcCenAcoorY      = centroid.cAcoorY;
                        TC[lowptr].tcCenIcoorX      = centroid.cIcoorX;
                        TC[lowptr].tcCenIcoorY      = centroid.cIcoorY;
                        TC[lowptr].tcCenArea        = centroid.cArea;
                        TC[lowptr].tcRange   = centroid.cRange;
                        TC[lowptr].tcCenIntensity = centroid.cIntensity;
                    }
                }
            }
        }

        UpdateTracks();

        InitializeNewTracks();

        RankTracks();

}

/*=============================================================
procedure SelectTarget; */

SelectTarget()
{
        int i;
    float maxNumCor, maxArea, maxIntensity, Weight, TargetWeight;

        maxNumCor = 0;
        maxArea = 0;
        maxIntensity = 0;
        TargetIntensity = 0;
        /* select the brightest object among the objects that meets minimum
            number of Corelation */
        for (i=0 ;i<=(LastTrack - 1);i++) {
            if (TF[i].tfNumCor > maxNumCor)          maxNumCor = TF[i].tfNumCor;
            if (TF[i].tfArea > maxArea )             maxArea = TF[i].tfArea;
            if (TF[i].tfIntensity > maxIntensity )
                maxIntensity = TF[i].tfIntensity;
        }
        TargetWeight = 0;
        TargetTrack = -1;

        for (i=0; i<=(LastTrack - 1);i++)  {
            Weight = (NumCorWeight*TF[i].tfNumCor/maxNumCor) +
                (AreaWeight*TF[i].tfArea/maxArea) +
                (IntensityWeight*TF[i].tfIntensity/maxIntensity);
            if (Weight > TargetWeight) {
                TargetTrack = i;
                TargetWeight = Weight;
            }
        }
        if (LastTrack > 0 ) TargetSelected = TRUE;
}

FindTarget()
{       int i;
        TargetIntensity = 0;
        /* select the brightest object among the objects with 0 misses */
```

```
    for (i=0 ;i<=(LastTrack - 1);i++) {
        if ((TF[i].tfNumMiss == 0) && (TF[i].tfIntensity > TargetIntensity )) {
            TargetIntensity = TF[i].tfIntensity;
            TargetTrack = i;
        }
    }
}
```

9.2.7 Guidance *los_gnc.c*

```
#include "track.h"

extern struct ControlType Control;
extern struct initializeType initialize;

static float angNavConst, linNavConst, thruster_interval,
             time_step, thruster_time, max_divert;

/* compute zero mass line of site guidance law */
los_gnc()
{

    float div;
    if (thruster_time >= thruster_interval) {
        Control.delta_theta = angNavConst * Control.theta_dot;
        Control.delta_phi   = angNavConst * Control.phi_dot;

        div = linNavConst * Control.x_dot;
        if (div>max_divert) div = max_divert;   /* Limit divert */
        else if (div< -max_divert) div = -max_divert;
        Control.delta_x     = div;

        div = linNavConst * Control.y_dot;
        if (div>max_divert) div = max_divert;
        else if (div< -max_divert) div = -max_divert;
        Control.delta_y     = div;

        thruster_time = time_step;

    }
    else {

        thruster_time += time_step;
        Control.delta_x = 0;
        Control.delta_y = 0;
        Control.delta_theta = 0;
        Control.delta_phi = 0;
    }

}

initialize_los_gnc()
{
    angNavConst = initialize.ang_nav_const;
    linNavConst = initialize.lin_nav_const;
    thruster_interval = initialize.thruster_interval;
    time_step = initialize.time_step;
    thruster_time = thruster_interval;
    max_divert = initialize.max_divert;
}
```

9.3 Ada Code

9.3.1 HOST interface routine *hostio.ada*

```
#ifndef NET
#define NET net
#endif

procedure hostio is
```

```
        fval : float;
--- for inputing floats you cant input
        ival : integer;
--- for inputing integers you cant input

        fov : float;

        type float_3        is array (0 .. 2) of float;
        type float_3x3      is array (0 .. 2, 0 .. 2) of float;
        type float_threats  is array (0 .. 31) of float;
        type float_threatsx3 is array (0 .. 31, 0 .. 2) of float;
        type integer_threats is array (0 .. 31) of integer;

        nearest_range,
        nearest_miss_distance,
        panview_view,
        panview_range : float;

        observation_point,
        missile_coord,
        missile_coordr : float_3;

        pv_orient : float_3x3;

        _range,
        miss_distance,
        prev_distance : float_threats;

        threat_coordr,
        threat_coord : float_threatsx3;

        KEW_x,
        KEW_y,
        frame_number,
        frame_update,
        hidden_frame_cnt,
        Num_obj : integer;

        pixel_x,
        pixel_y,
        radius,
        total_int,
        threat_x,
        threat_y : integer_threats;

---     pix_x,pix_y : integer_threats;

procedure initialize_hostio is

begin

        ll_receive(host, panview_view);
        ll_receive(host, panview_range);

        fov := tan(panview_view / panview_range);

        ll_receive(host, fval);
        observation_point(0) := fval;
        ll_receive(host, fval);
        observation_point(1) := fval;
        ll_receive(host, fval);
        observation_point(2) := fval;

        ll_receive(host, fval);
        pv_orient(0, 0) := fval;
        ll_receive(host, fval);
        pv_orient(0, 1) := fval;
        ll_receive(host, fval);
        pv_orient(0, 2) := fval;
        ll_receive(host, fval);
        pv_orient(1, 0) := fval;
        ll_receive(host, fval);
```

```
        pv_orient(1, 1) := fval;
        ll_receive(host, fval);
        pv_orient(1, 2) := fval;
        ll_receive(host, fval);
        pv_orient(2, 0) := fval;
        ll_receive(host, fval);
        pv_orient(2, 1) := fval;
        ll_receive(host, fval);
        pv_orient(2, 2) := fval;

        Num_obj := 0;
        ll_receive(host, Num_obj, lsw);

        frame_update :=0;
        ll_receive(host, frame_update, lsw);

        frame_number := 0;

end initialize_hostio;


procedure dumptohost is
i : integer;
begin
            -- Output the pan view data for the KEW
            if ((((0 <= KEW_x AND KEW_x < 1024)
                        AND 0 <= KEW_y) AND KEW_y < 512)
                        AND missile_coordr(2) <= 0.0) then

                ll_send(host, KEW_x, lsw);
                ll_send(host, 512 - KEW_y, lsw);
                ll_send(host, 2, lsw);  -- Threat type
            else

                ll_send(host, -1, lsw);
                ll_send(host, -1, lsw);
                ll_send(host, -1, lsw);
            end if;

            ll_send(host,nearest_range);
            ll_send(host,nearest_miss_distance);

            -- Send out the the objects (targets)
            for i in 0 .. Num_obj-1 loop

                ival := pixel_x(i);
                ll_send(host, ival, lsw);
--              ll_send(host,pix_x(20-i),lsw);

                ival := pixel_y(i);
                ll_send(host, ival, lsw);
--              ll_send(host,pix_y(20-i),lsw);

                ival := radius(i);
                ll_send(host, ival, lsw);
                ll_send(host, total_int(i));


