AMES GRANT
IN-32-CR
3344
P-155

T · H · E
OHIO
STATE
UNIVERSITY

A System for the Real-Time Display of Radar and Video Images of Targets

W.W. Allen and W.D. Burnside

(NASA-CR-188058) A SYSTEM FOR THE REAL-TIME DISPLAY OF RADAR AND VIDEO IMAGES OF TARGETS (Ohio State Univ.) 155 p CSCL 17I

N91-21394

Unclas
G3/32 0003344

The Ohio State University

ElectroScience Laboratory

Department of Electrical Engineering
Columbus, Ohio 43212

Technical Report 722780-1
Grant No. NAG 2-542, Supp. No. 2
August 1990

National Aeronautics and Space Administration
Ames Research Center
Moffett Field, CA 94035

and

Pacific Missile Test Center
Point Mugu, CA 93042

NOTICES

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO.	2.	3. Recipient's Accession No.
4. Title and Subtitle A System for the Real-Time Display of Radar and Video Images of Targets			5. Report Date August 1990 6.
7. Author(s) W.W. Allen and W.D. Burnside			8. Performing Org. Rept. No. 722780-1
9. Performing Organization Name and Address The Ohio State University ElectroScience Laboratory 1320 Kinnear Road Columbus, OH 43212			10. Project/Task/Work Unit No. 11. Contract(C) or Grant(G) No. (C) (G) NAG 2-542, Supp. No. 2
12. Sponsoring Organization Name and Address NASA Ames Research Center Moffett Field, CA 94035	Pacific Missile Test Center Point Mugu, CA 93042		13. Report Type/Period Covered Technical Report 14.
15. Supplementary Notes			

16. Abstract (Limit: 200 words)

This report describes a software and hardware system for the real-time display of radar and video images for use in a measurement range. Its main purpose is to give the reader a clear idea of the software and hardware design and its functions. This system is designed around a Tektronix XD88-30 graphics workstation, used to display radar images superimposed on video images of the actual target. The system's purpose is to provide a platform for the analysis and documentation of radar images and their associated targets in a menu-driven, user-oriented environment.

17. Document Analysis a. Descriptors

IMAGING	2-D
SIGNAL PROCESSING	3-D
SIGNATURES	VISUALIZATION

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement A. Approved for public release; Distribution is unlimited.	19. Security Class (This Report) Unclassified	21. No. of Pages 155
	20. Security Class (This Page) Unclassified	22. Price

(See ANSI-Z39.18) *See Instructions on Reverse* OPTIONAL FORM 272 (4-77) Department of Commerce

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The documentation and visual correlation of radar images with their associated targets is a continuing problem since a radar image is not necessarily recognizable on its own. Figure 1 is an inverse synthetic aperture radar (ISAR) image of an aircraft test body [2]. This problem is particularly acute for high speed, high resolution imaging techniques, such as the 3-D scattering center imaging as proposed by Dominek, et al. [1] The usual solution to this problem is to superimpose the radar image on a scaled outline drawing of the target. Figure 2 shows an example of this method, showing the ISAR image of Figure 1 superimposed on an outline drawing of the target [2]. This technique is used for both computer displays and for publications. While this gives the observer information about the outline surface of the target, it does not convey information about specific target features, such as joints, window openings, etc., all of which can cause significant radar image features. Further, it does not provide documentation of the target itself or any modifications that might have been carried out on the target, such as the application of absorbent material to specific areas of the target.

A far better solution to this problem is proposed in this report. If, instead of an outline drawing, let us imagine the radar image superimposed with a video image of the actual target. In this way, many more features of the target become evident,such as joints, window openings, surface treatment, fasteners (bolt and

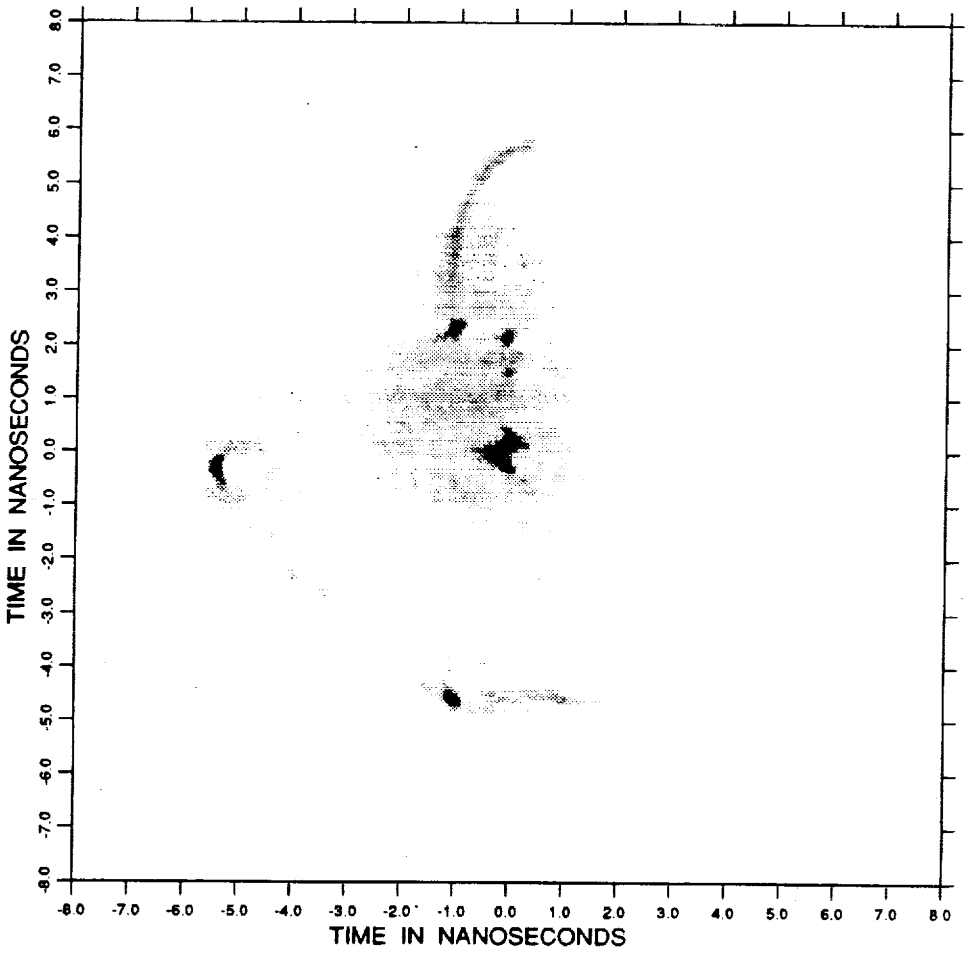


Figure 1: ISAR image of an aircraft test body.