                -- Output the pan view data for the threat
                if ((((0 <= threat_x(i)  AND threat_x(i) < 1024)
                        AND 0 <= threat_y(i)) AND threat_y(i) < 512)
                        AND threat_coordr(i, 2) <= 0.0) then

                    ll_send(host, threat_x(i), lsw);
                    ll_send(host, 512 - threat_y(i), lsw);
                    ll_send(host, 3, lsw);  -- Threat type
                    ll_send(host, _range(i));
                    ll_send(host, miss_distance(i));
                else
                    ll_send(host, -1, lsw);
                    ll_send(host, -1, lsw);
                    ll_send(host, -1, lsw);
                    ll_send(host, -1.0);
                    ll_send(host, -1.0);
```

```
                end if;

            end loop; — for i in 0 .. Num_obj–1
            ll_send(host, frame_number, lsw);
end dumptohost;

procedure dohostio is

    j : integer;
    dummy : integer;

begin

    frame_number := frame_number + 1;

    j := 0;
    while j < Num_obj loop

        ival := 0;
        fval := 7.0;
        ll_receive(NET, ival,lsw);
        fval := 7.0;
        pixel_x(j) := ival;
—       pix_x(20–j) := ival;

        ival := 0;
        fval := 7.0;
        ll_receive(NET, ival,lsw);
        fval := 7.0;
        pixel_y(j) := ival;
—       pix_y(20–j) := ival;

        ival := 0;
        ll_receive(NET, ival, lsw);
        radius(j) := ival;

        ll_receive(NET, ival);
        total_int(j) := ival;

        ll_receive(NET, fval);
        _range(j) := fval;

        if (j = 0) then

            ll_receive(NET, fval);
            missile_coord(0) := fval;
            ll_receive(NET, fval);
            missile_coord(1) := fval;
            ll_receive(NET, fval);
            missile_coord(2) := fval;

            missile_coord(0)
                := missile_coord(0) – observation_point(0);
            missile_coord(1)
                := missile_coord(1) – observation_point(1);
            missile_coord(2)
                := missile_coord(2) – observation_point(2);

—   hio_solve(pv_orient, missile_coordr, missile_coord);

— User Cramers rule to solve, assumes det of matrix := 1

            missile_coordr(0) :=
            (missile_coord(0) * (pv_orient(1, 1) * pv_orient(2, 2)
                        – pv_orient(2, 1) * pv_orient(1, 2))
                – pv_orient(0, 1) * (missile_coord(1) * pv_orient(2, 2)
                        – pv_orient(1, 2) * missile_coord(2))
                + pv_orient(0, 2) * (missile_coord(1) * pv_orient(2, 1)
                        – pv_orient(1, 1) * missile_coord(2))
                );

            missile_coordr(1) :=
```

```
                    (pv_orient(0, 0) * (missile_coord(1) * pv_orient(2, 2)
                        - missile_coord(2) * pv_orient(1, 2))
                    - missile_coord(0)   * (pv_orient(1, 0) * pv_orient(2, 2)
                        - pv_orient(1, 2) * pv_orient(2, 0))
                    + pv_orient(0, 2) * (pv_orient(1, 0) * missile_coord(2)
                        - missile_coord(1) * pv_orient(2, 0))
                    );


            missile_coordr(2) :=
                    (pv_orient(0, 0) * (pv_orient(1, 1) * missile_coord(2)
                            - pv_orient(2, 1) * missile_coord(1))
                    - pv_orient(0, 1) * (pv_orient(1, 0) * missile_coord(2)
                            - missile_coord(1) * pv_orient(2, 0))
                    + missile_coord(0) * (pv_orient(1, 0) * pv_orient(2, 1)
                            - pv_orient(1, 1) * pv_orient(2, 0))
                    );

                KEW_x := integer(511.5 + 1024.0 *
                    atan((missile_coordr(0)/(-missile_coordr(2)))) / fov );
                KEW_y := integer(255.5 + 1024.0 *
                    atan((missile_coordr(1)/(-missile_coordr(2)))) / fov );

        end if;

        ll_receive(NET, fval);
        threat_coord(j, 0) := fval;
        ll_receive(NET, fval);
        threat_coord(j, 1) := fval;
        ll_receive(NET, fval);
        threat_coord(j, 2) := fval;

        threat_coord(j, 0)
            := threat_coord(j, 0) - observation_point(0);
        threat_coord(j, 1)
            := threat_coord(j, 1) - observation_point(1);
        threat_coord(j, 2)
            := threat_coord(j, 2) - observation_point(2);

--  hio_solve(pv_orient, threat_coordr, threat_coord, j);

-- User Cramers rule to solve, assumes det of matrix := 1

  threat_coordr(j, 0) :=
    (threat_coord(j, 0) * (pv_orient(1, 1) * pv_orient(2, 2)
                    - pv_orient(2, 1) * pv_orient(1, 2))
    - pv_orient(0, 1) * (threat_coord(j, 1) * pv_orient(2, 2)
                    - pv_orient(1, 2) * threat_coord(j, 2))
    + pv_orient(0, 2) * (threat_coord(j, 1) * pv_orient(2, 1)
                    - pv_orient(1, 1) * threat_coord(j, 2))
    );

  threat_coordr(j, 1) :=
    (pv_orient(0, 0) * (threat_coord(j, 1) * pv_orient(2, 2)
                    - threat_coord(j, 2) * pv_orient(1, 2) )
    - threat_coord(j, 0) * (pv_orient(1, 0)*pv_orient(2, 2)
                    - pv_orient(1, 2) * pv_orient(2, 0))
    + pv_orient(0, 2) * (pv_orient(1, 0) * threat_coord(j, 2)
                    - threat_coord(j, 1) * pv_orient(2, 0) )
    );


  threat_coordr(j, 2) :=
    (pv_orient(0, 0) * (pv_orient(1, 1) * threat_coord(j, 2)
                    - pv_orient(2, 1)*threat_coord(j, 1))
    - pv_orient(0, 1) * (pv_orient(1, 0) * threat_coord(j, 2)
                    - threat_coord(j, 1) * pv_orient(2, 0))
    + threat_coord(j, 0) * (pv_orient(1, 0) * pv_orient(2, 1)
                    - pv_orient(1, 1) * pv_orient(2, 0))
    );

        ll_receive(NET, fval);
        miss_distance(j) := fval;
```

```
            threat_x(j) := integer(511.5 + 1024.0 *
                  atan(threat_coordr(j, 0)/(-threat_coordr(j, 2)))
                  / fov );
            threat_y(j) := integer(255.5 + 1024.0 *
                  atan(threat_coordr(j, 1)/(-threat_coordr(j, 2)))
                  / fov );

            j := j + 1;
      end loop; -- while j < Numobj

-- find nearest range and miss distance for display on host

      nearest_miss_distance := 1.0e20;
      nearest_range         := 1.0e20;

      for k in 0..Num_obj-1  loop
            if (miss_distance(k)<nearest_miss_distance) then
                  nearest_miss_distance := miss_distance(k);
            end if;
            if (_range(k)<nearest_range) then
                  nearest_range := _range(k);
            end if;
      end loop;

      if(frame_update = 0 ) then
            if(hw_receive_test(host)) then
                  hw_receive(host, dummy, lsw);
                  dumptohost;
            end if;
      else
            hidden_frame_cnt := hidden_frame_cnt + 1;
            if(hidden_frame_cnt>=frame_update) then
                  ll_receive(host,dummy,lsw);
                  dumptohost;
                  hidden_frame_cnt := 0;
            end if;
      end if;

end dohostio;

begin --main

      initialize_hostio;
      loop
            dohostio;
      end loop;

end hostio;
```

9.3.2 Target Generation Code *threat.ada*

```
#ifndef NET
#define NET net
#endif

procedure threat is

--=============================================================--
--
--    Take the center of Earth as x,y,z=(0,0,0) for earth relative
--  cordinates fpa coordinates is a cartesian coordinate
--  system with the orgin on the interceptor. The negative z axis
--  of this coordinate system extends through the center of
--  the fpa out into the field of view, normal to the fpa.
--  The x axis is runs through the extended plane of the
--  fpa horizontally, and the y axis vertically.
--
--=============================================================--