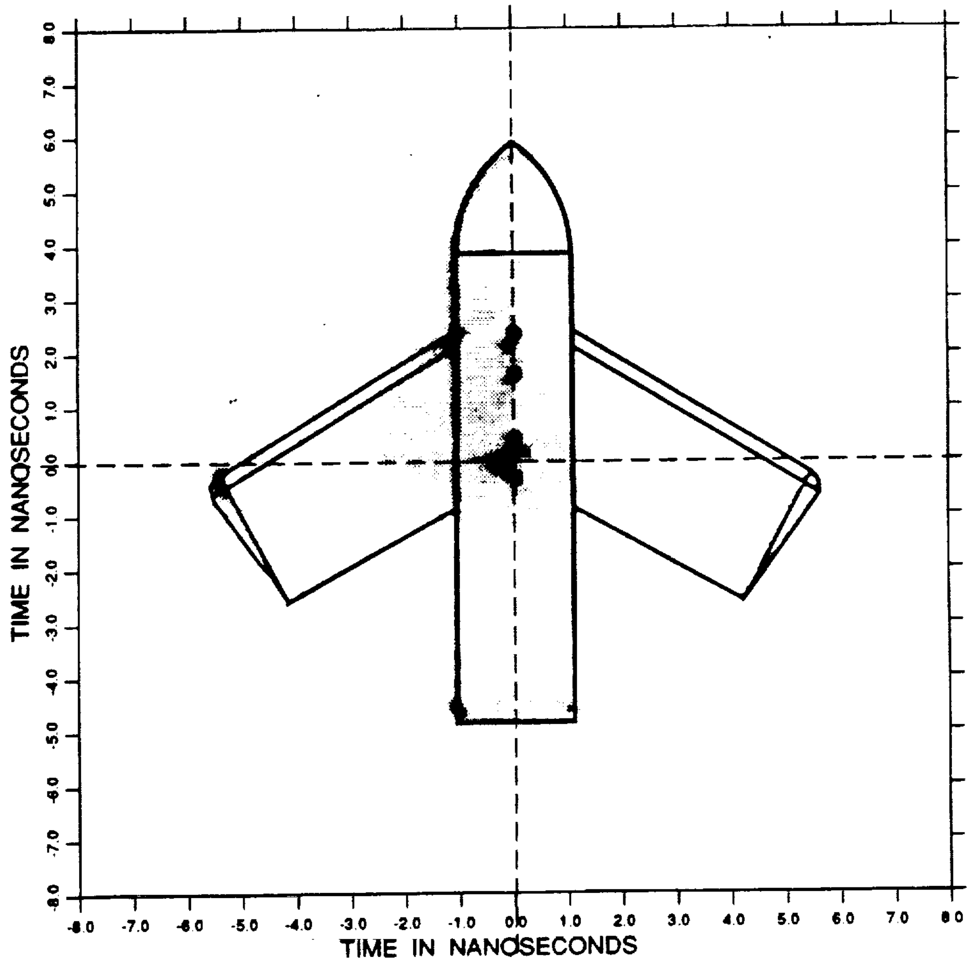


Figure 2: ISAR image with outline of target.

rivet heads), etc., can be easily identified. Additionally, it provides unambiguous documentation of the target and modifications made to the target. Figure 3 is a screen print from the system described in this report. (Note: magnitude data in the radar images in this report has been suppressed to offset the limitations of black and white reproduction.) The figure shows the same target as in Figure 2 with a scattering center radar image superimposed on it. Clearly, the method described herein provides much more information about the target than in Figure 2. Such a system can also be an extremely useful diagnostic tool for diagnostic radar ranges, serving as a visual check on the validity of the data during the data collection and processing sequence.

ORIGINAL PAGE IS
OF POOR QUALITY

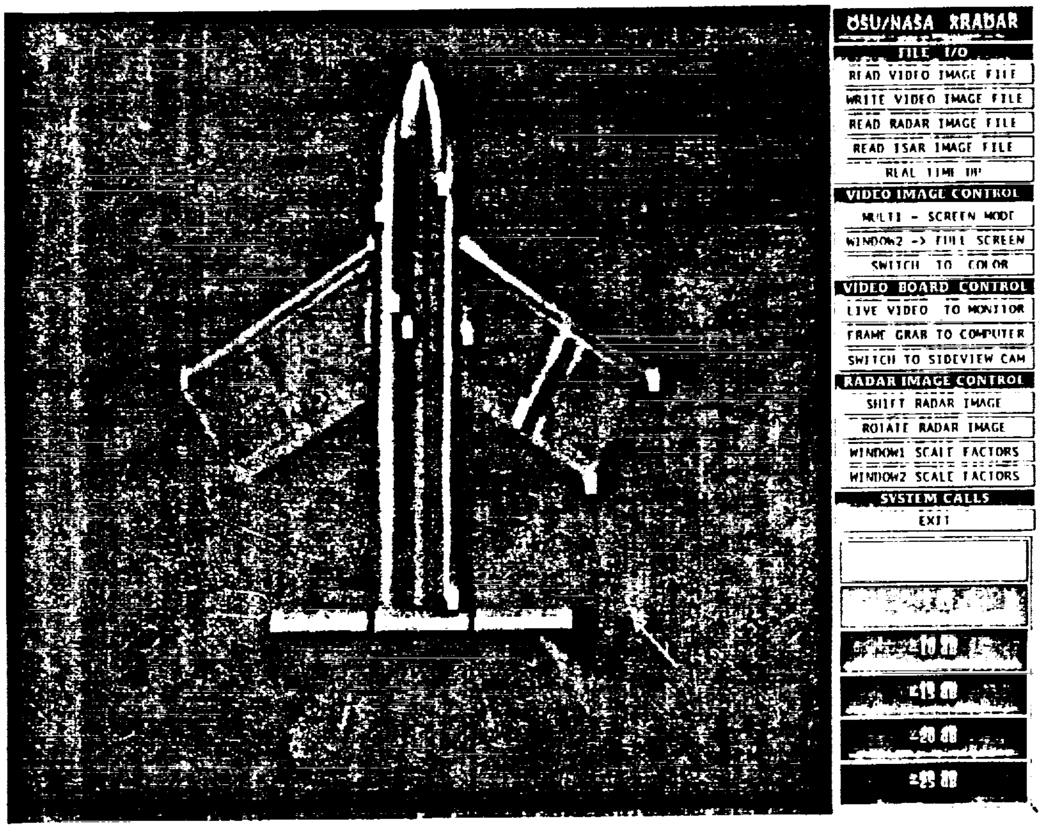


Figure 3: Scattering center image superimposed on a video image of the target.

The system described in this report is composed of a high-speed graphics workstation interfaced to two video cameras. It is also interfaced to the data acquisition computer which controls the compact radar range hardware. Two independent pieces of software running concurrently on the graphics workstation handle the data processing and the radar/video image display. This report focuses on the video/radar image acquisition and display portion of the system. This system was developed for use in the compact radar range measurement facility at the Ohio State University ElectroScience Laboratory.

Chapter 2

Compact Range and Radar Imaging Principles

While there are several different techniques for generating radar images, they are all closely related to radar cross section (RCS) measurement and analysis. High resolution time or frequency domain data is required to produce these images, so any discussion of radar imaging techniques must begin with a description of the techniques used to generate this data.

2.1 Compact Range Theory and Operation

Pure time domain measurements are difficult to achieve, so an equally viable method of producing this data is to measure data in the frequency domain and then transform it, using the inverse Fourier transform, to the time domain. A diagram of an RCS measurement system is shown in Figure 4. The important elements in the system are a swept-frequency transmitter, a receiver, a broad-band antenna system, a recording and processing system and a target pedestal which may be rotated to position the target. For most imaging techniques, the target must be in the far field zone of the antenna, and in order to conserve space, this is achieved by placing the feed antenna at the focus of a parabolic reflector. The parabolic reflector produces a plane wave in the area of the target, and the system is known as a compact range RCS measurement system.

Three basic problems must be overcome in any RCS measurement system. First, the system must be calibrated to compensate for variations in system gain. Second, clutter from scatterers other than the target must be reduced to acceptable levels. And finally, compensation must be made for system drift.

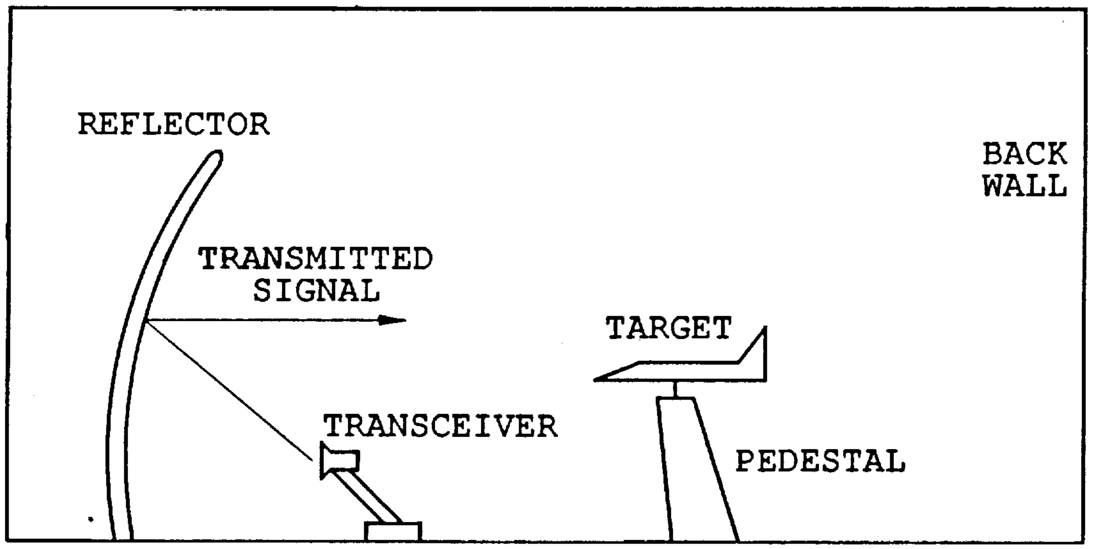


Figure 4: A compact RCS range

2.1.1 System Calibration

The variations in system gain as a function of frequency are determined and compensated for by comparing measured RCS data with the theoretical RCS data of the same target. A sphere is usually used as the target for this procedure, as the theoretical RCS of a sphere as a function of frequency is easily calculated. The following equation defines the system gain :

$$SystemGain = \frac{ReferenceData}{TheoreticalData} \tag{1}$$

where Reference Data is the measured backscattered fields of a sphere as a function of frequency and Theoretical Data is the backscattered fields of a sphere as a function of frequency as calculated by the MEI solution [3].

The target data is calibrated by dividing it by the system gain, giving the following calibration equation :

$$CalibratedData = \frac{RawData}{ReferenceData} \cdot TheoreticalData \qquad (2)$$

where Raw Data is the measured backscattered fields as a function of frequency of the target under measurement.

This allows the variations in system gain to be removed from the raw data before performing the transformation to the time domain.

2.1.2 Clutter Reduction

Virtually any object, aside from the target itself, in the radar range will cause clutter which interferes with desired the RCS measurement. The antenna mismatch, the parabolic reflector, and the walls of the room are all major clutter sources. However, most of these components can be removed from the received signal by time gating. Figure 5 illustrates the timing of these various components. At time t_0, a transmit pulse is sent to the feed antenna. There is an immediate return due to the impedance mismatch in the antenna. At time t_1, the receiver receives the direct return from the reflector. At time t_2, the desired signal from the target is received and finally, at time t_3, the reflection from the back wall of the room is received. So that only the desired return from the target is received, the receiver is only enabled during the 'range gate' surrounding the target return. The timing and length of the range gate are determined by the position and size of the target so that unwanted reflections are kept to a minimum. Even with the range gating technique, there is still enough residual clutter that another technique must be applied. This is known as background subtraction.

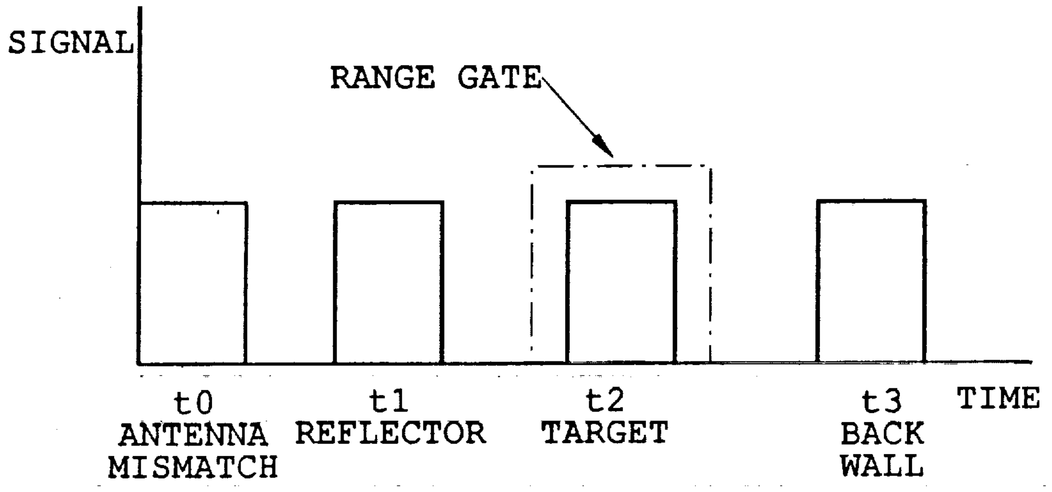


Figure 5: The timing diagram of various clutter and target returns.

Background subtraction may be used to remove clutter which is present within the receive range gate since the system can measure the complex scattered fields. This clutter is from scatterers in close proximity to the target itself, such as the target pedestal and absorber. A background scan is performed first, without the target in place. RCS scans of the target are then made and the background scan data is subtracted from this data. This process removes the contributions from the unwanted scatterers in the frequency domain data. This process works successfully provided that there is little or no electromagnetic interaction between the target and the pedestal or chamber.

2.1.3 Compensation for System Drift

Any analog electronic equipment is subject to long term drift due to temperature and aging effects on the components. The effects of this drift can be removed from the data by comparing the measured raw data with data from a fixed target which is always present in the range. In the OSU compact range, the backscatter from the parabolic reflector is used as this reference target. The compensation is performed by switching the receive gate between the reference and target returns.

The target data is then scaled to the reference data. Since the time difference between the reference and target measurements is very small, very little drift can occur within this short period. This procedure minimizes the variations due to system drift.

2.2 Inverse Synthetic Aperture Radar Imaging

The traditional method of producing radar images from a fixed radar system is known as inverse synthetic aperture radar (ISAR) imaging [5]. In this method, the target is rotated through 360 degrees (for a fully focussed image) and frequency domain response data is recorded at small incremental angles. A two dimensional inverse Fourier transform is then applied to this data to yield a two dimensional image matrix, with the elements of the matrix representing the relative response at a particular down range and cross range location. Since the radar signal travels at the speed of light, the absolute down range and cross range location can be inferred from this data.

One of the severe disadvantages of this method is the amount of data required to produce the image and, hence, the computational resources needed. Typically, a fully focussed ISAR image requires approximately 30-60 minutes of CPU time on the Digital Equipment Corporation VAX 8550 at the ElectroScience Laboratory. Clearly, it is not possible to apply this method to a high speed radar imaging system, although this method does produce a high quality image. Figure 6 is an ISAR image of a tilted square plate, from data scanned over 90 degrees. The responses outside the obvious boundaries of the plate are due to interaction terms, which are caused by energy coupling to the surface of the plate and diffracting off the edges and corners. Note that the image processing is based on direct scatter back to the radar which means that mechanisms that propagate on the surface of the target are not imaged properly. Thus, these terms don't appear on the extent of the target.