GRAV       : constant float := 6.67E-11;
EarthRadius : constant float := 6.37E+06;
EarthMass   : constant float := 5.97E+24;
```

```
Arraysize   : constant integer := 128;
Z_OFFSET    : constant float  := 6.37E+06;
RAD_PER_PXEL : constant float := 75.0E-06;
PXEL_PER_RAD : constant float := 13333.3333;  -- 1.0 / RAD_PER_PXEL
Negative_One : constant float := -1.0;
PI          : constant float := 3.14159265;
done    : constant integer := -333;

type A_3x3_float_array is array(0..2, 0..2) of float;
type three_element_float_array is array (0..2) of float;

aim     : A_3x3_float_array;            -- orientation of
                                        --  interceptor in radians

Missile_Coord          : three_element_float_array; -- X of KEW
                                        --  missile World coord
Missile_Velocity       : three_element_float_array; -- Vel of KEW missile World coord
mr    : three_element_float_array;      -- location of this object in
                                        --  missile ref frame
distance : float;
previous_distance : float;
threat_intense : float;                 -- object brightness
Threat_Coord   : three_element_float_array;     -- X of target
Threat_Velocity : three_element_float_array;    -- velocity of target
threat_size : float;                    -- diameter of the target
time_step : float;                      -- Time step to take
velocity : float;
t : float;                              -- time

debugtemp : integer;

temp     : float;
temp_area : float;
-- intens    : float;

Threat_Number : integer;

miss_distance : float;

pixel_x, pixel_y, radius, total_intensity : integer;
intensity : integer;

delta_theta,delta_phi : float;          -- interceptor rotation in radians

delta_x, delta_y : float;

procedure compute_KEW_position is

    r,r3,gx,gy,gz, z : float;
    ar : array(0..1) of float;
    am : three_element_float_array;

begin
    z := Missile_Coord(2) + Z_OFFSET;
    r := sqrt(( Missile_Coord(0)*Missile_Coord(0) + Missile_Coord(1)*Missile_Coord(1) + z*z) );
    r3 := (GRAV * EarthMass) / ( r * r * r );

    gx := -Missile_Coord(0) * r3;
    gy := -Missile_Coord(1) * r3;
    gz := -z * r3;

    ll_receive(NET, delta_x);
    ar(0) := delta_x * time_step;

    ll_receive(NET, delta_y);
    ar(1) := delta_y * time_step;

    am(0) := aim(0,0) * ar(0) + aim(0,1) * ar(1);
    am(1) := aim(1,0) * ar(0) + aim(1,1) * ar(1);
    am(2) := aim(2,0) * ar(0) + aim(2,1) * ar(1);
```

```
      Missile_Velocity(0) := Missile_Velocity(0) + (am(0)+ gx) * time_step;
      Missile_Velocity(1) := Missile_Velocity(1) + (am(1)+ gy) * time_step;
      Missile_Velocity(2) := Missile_Velocity(2) + (am(2)+ gz) * time_step;

      Missile_Coord(0) := Missile_Coord(0) + Missile_Velocity(0) * time_step;
      Missile_Coord(1) := Missile_Coord(1) + Missile_Velocity(1) * time_step;
      Missile_Coord(2) := Missile_Coord(2) + Missile_Velocity(2) * time_step;
end;



-- r = axb
procedure cross_prod(r : out three_element_float_array;
                     a : in  three_element_float_array;
                     b : in  three_element_float_array) is
begin
      r(0) := a(1)*b(2) - a(2)*b(1);
      r(1) := a(2)*b(0) - a(0)*b(2);
      r(2) := a(0)*b(1) - a(1)*b(0);
end;

-- Given thetax, thetay, calculate new position Orientation matrix A
procedure rotate(A      : in out A_3x3_float_array;
                 thetax : in float;
                 thetay : in float) is

      p,q,r : three_element_float_array;
      pmag,qmag,rmag : float;

begin

      -- Calculate rotation about x axis
      for i in 0..2 loop
            p(i) := A(i,0) + A(i,2)*thetay;
            q(i) := A(i,1) + A(i,2)*thetax;
      end loop;

      -- force orthogonal coordinate system
      cross_prod(r,p,q);            -- r=pxq
      cross_prod(q,r,p);            -- q=rxp
      -- Force vectors to be unit length !
      pmag := sqrt(( p(0)*p(0) + p(1)*p(1) + p(2)*p(2)) );
      qmag := sqrt(( q(0)*q(0) + q(1)*q(1) + q(2)*q(2)) );
      rmag := sqrt(( r(0)*r(0) + r(1)*r(1) + r(2)*r(2)) );

      for i in 0..2 loop
            A(i,0) := p(i)/pmag;   --     p0 q0 r0
            A(i,1) := q(i)/qmag;   --  A = p1 q1 r1
            A(i,2) := r(i)/rmag;   --     p2 q2 r2
      end loop;

end;


-- User Cramers rule to solve , assumes det of matrix = 1
procedure solve(A : in A_3x3_float_array;
                x : out three_element_float_array;
                b : in three_element_float_array) is
      tmp1,tmp2,tmp3 : float;
begin

      tmp1 := b(0) * ( A(1,1) * A(2,2) - A(2,1) * A(1,2));
      tmp2 := A(0,1) * ( b(1) * A(2,2) - A(1,2) * b(2));
      tmp3 := A(0,2) * ( b(1) * A(2,1) - A(1,1) * b(2));
      x(0) := tmp1 - tmp2 + tmp3;

      --x(0) := (
      --      b(0) * ( A(1,1)*A(2,2) - A(2,1)*A(1,2) )
      --    - A(0,1) * ( b(1)  *A(2,2) - A(1,2)*b(2)  )
      --    + A(0,2) * ( b(1)  *A(2,1) - A(1,1)*b(2)  )
      --          );
```

```
        tmp1 := A(0,0) * ( b(1) * A(2,2) - b(2) * A(1,2));
        tmp2 := b(0) * ( A(1,0) * A(2,2) - A(1,2) * A(2,0));
        tmp3 := A(0,2) * ( A(1,0) * b(2) - b(1) * A(2,0));
        x(1) := tmp1 - tmp2 + tmp3;