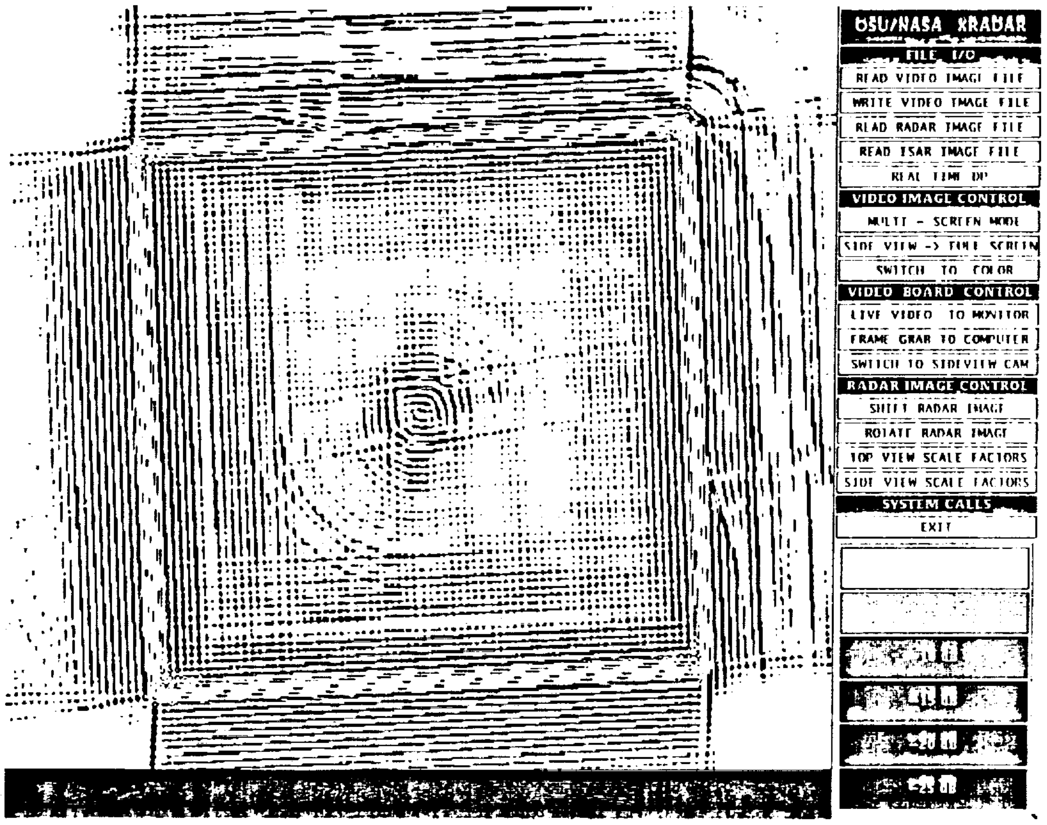


Figure 6: ISAR image of a square aluminum plate.

ORIGINAL PAGE IS
OF POOR QUALITY

2.3 Scattering Center Imaging

While the traditional ISAR method described above assumes that either the target or the radar can be rotated to gather data at the required look-angles, some targets cannot be rotated due to size or fragility. For common RCS measurements, the usual solution is to laterally defocus the feed to obtain backscatter measurements over a relatively small range of look-angles. However, this assumes that the phase front of the incident wave remains planar as the feed is laterally displaced and this is only valid for small displacements. The phase front produced by lateral displacement of the feed is actually astigmatic or nonspherical and cannot be used for imaging, where absolute phase accuracy is required. Recently however, a technique for focussing this astigmatism has been developed at the ElectroScience Laboratory [4]. Further, this technique is applicable to both near- and far-zone measurements.

This technique makes use of the fact that the target area of the compact radar range is well defined. If an arbitrary down-range image plane is defined in the target area, then there will be a well defined distance between a test point in the image plane to each feed element or feed location. Subsequently, the phase of the backscattered field received at each feed element from a scatterer located at the test point can be calculated. If the measured backscattered field is multiplied by the complex conjugate of the calculated phase and summed over all the feed elements, then the backscattered field from points in the image plane where scatterers exist will sum coherently, and incoherently for all other points. Consequently, a cross-range image of the scattering centers will be obtained. The down-range location of a scattering center is found either by moving the image plane or by a frequency scan procedure, as outlined in [4]. Figure 7 shows a radar image, displayed as a surface plot, of three spheres generated using this method.

The main advantage of this technique is that the target does not have to be

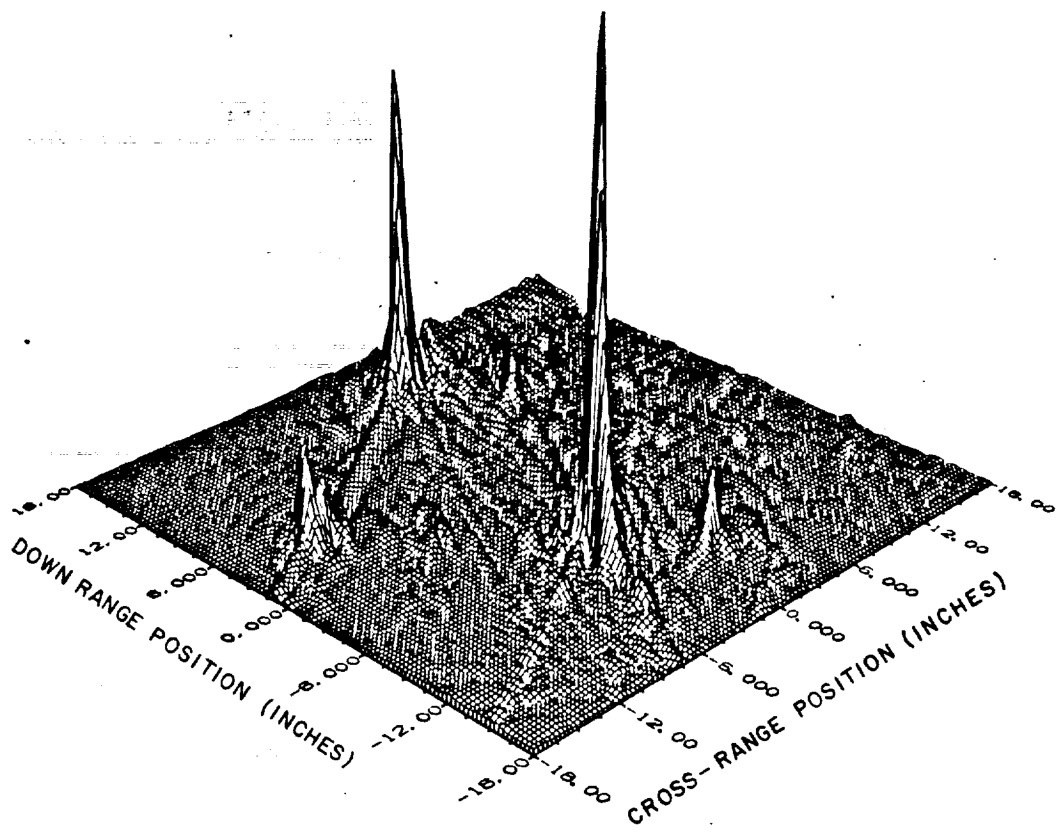


Figure 7: Scattering center image of three spheres.

rotated to obtain the scattering center image, thereby making this process very fast, and therefore it is well suited to producing images in real time.

2.4 Time Domain Response Tracking

Another method of creating radar images is known as Time Domain Response Tracking. This approach requires a minimum of two swept frequency scans at different look angles or different feed positions. Each swept frequency scan is the backscatter response of the target which is transformed to the time domain to

ORIGINAL PAGE IS
OF POOR QUALITY

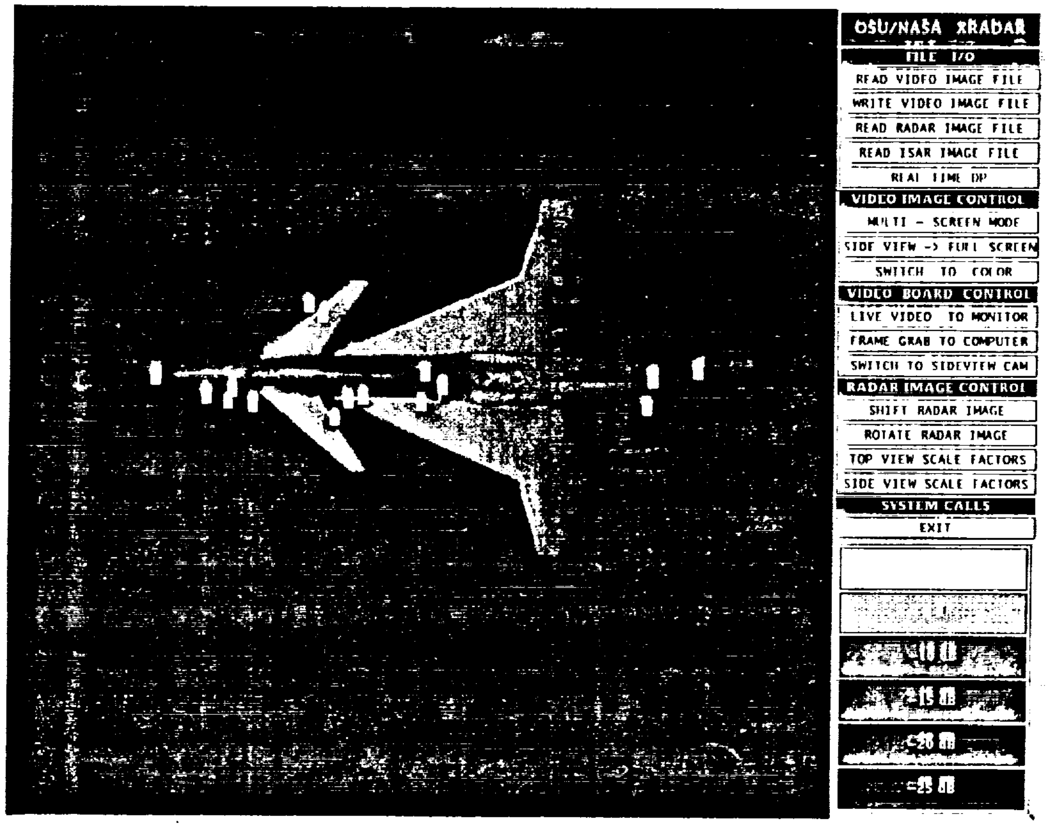


Figure 8: Time-domain tracking image of a aircraft test body.

find the down range location of the scattering centers. The cross range location of the scattering centers is obtained from the change in position of the time domain scattering features between successive look angles. Calculating the impulse response from the frequency scans requires only a one dimensional Fourier transform. The white rectangles in Figure 8 represent the radar image of a aircraft test body generated using this technique.

Obviously, since a minimum of two one dimensional transforms are required, this method is extremely fast, and it has the advantage that the relative amplitudes between the various scattering centers are well known. The disadvantage to this method is that there are substantial ambiguities in the tracking of the time domain scattering features. However, recent work with this method has improved by adding tracking algorithms used to correlate the scattering features between successive look angles.

2.5 Image Presentation

The radar image display system described in this report is capable of displaying radar images generated by any of these methods, but since the system was designed to display the images in real time, the high speed imaging techniques are of more interest.

Chapter 3

System Hardware

3.1 Introduction

The radar image display system described in this report was envisioned to operate in real time, in the sense that the imaging system could operate concurrently with the data collection and processing system. Further, the imaging system needs the capability to capture and display video images together with color presentations of scattering center images. In addition to these basic requirements, it was felt that the ability to transfer the video/radar image to video tape would be invaluable, particularly to visiting users of the facility.

The choices involved in selecting the system hardware were made based both on availability and functionality. The system was originally implemented on two IBM PC/AT's and a DEC VAX 8550, as outlined in Figure 9. This original system was tested for functionality and ease of use for several weeks. While this evaluation confirmed the usefulness of the concept, the speed limitations inherent with that approach didn't make it as attractive as necessary for day-to-day operations. It was found that the primary speed limitation was the necessity to send raw data to the VAX and then send processed data back to the PC used to display the images. The software needed in the PC was too slow to maintain the desired image rate. Writing new software for the image display PC was investigated, but it was decided that even this would not solve the problem. At this point, it was obvious

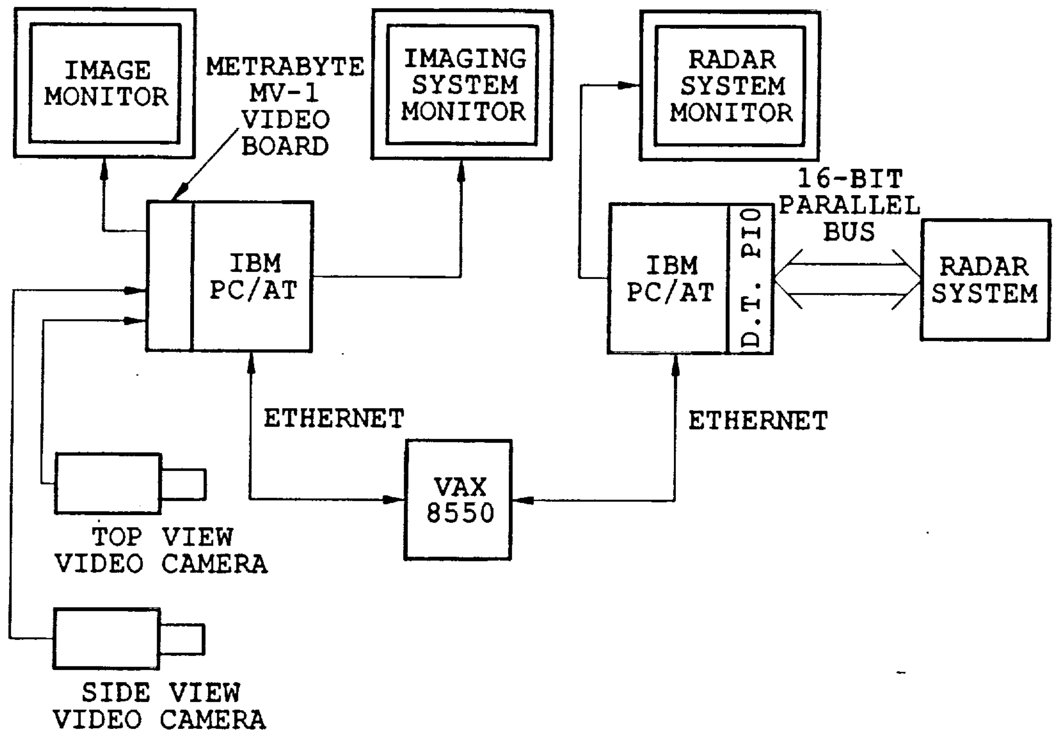


Figure 9: Block diagram of PC/AT - based imaging system.

that a whole different system would be necessary. In fact, it was decided that the processing and image display had to be done on the same computer. Several options, such as adding an array processor to the PC, were explored, but these were also too slow. The Tektronix XD88-30 Graphics Workstation was chosen due to it's extremely high speed, it's ability to process and display high resolution graphics and because it is a multi-tasking computer, which is capable of running more than one process at a time. The other components of the system were chosen to be compatible with the XD88-30. The system hardware consists of the Tektronix XD88-30 Graphics Workstation, an Imaging Technologies FG100V Video Processing board, a Matrix Corporation VME Parallel I/O board,two NEC Corporation NC-8 Color Video Cameras, an IBM PC/AT, and a Data Translation Parallel I/O board. The interconnections between the various components are shown in Figure 10.