        --x(1) := (
--          A(0,0) * ( b(1)  *A(2,2) - b(2)  *A(1,2) )
--            - b(0)   * ( A(1,0)*A(2,2) - A(1,2)*A(2,0) )
--            + A(0,2) * ( A(1,0)*  b(2) - b(1)  *A(2,0) )
--          );

        tmp1 := A(0,0) * ( A(1,1) * b(2) - A(2,1) * b(1));
        tmp2 := A(0,1) * ( A(1,0) * b(2) - b(1) * A(2,0));
        tmp3 := b(0) * ( A(1,0) * A(2,1) - A(1,1) * A(2,0));
        x(2) := tmp1 - tmp2 + tmp3;

--      x(2) := (
--          A(0,0) * ( A(1,1)*b(2)   - A(2,1)*b(1) )
--            - A(0,1) * ( A(1,0)*b(2)   - b(1)  *A(2,0) )
--            + b(0)   * ( A(1,0)*A(2,1) - A(1,1)*A(2,0) )
--                );

end;

-- compute current position and velocity
procedure ballistic(x    : in out three_element_float_array;
                    v    : in out three_element_float_array;
                    tnow : in out float) is

        r,r3,gx,gy,gz,z : float;

begin

        z := x(2) + Z_OFFSET;
        r := sqrt( ( x(0)*x(0) + x(1)*x(1) + z*z ));
        r3 := (GRAV * EarthMass) / ( r * r * r );
        gx := -x(0) * r3;
        gy := -x(1) * r3;
        gz := -z * r3;

        v(0) := v(0) + gx * time_step;
        v(1) := v(1) + gy * time_step;
        v(2) := v(2) + gz * time_step;

        x(0) := x(0) + v(0) * time_step;
        x(1) := x(1) + v(1) * time_step;
        x(2) := x(2) + v(2) * time_step;

end;



--
-- initialize these things
--initialize_threat_object(aim, threat_intense, Threat_Coord, Threat_Velocity, threat_size, time_step, t );
--

procedure initialize_threat_object is

        temp :float;

begin

miss_distance := -1.0;

-- Receive missile aim AIM

        for i in 0..2 loop
                for j in 0..2 loop
                        ll_receive(host, temp);
                        aim(i,j) := temp;
                        -- ll_send(host,aim(i,j));
```

```
        end loop;
    end loop;

-- Receive missile coordinates XM

    for i in 0..2 loop
        ll_receive(host, temp);
        Missile_Coord(i) := temp;
        -- ll_send(host,Missile_Coord(i));
    end loop;

-- Receive missile velocity VM

    for i in 0..2 loop
        ll_receive(host, temp);
        Missile_Velocity(i) := temp;
        -- ll_send(host,Missile_Velocity(i));
    end loop;

    ll_receive(host, time_step);
    -- ll_send(host, time_step);

    Threat_Number := 0;
    ll_receive(host, Threat_Number, lsw);
    -- ll_send(host, Threat_Number, lsw);

-- Receive threat coordinates XO

    for i in 0..2 loop
        ll_receive(host, temp);
        Threat_Coord(i) := temp;
        -- ll_send(host,Threat_Coord(i));
    end loop;

-- Receive threat velocity VO

    for i in 0..2 loop
        ll_receive(host, temp);
        Threat_Velocity(i) := temp;
        -- ll_send(host,Threat_Velocity(i));
    end loop;

    ll_receive(host, threat_size);
    -- ll_send(host, threat_size);

    ll_receive(host, threat_intense);
    -- ll_send(host, threat_intense);

    t := 0.0;

    previous_distance := 2.0e+30;
    distance := 1.0e+30;
end;



procedure fpa_map is

ft1, ft2 : float;
    -- Calculate which pixels are actually turned on
    --   see note at top for weird stuff in y on missile
    --   relative coordinate system
begin

        radius := integer(atan(threat_size/mr(2)) * PXEL_PER_RAD);
        if (radius < 0) then
            radius := (-radius);
        end if;

        if (mr(2) < 0.0) then
            ft1 := -0.5 + (PXEL_PER_RAD * atan( mr(0)/(-mr(2)) ));
```

```
            pixel_x := ArraySize/2 + integer( ft1 );
            ft2 := -0.5 + (PXEL_PER_RAD * atan( mr(1)/(-mr(2)) ));
            pixel_y := ArraySize/2 + integer( ft2 );

        end if;

        if ((mr(2) > 0.0) or (pixel_x > (ArraySize-1))
                or (pixel_x < 0) or (pixel_y > (ArraySize - 1)) or (pixel_y < 0)) then
            pixel_x := -1;
            pixel_y := -1;
            radius := 0;
        end if;


        -- Here is where we figure out how big it is, and how bright.
        --   Intensity goes as inverse r squared
        total_intensity := integer( threat_intense / (mr(2) * mr(2)));
        if (radius > 1) then
            total_intensity := total_intensity / (radius * radius);
        end if;

        if (total_intensity < 2 ) then
            total_intensity := 1;
        end if;

        if (total_intensity > 63 ) then
            total_intensity := 63;
        end if;

end;


-- finds the relative vector  from xb to xa, then transforms
--  by the inverse of the rotation matrix rot.
--  Result is put in res, note that all of these
--  must be declared!
procedure rel_vector(xa : in three_element_float_array;
                     xb : in three_element_float_array;
                     rot: in A_3x3_float_array;
                     res: out three_element_float_array) is

  d : three_element_float_array;

begin

  d(0) := (xa(0)-xb(0));
  d(1) := (xa(1)-xb(1));
  d(2) := (xa(2)-xb(2));

  solve(rot,res,d);

end;


procedure transpose(A : in A_3x3_float_array;
                    B : out A_3x3_float_array) is

begin

    for i in 0..2 loop
        for j in 0..2 loop
            B(j,i) := A(i,j);
        end loop;
    end loop;
end;

function compute_miss_distance return float is

    t,dt,dist,prev_dist : float;
    dx,tm_Missile_Coord,tm_Threat_Coord : three_element_float_array;

begin
```

```
  dt := 0.00001;
  dist := 1.0e30;
  prev_dist := 2.0e30;
     for i in 0..2 loop
          tm_Missile_Coord(i) := Missile_Coord(i);
          tm_Threat_Coord(i) := Threat_Coord(i);
     end loop;
  while(prev_dist > dist ) loop
     prev_dist := dist;
     for i in 0..2 loop
        tm_Missile_Coord(i) := tm_Missile_Coord(i) – Missile_Velocity(i) * dt;
        tm_Threat_Coord(i) := tm_Threat_Coord(i) – Threat_Velocity(i) * dt;
     end loop;

     for i in 0..2 loop
        dx(i) := tm_Missile_Coord(i) – tm_Threat_Coord(i);
        dx(i) := dx(i) * dx(i);
     end loop;
     dist := sqrt( dx(0) + dx(1) + dx(2) );
  end loop;
  return(prev_dist);
end;