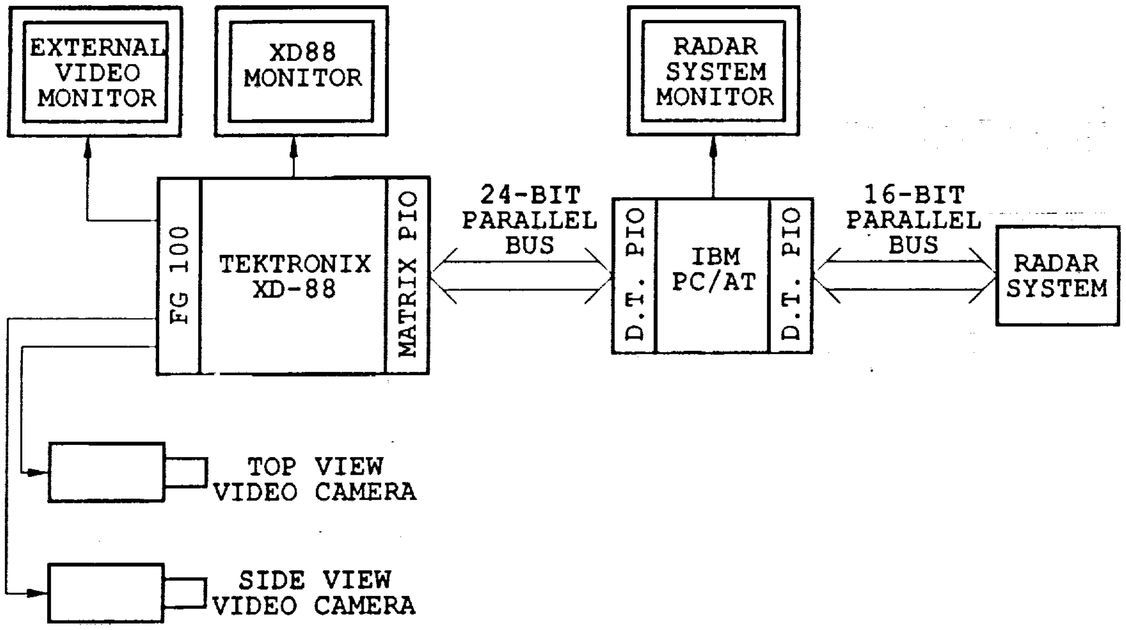


Figure 10: Block diagram of XD88 - based imaging system.

3.2 Tektronix XD88-30 Workstation

The Tektronix XD88-30 is a new super-mini graphics workstation, designed to handle both graphics processing/display and data processing. It runs under the UTEK System V operating system (a variation of the UNIX operating system). The normal user interface which is used in this system is X-Windows, which is described in detail in a later chapter. The XD88-30 used in this system has 16M bytes of main memory, a 160M byte hard disk and a high speed streaming tape drive. The system display is a 19 inch (diagonal) full color RGB monitor with 1280x1024 pixel resolution.

3.3 Imaging Technology FG100V Video Processor

The Imaging Technology FG100V Video Processor is a VME-bus compatible circuit board installed in the XD88-30. It functions as a video frame grabber which digitizes a frame from a video camera or other video source and stores this digitized video image in its own on-board memory. This data is then available to the host computer for processing and display, or the image can be displayed on a separate video monitor connected directly to the board. The FG100V is capable of storing up to four separate frames of video, which are captured from any one of four video inputs. Each frame of video is digitized to a resolution of 512x480 pixels (from an NTSC standard composite video signal). In addition, the FG100V has provisions for many specialized image processing techniques, although in this application these capabilities are not used.

3.4 NEC Corporation NC-8 Video Camera

The NEC NC-8 Video Camera is a color video camera which is suited to the low light-level conditions found in the compact range at The Ohio State University's ElectroScience Laboratory. Two of these cameras are used : one mounted on an overhead crane to provide a top view of the target (showing the target in the X-Y plane) and another is mounted on the wall of the compact range room to provide a side view of the target (showing the target in the Y-Z plane).

3.5 Matrix VME Parallel I/O Board

The Matrix VME Parallel I/O board is a VME-bus compatible 32-bit digital I/O board installed in the XD88-30. It, along with the companion board installed in the IBM PC/AT, serves as a high speed parallel communication bus between the XD88-30 and the IBM PC/AT.

3.6 IBM PC/AT

The IBM PC/AT controls the compact radar range hardware and handles data collection [6]. Raw data is transferred to the XD88-30 on the 32-bit parallel I/O bus, where all data processing, display, and data storage is done. The IBM PC/AT unit used in this system is configured with 640K bytes of main memory and 2M bytes of extended meory. The system monitor is an EGA-compatible RGB display.

3.7 Data Translation Parallel I/O Board

The Data Translation Parallel I/O Board is a circuit board installed in the IBM-PC/AT. It is the companion board to the Matrix Parallel I/O board installed in the XD88-30 and is used to transfer data between the IBM PC/AT and the XD88-30.

Chapter 4

X-Windows Overview

The X-Windows user interface was chosen for this system because it provides a programming environment specifically designed to allow graphical user interfaces to exist in a multi-user network environment. Although X-Windows provides a very extensive set of basic functions for creating and manipulating graphics, these functions tend to be too basic for creating large applications [9]. Therefore, most X-Window programming packages also include a set of graphics objects known as 'widgets' and a set of functions for creating and manipulating these widgets. For this project, the widget set created at MIT known as the Athena widget set [Xaw] and the widget toolbox known as Xt toolbox [Xt] was used [7,8].

4.1 Widget Programming

Since X-Windows is a hierarchical system, all widgets are said to be either a 'toplevel' widget or 'children' of some other widget. Below a toplevel widget, all the children are organized in a tree structure such that an operation performed on a parent (such as deleting the parent) usually affects all the children below it in the tree structure. Within a program, a widget is actually just a specific instance of a dynamically allocated data structure which holds all the information necessary to describe the widget. In addition to setting the parameters which describe the appearance and the hierarchy of widgets, the major task of the programmer is to describe what action is to be taken in response to the various actions of the user.

The primary benefit of a software environment, such as Xaw and Xt, is the consistency of the interface between the programmer and the system. All widgets have a common set of basic parameters. Identification of the parent widget, location of the widget within the parent widget, foreground and background colors, as well as others are specified for all widgets. In addition, parameters specific to each type of widget are included. Further, these parameters can be set either within the program, at run-time from a data file or from a command line. The details of widget appearance and programming are best illustrated with several examples.

4.1.1 Example : The Command Button Widget

The Command Button Widget is a rectangle that contains a text label. When the pointing device[1] cursor is placed within the rectangle, its border is highlighted to indicate that the button is available for selection. When the pointing device button is clicked, the command button widget is selected and the callback routine associated with the widget is executed. When creating a command button widget, numerous resources are available to the programmer to control the appearance and behavior of the widget. Table 1 lists the available resources, the default value and a brief description of what each resource does.

The menu of X-Radar is made up of command button widgets, each of whose callback routines accomplishes a particular task. These callback routines can be very simple, such as the 'EXIT' button, which simply exits the program, or very complex, as is the callback for the 'RTDP IMAGING' button, which itself contains a separate event processing loop. Further, the resources associated with a particular widget can be changed dynamically during program execution. For example, the text labels for some of the buttons in the menu of X-Radar are changed dynamically to reflect the possible menu choices the user has available at a given moment.

[1]A pointing device is a mouse or trackball

Table 1: Command button widget resources and their default values.

Name	*Default Value*	*Description*
XtNbackground	White	Window background color
XtNbackgroundPixmap	none	Window background pixmap
XtNborderColor	Black	Window border color
XtNborderPixmap	none	Window border pixmap pattern
XtNborderWidth	1	Width of button border in pixels
XtNcallback	NULL	Callback for button select
XtNcursor	opendot	Pointer cursor style in widget
XtNdestroyCallback	NULL	Callback for *XtDestroyWidget*
XtNfont	fixed	Label font
XtNforeground	Black	Foreground color
XtNheight	text height	Button height in pixels
XtNhighlightThickness	2	Width of highlighted border in pixels
XtNinsensitiveBorder	Gray	Border color when not sensitive
XtNinternalHeight	2	Internal border height for highlight
XtNinternalWidth	2	Internal border width for highlight
XtNjustify	XtJustifyCenter	Type of text alignment
XtNlabel	Button name	Button label
XtNmappedWhenManaged	True	Whether *XtMapWidget* is automatic
XtNsensitive	True	Whether widget receives input
XtNtranslations	none	event-to-action translations
XtNwidth	text width	Button width in pixels
XtNx	0	Widget x coordinate in pixels
XtNy	0	Widget y coordinate in pixels

Table 2: Box widget resources and their default values.