— Main — main —
begin


     pixel_x := 0;
     pixel_y := 0;
     radius := 0;
     intensity :=0;
     distance := 0.0;
     Missile_Coord(0) := 0.0;
     Missile_Coord(1) := 0.0;
     Missile_Coord(2) := 0.0;
     Threat_Coord(0) := 0.0;
     Threat_Coord(1) := 00.0;
     Threat_Coord(2) := 00.0;
     miss_distance := –1.0;

     initialize_threat_object;

     loop
          ll_send(NET, pixel_x,lsw);
          — ll_send(host, pixel_x,lsw);

          ll_send(NET, pixel_y,lsw);
          — ll_send(host, pixel_y,lsw);

          ll_send(NET, radius,lsw);
          — ll_send(host, radius,lsw);

          ll_send(NET, intensity);
          — ll_send(host, intensity);

          ll_send(NET, distance);
          — ll_send(host, distance);

          if Threat_Number = 0 then
               for i in 0..2 loop
                    temp := Missile_Coord(i);
                    ll_send(NET, temp);
                    — ll_send(host, temp);
               end loop;
          end if;

          for i in 0..2 loop
               temp := Threat_Coord(i);
               ll_send(NET, temp);
               — ll_send(host, temp);
          end loop;

          ll_send(NET, miss_distance);
          — ll_send(host, miss_distance);
```

```
            ll_receive(NET, delta_theta);          — delta-theta
            ll_receive(NET, delta_phi);       — delta-phi

          . t := t + time_step;

            — compute ballistic path of this object
            ballistic(Threat_Coord,Threat_Velocity,t);

            compute_KEW_position;
            rotate(aim,delta_theta,delta_phi);    — Update Missile orientation
            rel_vector(Threat_Coord,Missile_Coord,aim,mr);
            fpa_map;

            temp_area := PI * float(radius*radius);
            if (temp_area < 1.0) then
                 temp_area := 1.0;
            end if;

            intensity := total_intensity * integer(temp_area);

            distance := sqrt((mr(0)*mr(0) + mr(1)*mr(1) + mr(2)*mr(2)));

            if (distance > previous_distance) and (miss_distance < 0.0 )  then
                 miss_distance := compute_miss_distance;
            end if;
            previous_distance := distance;

  end loop;
end;
```

9.3.3 Target Selection *track.ada*

```
#ifndef NET
#define NET net
#endif
#define BOGUS bogus := 1.0

procedure TRACK is

—————————————————————————————————————————————————————————————
—
— Procedure object_tracking
—
—    Receives cluster data from clustering chip.
—    Predicts the movement of each cluster in tfEstimateVx,
—    and tfEstimateVy. Receives updated orientation of the
—    FPA in delta_theta and delta_phi.
—
—————————————————————————————————————————————————————————————

  ArraySize      : constant integer :=  128;

  RadiansPerPixel : constant float  := 75.0E-6;
  MaxTracks       : constant integer := 32;
  MaxTracksP1     : constant integer := 33;
  MaxMiss         : constant integer := 4;
  IntensityWeight : constant float  := 0.5;
  AreaWeight      : constant float  := 0.4;
  NumCorWeight    : constant float  := 0.1;
  Gravity         : constant float  := 6.67E-11;
  Earth_Radius    : constant float  := 6.36E06;
  Earth_Mass      : constant float  := 5.97E24;
  FPA_WIN_X       : constant integer := 30;
  FPA_WIN_Y       : constant integer := 50;
  PAN_WIN_X       : constant integer := 20;
  PAN_WIN_Y       : constant integer := 350;
done              : constant integer := -555;

 tcTrackNum     : array (0..33) of integer;
 tcPriority     : array (0..33) of integer;
 tcRPriority    : array (0..33) of float;
 tcPositionX    : array (0..33) of float;
```

```
tcPositionY   : array (0..33) of float;
tcWeightX     : array (0..33) of float;
tcWeightY     : array (0..33) of float;
tcMinDistance : array (0..33) of float;
tcNumCor      : array (0..33) of float;
tcCenID       : array (0..33) of float;
tcCenAcoorX   : array (0..33) of float;
tcCenAcoorY   : array (0..33) of float;
tcCenIcoorX   : array (0..33) of float;
tcCenIcoorY   : array (0..33) of float;
tcRange       : array (0..33) of float;
tcCenArea     : array (0..33) of integer;
tcCenIntensity : array (0..33) of integer;

tfTrackID     : array (0..32) of integer;
tfNumCor      : array (0..32) of integer;
tfNumMiss     : array (0..32) of integer;
tfEstimateX   : array (0..32) of float;
tfEstimateVX  : array (0..32) of float;
tfXP11        : array (0..32) of float;
tfXP12        : array (0..32) of float;
tfXP22        : array (0..32) of float;
tfArea        : array (0..32) of float;
tfEstimateY   : array (0..32) of float;
tfEstimateVY  : array (0..32) of float;
tfYP11        : array (0..32) of float;
tfYP12        : array (0..32) of float;
tfYP22        : array (0..32) of float;
tfCenAcoorX   : array (0..32) of float;
tfCenAcoorY   : array (0..32) of float;
tfCenIcoorX   : array (0..32) of float;
tfCenIcoorY   : array (0..32) of float;
tfRange       : array (0..32) of float;
tfIntensity   : array (0..32) of integer;
tfCSO         : array (0..32) of integer;

cID       : integer;
cAcoorX   : float;
cAcoorY   : float;
cIcoorX   : float;
cIcoorY   : float;
cArea     : integer;
cIntensity : integer;
cRange    : float;

pixel_x, pixel_y        : integer;

LastTrack, LastNew  : integer;
TrackID, CentroidID : integer;
ptr, lowptr         : integer;
dx, dy, dist2, priority, lowpriority : float;
NotMatchCentroid    : boolean;
LastCentroid        : integer;

    temp : float;
    debugtemp : integer;
    bogus : float;

iterations    : integer;
dt, dt2, dtsqr : float;

RadPix2, PixelsPerRadian, ProcessNoise, MeasurementNoise : float;
initialP11, initialP12, initialP22, maxNumCor  : float;
PixelOffset : float;

x_dot, y_dot : float;
TargetIntensity : float;
mge : float;
MaxDist, time : float;
previous_x, previous_y : float;
Num_obj : integer;
FinalAcquisition : boolean;
TrackingRange : float;
```

```
PixelTrackingRange : integer;
FinalAcquisitionRange : float;

TargetSelectTime, delta_theta, delta_phi, theta_dot, phi_dot : float;

TargetTrack : integer;
TargetSelected : boolean;

frame  : integer;
error  : boolean;

-- Variables local to main routine.
xdot_range_comp, y_dot_range_comp, a : float;

-------------------------------------------------------------------------

procedure KalmanTimeUpdate (position : in out float;
                            velocity : in out float;
                            P11      : in out float;
                            P12      : in out float;
                            P22      : in out float) is

  begin
    position := position + (dt * velocity);
    P11 := P11 + ((dt2 * P12) + (dtsqr * P22));
    P12 := P12 + (dt * P22);
    P22 := P22 + ProcessNoise;
  end KalmanTimeUpdate;

-------------------------------------------------------------------------

procedure KalmanMeasurementUpdate (MeasuredPosition : in float;
            position : in out float;
                       velocity : in out float;
                       P11      : in out float;
                       P12      : in out float;
                       P22      : in out float) is


  temp, K1, K2, StateError : float;

 begin
    StateError := MeasuredPosition - position;
    temp := 1.0 / (P11 + MeasurementNoise);
    K1 := temp * P11;
    K2 := temp * P12;
    temp := P12;
    P11 := (1.0 - K1) * P11;
    P12 := (1.0 - K1) * P12;
    P22 := P22 - (K2 * temp);
    position := position + (K1 * StateError);
    velocity := velocity + (K2 * StateError);
 end;

-------------------------------------------------------------------------

procedure Initialize_track_object is

 begin

      ll_receive(host,dt);
      dt2 := 2.0 * dt;
      dtsqr := dt * dt;

      FinalAcquisition := false;

      RadPix2        := RadiansPerPixel * RadiansPerPixel;
      PixelsPerRadian :=  1.0 / RadiansPerPixel;
      ProcessNoise    :=  2.5 * RadPix2;
      MeasurementNoise :=  1.0 * RadPix2;

      TargetTrack := 0;
```

```
      InitialP11     :=  2.0 * RadPix2;
      InitialP12     :=  0.0 * RadPix2;
      InitialP22     := 100.0 * RadPix2;

      PixelOffset    := float(ArraySize) / 2.0 - 0.5;

   LastTrack := 0;
   LastNew := 0;
   for i in 0..MaxTracks loop
      tcTrackNum(i) := i;
      tcPriority(i) := 0;
   end loop;
   TrackID := 0;
   TargetSelected := false;
   time := 0.0;
   x_dot := 0.0;
   y_dot := 0.0;
   theta_dot := 0.0;
   phi_dot := 0.0;

      frame := 0;
      error := false;

   ll_receive(host,TargetSelectTime);
      Num_obj := 0;
      ll_receive(host,Num_obj,lsw);
      ll_receive(host,FinalAcquisitionRange);
      PixelTrackingRange := 0;
      ll_receive(host,PixelTrackingRange,lsw);

     TrackingRange := float(PixelTrackingRange) * RadiansPerPixel;
end Initialize_track_object;
```

```
procedure Compensate_FPA_Rotation is
begin
   for i in 0..LastTrack-1 loop
      tfEstimateX(tcTrackNum(i)) := tfEstimateX(tcTrackNum(i)) - delta_phi;
      tfEstimateY(tcTrackNum(i)) := tfEstimateY(tcTrackNum(i)) - delta_theta;
   end loop;
end;
```

```
procedure UpdateTracks is
begin
 for i in 0..(LastTrack-1) loop
   -- check for multiple centroid associations --
   IF (tcNumCor(i) > 0.0) AND (tcCenID(i) >= 0.0) THEN
      ptr := i + 1;
      WHILE (ptr < LastTrack) loop
       IF (tcCenID(i)  = tcCenID(ptr)) THEN
          IF (tcMinDistance(i) > tcMinDistance(ptr)) THEN
              tcCenID(i) := -1.0;
          ELSE
              tcCenID(ptr) := -1.0;
                  END IF;
             end if;
        ptr := ptr + 1;

      end loop;
       end if;
   IF (tcNumCor(i) = 0.0) THEN
      -- no correlation with track file --
      IF tfNumMiss(tcTrackNum(i)) >= MaxMiss THEN
         tcPriority(i) := 0;
      ELSE
        tfNumMiss(tcTrackNum(i)) := tfNumMiss(tcTrackNum(i)) + 1;
        tcPriority(i) := tfNumCor(tcTrackNum(i)) - tfNumMiss(tcTrackNum(i));
      end if;
   ELSE
       IF  (tcCenID(i) = (-1.0)) THEN
```

```
    — correlation but another track file is closer —

     tfNumMiss(tcTrackNum(i)) := 0;
     tcPriority(i) := tfNumCor(tcTrackNum(i));
     KalmanMeasurementUpdate(tcPositionX(i),
                             tfEstimateX(tcTrackNum(i)),
                             tfEstimateVX(tcTrackNum(i)),
                          tfXP11(tcTrackNum(i)),
                             tfXP12(tcTrackNum(i)),
                             tfXP22(tcTrackNum(i)));
     KalmanMeasurementUpdate(tcPositionY(i),
                             tfEstimateY(tcTrackNum(i)),
                             tfEstimateVY(tcTrackNum(i)),
                          tfYP11(tcTrackNum(i)),
                             tfYP12(tcTrackNum(i)),
                             tfYP22(tcTrackNum(i)));

   ELSE
     — update from correlated centroid —

     tfNumCor(tcTrackNum(i)) := tfNumCor(tcTrackNum(i)) + 1;
     tfNumMiss(tcTrackNum(i)) := 0;
     tcPriority(i) := tfNumCor(tcTrackNum(i));
     tfIntensity(tcTrackNum(i)) := tcCenIntensity(i);
      tfArea(tcTrackNum(i)) := float(tcCenArea(i));
      tfRange(tcTrackNum(i)) := tcRange(i);
      tfCenAcoorX(tcTrackNum(i)) := tcCenAcoorX(i);
      tfCenAcoorY(tcTrackNum(i)) := tcCenAcoorY(i);
      tfCenIcoorX(tcTrackNum(i)) := tcCenIcoorX(i);
      tfCenIcoorY(tcTrackNum(i)) := tcCenIcoorY(i);

     KalmanMeasurementUpdate(tcCenAcoorX(i),
                             tfEstimateX(tcTrackNum(i)),
                             tfEstimateVX(tcTrackNum(i)),
                          tfXP11(tcTrackNum(i)),
                             tfXP12(tcTrackNum(i)),
                             tfXP22(tcTrackNum(i)));
     KalmanMeasurementUpdate(tcCenAcoorY(i),
                             tfEstimateY(tcTrackNum(i)),
                             tfEstimateVY(tcTrackNum(i)),
                          tfYP11(tcTrackNum(i)),
                             tfYP12(tcTrackNum(i)),
                             tfYP22(tcTrackNum(i)));
    end if;
   end if;
  end loop;
end;
```

```
procedure InitializeNewTracks is

begin
  for i in LastTrack..(LastNew – 1) loop
     tfTrackID(tcTrackNum(i)) := TrackID;
     TrackID := TrackID + 1;
     tfNumCor(tcTrackNum(i)) := 1;
     tfNumMiss(tcTrackNum(i)) := 0;
     tcPriority(i) := 1;

     tfIntensity(tcTrackNum(i)) := tcCenIntensity(i);
     tfArea(tcTrackNum(i)) := float(tcCenArea(i));
       tfRange(tcTrackNum(i)) := tcRange(i);
     tfEstimateX(tcTrackNum(i)) := tcCenAcoorX(i);
     tfEstimateVX(tcTrackNum(i)) := 0.0;
     tfXP11(tcTrackNum(i)) := initialP11;
     tfXP12(tcTrackNum(i)) := initialP12;
     tfXP22(tcTrackNum(i)) := initialP22;
     tfEstimateY(tcTrackNum(i)) := tcCenAcoorY(i);
     tfEstimateVY(tcTrackNum(i)) := 0.0;
     tfYP11(tcTrackNum(i)) := initialP11;
     tfYP12(tcTrackNum(i)) := initialP12;
     tfYP22(tcTrackNum(i)) := initialP22;
     tfCenAcoorX(tcTrackNum(i)) := tcCenAcoorX(i);
```