Name	*Default Value*	*Description*
XtNbackground	White	Window background color
XtNbackgroundPixmap	none	Window background pixmap
XtNborderColor	Black	Window border color
XtNborderPixmap	none	Window border pixmap pattern
XtNborderWidth	1	Width of button border in pixels
XtNdestroyCallback	NULL	Callback for *XtDestroyWidget*
XtNhSpace	4	Horizontal space between children
XtNheight	text height	Button height in pixels
XtNmappedWhenManaged	True	Whether *XtMapWidget* is automatic
XtNtranslations	none	event-to-action translations
XtNvSpace	4	Vertical space between children
XtNwidth	text width	Button width in pixels
XtNx	0	Widget x coordinate in pixels
XtNy	0	Widget y coordinate in pixels

This capability allows the same command button to accomplish several different tasks, thereby saving system resources and speeding up program execution.

4.1.2 Example : The Box Widget

The box widget provides an environment which manages the placement of other arbitrary widgets within a box of specified dimensions. The children are rearranged to best fit within the box when the box is resized or when children are added or deleted. Since the placement of children within the box is automatic, the programmer has little control over the arrangement of the children. Table 2 lists the available resources and their description.

The menu area for X-Radar is contained in a box widget, although some control over placement of the command button widgets and label widgets is achieved by carefully sizing them so that they only fit within the box in a vertical arrangement. The advantage of using a box widget is that the programmer need not calculate the exact screen coordinates for each child widget, as would need to be done if a menu were constructed without the box. Further, if the menu needs to be suppressed at some point during program execution, only the box widget needs to be 'unmapped', which will cause all of its children to be unmapped as well.

There are several methods for actually creating a widget with the desired resource values, and the interested reader is referred to Appendix B for the method used for this system or to [10] for other methods.

4.2 Event Model Programming

Menu-style user interfaces are said to be event-driven which means that the program appears to be doing nothing until the user generates an 'event'. Events are usually generated by moving and/or clicking the pointing device, typing a key on the keyboard or by timers created within the program. X-Windows also provides the tools to describe and accept events from any other device which might be attached to the computer system.

Each particular type of widget provides its own mechanism for accepting events. Part of a widget's description are parameters which associate particular events with the widget and what action is to be taken following those particular events. It is the job of the programmer to create logical and useful action sequences for each particular event, such as what to do after the user clicks a mouse button on a 'button' widget.

4.3 Programming Style

Due to the very nature of the X-Windows environment, there is not a lot of room for individual programming style. The hierarchical structure of the environment dictates the basic order of the program, although the appearance of the application is completely up to the programmer. Initially, the hardware environment must be established through calls to functions which return the type of terminal or screen on which the application is being run. Then, for most types of applications, the widgets which will be needed for the application are created. Since widgets can be created without being displayed, it is usually advantageous to create all necessary widgets at the beginning of the program, rather than creating them as needed within the application. In addition to creating the widgets, the functions which are called in response to each particular event must be written. At this point, all that remains is the event processing loop. Depending on the complexity of the events and the way in which events are to be processed, the programmer can rely on a loop procedure included in the Xt toolbox or write their own event processing loop. Usually, the application ends from within the processing loop in response to some particular event.

Chapter 5

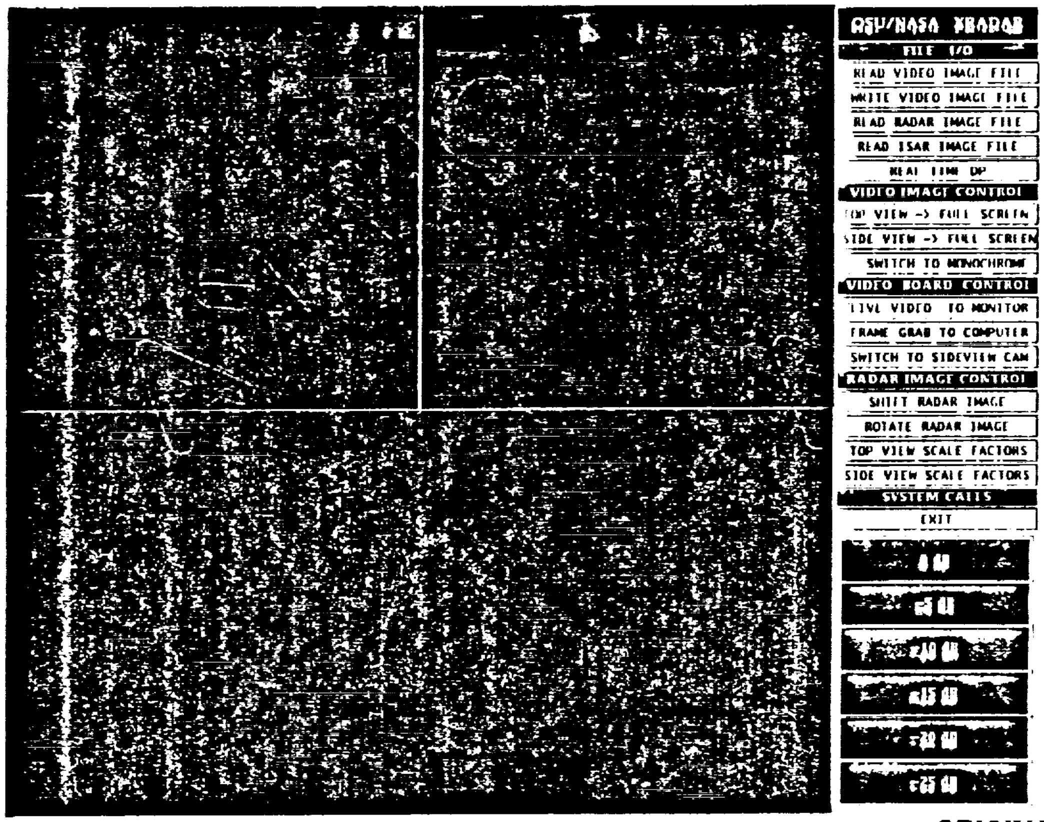
Software Design Philosophy

There were several major objectives in designing this system. First, to provide an easy to use environment to capture and display top and side views of video and radar images; second, to provide the user with a convenient method of storing and retrieving this information; and third, to provide an analytic tool for compact radar range data acquisition and analysis. These goals indicated that a menu-driven program was needed, as well as dictating that a large portion of the screen be dedicated to various image displays. In accordance with the program naming convention associated with X-Windows, this program has been named X-Radar.

As discussed in Chapter 4, X-Windows dictates, to a large part, the overall design of the software, but the appearance and ease of use of the system is programmer dependent. For this application, most of the individual users of this system will not use the system very often; thus, the software interface must be as intuitive as possible. As a result, the menu item labels must be simple and straightforward, and the error recovery procedures should be consistent and as graceful as possible. Finally, user prompts must be used liberally to guide the infrequent user.

5.1 Display Screen Partitioning

The resolution of the video acquisition board is 512x480 pixels, and the desired elements of the main display window dictated the display screen in terms of partitioning this data as well as others. Figure 11 shows one example of the partitioning of the main display window. The main menu is displayed along the right side of the display. All functional selections are listed in this menu.



ORIGINAL PAGE IS
OF POOR QUALITY

Figure 11: X-RADAR main window with sub-window partitions.

Two 512x480 windows across the top of the display are normally used to display the top and side views of the radar target. These are the main image display windows, although either one can be expanded to fill the entire display screen (excluding the main menu area).