```
    tfCenAcoorY(tcTrackNum(i)) := tcCenAcoorY(i);
    tfCenIcoorX(tcTrackNum(i)) := tcCenIcoorX(i);
    tfCenIcoorY(tcTrackNum(i)) := tcCenIcoorY(i);

  end loop;
end;
```

```
procedure RankTracks is

 temp1, temp2 : integer;

begin

 IF (LastNew > 1) THEN
   -- sort on priority --
   for i in 0..LastNew-2 loop

     ptr := i;
     for j in (i+1)..(LastNew-1) loop
      IF (tcPriority(i) < tcPriority(j)) THEN
                ptr := j;
          end if;
          end loop;
      IF (ptr /= i) THEN

         temp1 := tcTrackNum(ptr);
         temp2 := tcPriority(ptr);
         tcTrackNum(ptr) := tcTrackNum(i);
         tcPriority(ptr) := tcPriority(i);
         tcTrackNum(i) := temp1;
         tcPriority(i) := temp2;
       end if;
     end loop;

  end if;
 LastTrack := 0;

 WHILE (tcPriority(LastTrack) > 0) loop
  LastTrack := LastTrack + 1;
     temp1 := tcPriority(LastTrack);
 end loop;

 LastNew := LastTrack;
end;
```

```
procedure SetUpCorrelation is

begin
 for i in 0..(LastTrack - 1) loop
    KalmanTimeUpdate( tfEstimateX(tcTrackNum(i)),
                tfEstimateVX(tcTrackNum(i)),
                tfXP11(tcTrackNum(i)),
                tfXP12(tcTrackNum(i)),
                tfXP22(tcTrackNum(i)));
    KalmanTimeUpdate( tfEstimateY(tcTrackNum(i)),
                tfEstimateVY(tcTrackNum(i)),
                tfYP11(tcTracknum(i)),
                tfYP12(tcTrackNum(i)),
                tfYP22(tcTrackNum(i)));

    tcPositionX(i)  := tfEstimateX(tcTrackNum(i));
    tcPositionY(i)  := tfEstimateY(tcTrackNum(i));
    tcWeightX(i)    := 1.0 / tfXP11(tcTrackNum(i));
    tcWeightY(i)    := 1.0 / tfYP11(tcTrackNum(i));
    tcMinDistance(i) := MaxDist;
    tcNumCor(i)     := 0.0;
 end loop;
end;
```

```
procedure InputCentroid is

 iAcoorX, iAcoorY, iIcoorX, iIcoorY : float;
 temp : float;

begin
      pixel_x := 0;
      pixel_y := 0;
      ll_receive(NET,pixel_x,lsw);
      ll_receive(NET,pixel_y,lsw);
      iAcoorX := float(pixel_x);
      iAcoorY := float(pixel_y);
      iIcoorX := iAcoorX;
      iIcoorY := iAcoorY;

      cArea := 0;
      ll_receive(NET, cArea ,lsw);
      ll_receive(NET,cIntensity);
      ll_receive(NET,cRange);

 cID := CentroidID;
 CentroidID  := CentroidID + 1;

 cAcoorX := (iAcoorX – PixelOffset) * RadiansPerPixel;
 cAcoorY := (iAcoorY – PixelOffset) * RadiansPerPixel;
 cIcoorX := (iIcoorX – PixelOffset) * RadiansPerPixel;
 cIcoorY := (iIcoorY – PixelOffset) * RadiansPerPixel;

/*    debugtemp := 1;
      ll_send(host,debugtemp,lsw); */

end;
```

```
procedure PerformTracking is
      centroid_ctr : integer;
begin
 centroid_ctr := 0;
 Compensate_FPA_Rotation;
 SetUpCorrelation;
 LastCentroid := 0;
 CentroidID := 0;
 WHILE (centroid_ctr < Num_obj) loop
      centroid_ctr := centroid_ctr + 1;
      InputCentroid;

—     if cArea > 0 then
                — compare against tracks —
                NotMatchCentroid := TRUE;
                ptr := 0;
                WHILE (ptr < LastTrack) loop
                     dx := cAcoorX – tcPositionX(ptr);
                     dy := cAcoorY – tcPositionY(ptr);
                     dist2 := tcWeightX(ptr)*(dx*dx) + tcWeightY(ptr) * (dy*dy);
                     IF dist2 < MaxDist THEN
                          — correlated with this track file —
                               tcNumCor(ptr) := tcNumCor(ptr) + 1.0;
                               NotMatchCentroid := FALSE;
                               IF dist2 < tcMinDistance(ptr) THEN

                                    tcMinDistance(ptr)  := dist2;
                                    tcCenID(ptr)        := float(cID);
                                    tcCenAcoorX(ptr)    := cAcoorX;
                                    tcCenAcoorY(ptr)    := cAcoorY;
                                    tcCenIcoorX(ptr)    := cIcoorX;
                                    tcCenIcoorY(ptr)    := cIcoorY;
                                    tcCenArea(ptr)      := cArea;
                               tcRange(ptr)        := cRange;
                                    tcCenIntensity(ptr) := cIntensity;
                               end if;
```

```
                end if;
                ptr := ptr + 1;
            end loop;

    -- compare against uncorrelated centroids --

            IF (NotMatchCentroid AND (ptr < MaxTracks)) THEN
                -- check to start new track --
                priority := cAcoorX*cAcoorX + cAcoorY*cAcoorY;
                IF LastNew < MaxTracks THEN
                    -- room to start new track --
                  tcRPriority(LastNew)   := priority;
                  tcCenID(LastNew)       := float(cID);
                  tcCenAcoorX(LastNew)   := cAcoorX;
                  tcCenAcoorY(LastNew)   := cAcoorY;
                  tcCenIcoorX(LastNew)   := cIcoorX;
                  tcCenIcoorY(LastNew)   := cIcoorY;
                  tcCenArea(LastNew)     := cArea;
              tcRange(LastNew)       := cRange;
                  tcCenIntensity(LastNew) := cIntensity;
                  LastNew := LastNew + 1;
                ELSE
                    -- check other new track for priority --

                  lowpriority := -1.0;
                  lowptr := LastNew;
                  WHILE (ptr < MaxTracks) loop

                      IF (lowpriority < float(tcPriority(ptr))) THEN
                          lowptr := ptr;
                          lowpriority := float(tcPriority(ptr));
                      end if;
                      ptr := ptr + 1;
                  end loop;
                  IF (priority < lowpriority) THEN
                      tcRPriority(lowptr)   := priority;
                      tcCenID(lowptr)       := float(cID);
                      tcCenAcoorX(lowptr)   := cAcoorX;
                      tcCenAcoorY(lowptr)   := cAcoorY;
                      tcCenIcoorX(lowptr)   := cIcoorX;
                      tcCenIcoorY(lowptr)   := cIcoorY;
                  --  tcCenArea(lowptr)     := cArea;
                  --  tcRange(lowptr)       := cRange;

    -- Tan had an error in his code it read as follows :
                tcCenArea(lowptr)     := integer(cRange);
    -- He didnt change the variable name
                      tcCenIntensity(lowptr) := cIntensity;
                  end if;
              end if;
          end if;

    --    end if;
    end loop;

  UpdateTracks;
  InitializeNewTracks;
  RankTracks;

end;
```

```
procedure SelectTarget is
    maxArea, maxNumCor, Weight, TargetWeight : float;
    maxIntensity : integer;
begin
    maxNumCor := 0.0;
    maxArea := 0.0;
    maxIntensity := 0;
    TargetIntensity := 0.0;

-- Select the brightest object among objects that meets minimum
--    number of Correlation.
```

```
        for i in 0..LastTrack - 1 loop
            if tfNumCor(i) > integer(maxNumCor) then
                maxNumCor := float(tfNumCor(i));
            end if;
            if tfArea(i) > maxArea then
                maxArea := tfArea(i);
            end if;
            if tfIntensity(i) > maxIntensity then
                maxIntensity := tfIntensity(i);
            end if;
        end loop;
        TargetWeight := 0.0;
        TargetTrack := -1;

        for i in 0.. LastTrack - 1 loop
            Weight := (NumCorWeight * float(tfNumCor(i))/maxNumCor) +
                      (AreaWeight * tfArea(i)/maxArea) +
                      (IntensityWeight * float(tfIntensity(i))/
                          float(maxIntensity));
            if Weight > TargetWeight then
                TargetTrack := i;
                TargetWeight := Weight;
            end if;
        end loop;

        if (LastTrack > 0) then
            TargetSelected := true;
        end if;

        BOGUS; BOGUS; BOGUS;
--      TargetSelected := true;
--      TargetTrack := 0;
end;

-------------------------------------------------------------------------

procedure FindTarget is

begin
        TargetIntensity := 0.0;
-- Select the brightest object among objects with zero misses
        for i in 0..LastTrack - 1 loop
            if (tfNumMiss(i) = 0) and (float(tfIntensity(i)) > TargetIntensity)
              then
                TargetIntensity := float(tfIntensity(i));
                TargetTrack := i;
            end if;
        end loop;
end;

-------------------------------------------------------------------------
-- Main --

begin

        Initialize_track_object;

        x_dot := 0.0;
        y_dot := 0.0;
        phi_dot := 0.0;
        theta_dot := 0.0;

    loop

            ll_send(NET,x_dot);
            ll_send(NET,y_dot);
            ll_send(NET,phi_dot);
            ll_send(NET,theta_dot);

            debugtemp := 2;
/*          ll_send(host,debugtemp,lsw); */
```

```
ll_receive(NET,delta_theta);
ll_receive(NET,delta_phi);
rnge := tfRange(TargetTrack);

if rnge > 80.0E03 then
      MaxDist := 6.0;
else
      MaxDist := (80.0E03/rnge) * 6.0;
end if;

PerformTracking;
frame := frame + 1;
time := time + dt;

if (time >= TargetSelectTime) then
      if (not TargetSelected) then
            SelectTarget;

            phi_dot := tfEstimateX(TargetTrack) - 0.5 *
                  RadiansPerPixel;
            theta_dot := tfEstimateY(TargetTrack) -
                  0.5 * RadiansPerPixel;
            x_dot := rnge * tan(tfEstimateVX(TargetTrack))/dt;
            y_dot := rnge * tan(tfEstimateVY(TargetTrack))/dt;


      else
            if (rnge < FinalAcquisitionRange) and
                  ((tfNumMiss(TargetTrack) > 0) or
                   (FinalAcquisition)) then

                  FinalAcquisition := TRUE;
                  FindTarget;
                  rnge := tfRange(TargetTrack);

                  x_dot := rnge * tan(tfEstimateX(TargetTrack)
                        - 0.5 * RadiansperPixel) / dt;
                  y_dot := rnge * tan(tfEstimateY(TargetTrack)
                        - 0.5 * RadiansperPixel) / dt;
                  phi_dot := tfEstimateX(TargetTrack) -
                        0.5 * RadiansperPixel;
                  theta_dot := tfEstimateY(TargetTrack) -
                        0.5 * RadiansperPixel;

            else
                  if tfNumMiss(TargetTrack) > 0 then

                        FindTarget;
                        rnge := tfRange(TargetTrack);
                  end if;

                  if (tfEstimateX(TargetTrack) > TrackingRange)
                        or (tfEstimateX(TargetTrack) < -TrackingRange)
                   then

                        phi_dot := tfEstimateX(TargetTrack) -
                              0.5 * RadiansperPixel;
                  else
                        phi_dot := 0.0;
                  end if;

                  if (tfEstimateY(TargetTrack) > TrackingRange)
                    or (tfEstimateY(TargetTrack) <
                         -TrackingRange) then

                        theta_dot := tfEstimateY(TargetTrack)
                        - 0.5 * RadiansperPixel;
                  else
                        theta_dot := 0.0;
                  end if;

                  x_dot := rnge * tan(tfEstimateVX(TargetTrack))
                        / dt;
```

```
                    y_dot := mge * tan(tfEstimateVY(TargetTrack))
                        / dt;
                end if;
                previous_x := tfEstimateX(TargetTrack);
                previous_y := tfEstimateY(TargetTrack);
            end if;
        end if;


 end loop;

end;
```

9.3.4 Guidance *los_gnc.ada*

```
#ifndef NET
#define NET net
#endif

procedure los_gnc is

    angNavConst, linNavConst, thruster_interval,
        time_step, thruster_time, max_divert : float;
    divert: float;
— compute zero mass line of sight guidance law
    delta_theta, delta_phi, delta_x, delta_y : float;
    x_dot, y_dot, phi_dot, theta_dot : float;

    debugtemp : integer;

procedure initialize_los_gnc is
begin
    ll_receive(host, angNavConst);
    ll_receive(host, linNavConst);
    ll_receive(host, thruster_interval);
    ll_receive(host, time_step);
    ll_receive(host, max_divert);

    ll_send(host, angNavConst);
    ll_send(host, linNavConst);
    ll_send(host, thruster_interval);
    ll_send(host, time_step);
    ll_send(host, max_divert);

    thruster_time := thruster_interval;
end;

 — Main
begin

    initialize_los_gnc;
    delta_theta := 0.0;
    delta_phi := 0.0;
    delta_x := 0.0;
    delta_y := 0.0;
  loop
        ll_send(NET, delta_theta);
        ll_send(NET, delta_phi);
        ll_send(NET, delta_x);
        ll_send(NET, delta_y);

/*      debugtemp := 1;
        ll_send(host,debugtemp,lsw); */

        ll_receive(NET, x_dot);
        ll_receive(NET, y_dot);
        ll_receive(NET, phi_dot);
        ll_receive(NET, theta_dot);

/*      debugtemp := 2;
        ll_send(host,debugtemp,lsw); */
```

```
    if thruster_time >= thruster_interval then
        divert := linNavConst * x_dot;
        if (divert > max_divert) then
            divert := max_divert;   — Limit divert
        else
          if (divert< (–max_divert) ) then
            divert := –max_divert;
          end if;
        end if;

        delta_x := divert;

        divert := linNavConst * y_dot;
        if (divert>max_divert) then
            divert := max_divert;
        else
            if (divert < (–max_divert) ) then
                divert := –max_divert;
            end if;
        end if;

        delta_y := divert;

        delta_phi := phi_dot * angNavConst;
        delta_theta := theta_dot * angNavConst;

        thruster_time := time_step;

    else

        thruster_time := thruster_time + time_step;

        delta_x := 0.0;
        delta_y := 0.0;
        delta_theta := 0.0;
        delta_phi := 0.0;

    end if;

end loop;
end;
```