The remaining 544x1024 window at the bottom is used to display system messages and data plots from the range hardware when this program is used in conjunction with the data processing code.

5.2 Main Menu

The main menu is divided into four parts, with related functions grouped together as shown in Figure 11. The top-most section has functions related to image file input and output. Next is a group of functions controlling how the video images are displayed; i.e., full-screen or multi-window. Following this section is a group of functions for controlling the FG100V video board. And finally, a group of functions for controlling how the radar image is displayed.

5.3 Pop-up Data Input Windows

When user input is required, a pop-up window is used. If only a single piece of data is required, such as a file name, all keyboard input is 'focused' or forced into that window. When input is complete, the user only needs to close the window by clicking the mouse on the 'EXIT' button. If more than one piece of data is requested, the user must click the mouse on the item desired and enter the requested information. For windows where input is essential, the user is blocked from exiting the window until all input fields are entered.

5.4 Error Recovery

Error recovery has been designed to be as painless for the user as possible. When a recoverable error occurs, a text window containing a descriptive message is 'popped up' on the screen. In order to assure that the user acknowledges the error, the message window is actually a single item menu, and the user must click the mouse on it in order to erase the message window. Several other approaches to this were tried, but this was found to be the most effective way to handle errors that must be acknowledged.

Recoverable run-time errors that the system is likely to encounter are limited to two types. First, file access errors are operating system generated, and are usually due to a specified file not being found or being otherwise unavailable. If this type of error occurs, an appropriate message is displayed, and the corresponding input window is re-displayed after the error is acknowledged. The second type of recoverable error is due to the user not using the system correctly. These errors are termed 'X-Radar Protocol Errors' and usually occur when the user does something which is inappropriate in the current system context. Appendix C lists all of the recoverable errors and their possible causes.

Other system errors can occur, but usually require more user attention than can be provided from within the application. If this type of error occurs, the program terminates, and the user is returned to the operating system.

5.5 Interfacing to the Real Time Data Processing Program

This software was designed to be used in a multi-tasking environment and can be run in conjunction with the real-time data processing program called RTDP, which is run as a background process to this program and is described in [14]. Data is passed from the RTDP program to X-Radar using an X Windows mechanism known as 'window properties' [9].

A window property is a programmer-defined data structure associated with a particular window, and as long as two applications share a common window, a pointer to the property data structure may be passed between the applications. The simplest window to use for this is the 'root' window, which is shared by all applications running on the workstation. Further, 'PropertyNotifyEvents' can be passed between applications to notify each application that another application has modified the shared property. X Windows prevents applications from accessing the property simultaneously by storing access requests in a queue. This prevents data corruption between applications which are writing and reading the property (this does not prevent applications from overwriting old data before it is read however).

For this software, the property used to share data is a floating point array, large enough to hold a complete radar image. In the RTDP IMAGING mode, X-Radar enters an event processing loop which handles only two kinds of events : PropertyNotifyEvents and ButtonPressEvents. When the RTDP program has completed an image, it sends X-Radar a PropertyNotifyEvent. X-Radar then retrieves the data in the array and displays it. When the display is complete, X-Radar generates a PropertyNotifyEvent, telling the RTDP program that it is ready for the next image. This full handshaking between the two applications guarantees that X-Radar does not miss any data transfers. This sequence is repeated until a ButtonPressEvent (generated by pushing a pointing device button) is encountered, which causes X-RADAR to exit the RTDP IMAGING mode.

Chapter 6

Lighting and Camera Placement for Video Acquisition

Any discussion of photographic lighting must begin with consideration of the subject matter, the environment in which the photography is to be done and the objective in photographing the subject. In addition, there is always an ideal solution and a realistic solution to a given problem, but the realistic solution requires some compromise. And so it is with the problem in this case, which is to photograph radar test targets which are actually mounted in the compact radar range.

The objective in photographing the target is straightforward : to render the object with as much detail as possible, in an uncluttered, simple field of view, so that when it is combined with the radar image of the target, only that information which is pertinent will be visible.

6.1 The Ideal Target and Lighting

It is perhaps most informative to discuss the ideal target and lighting combination first, as a means of illustrating the goals for which one should strive. Some of these goals are easily achieved, while others are more difficult.

Figure 12 illustrates the photographic effect which one would like to achieve. In the photograph, the model hangs in black space, with no extraneous background and with full surface detail easily visible. Also the mount for the model is visible,

ORIGINAL PAGE IS
OF POOR QUALITY

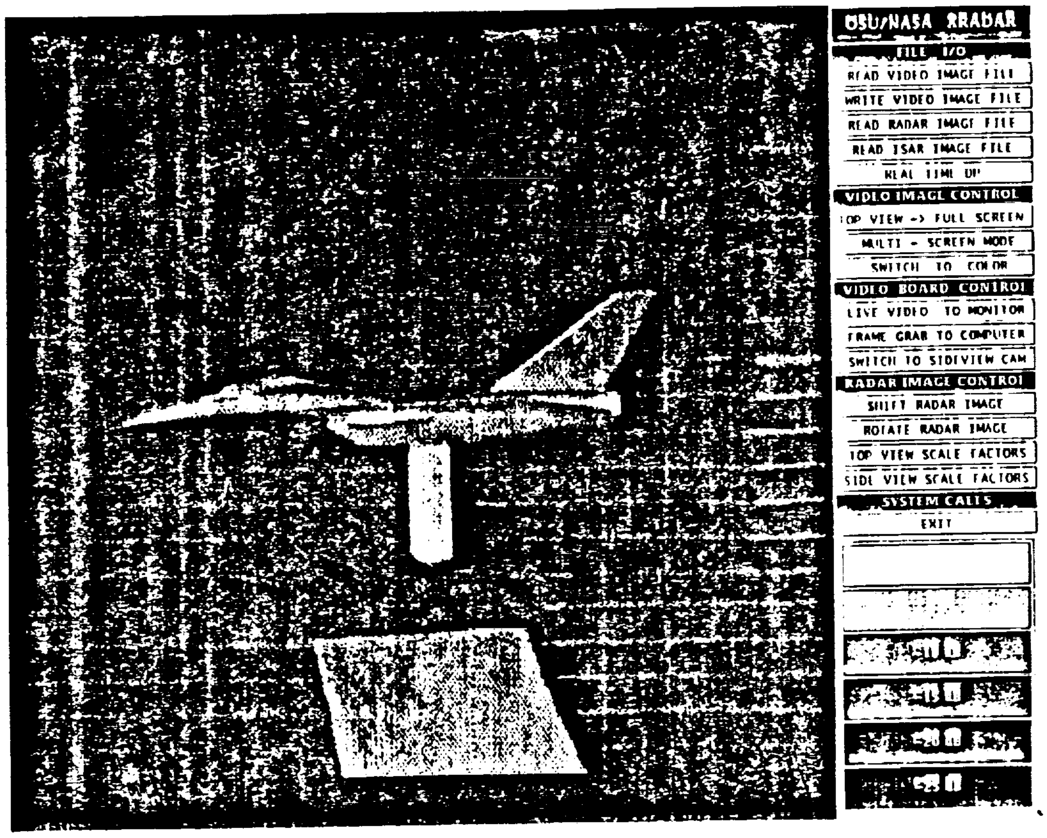


Figure 12: An example of the correct lighting and target.

which is important for documentation.

First, proper lighting is required to render the surface detail correctly. Specular reflections tend to wash out any surface details and also cause the video camera to saturate. The model in this illustration was lit with two photographic flood lights mounted at the same height as the model at a low angle of incidence to the model's fuselage, as shown in Figure 13. Positioning the lights at low angles of incidence and using diffuse lighting are two ways to minimize specular reflections.

Secondly, the background is completely black, which focuses all of the viewer's attention on the model. While this a seemingly simple point, it is surprisingly

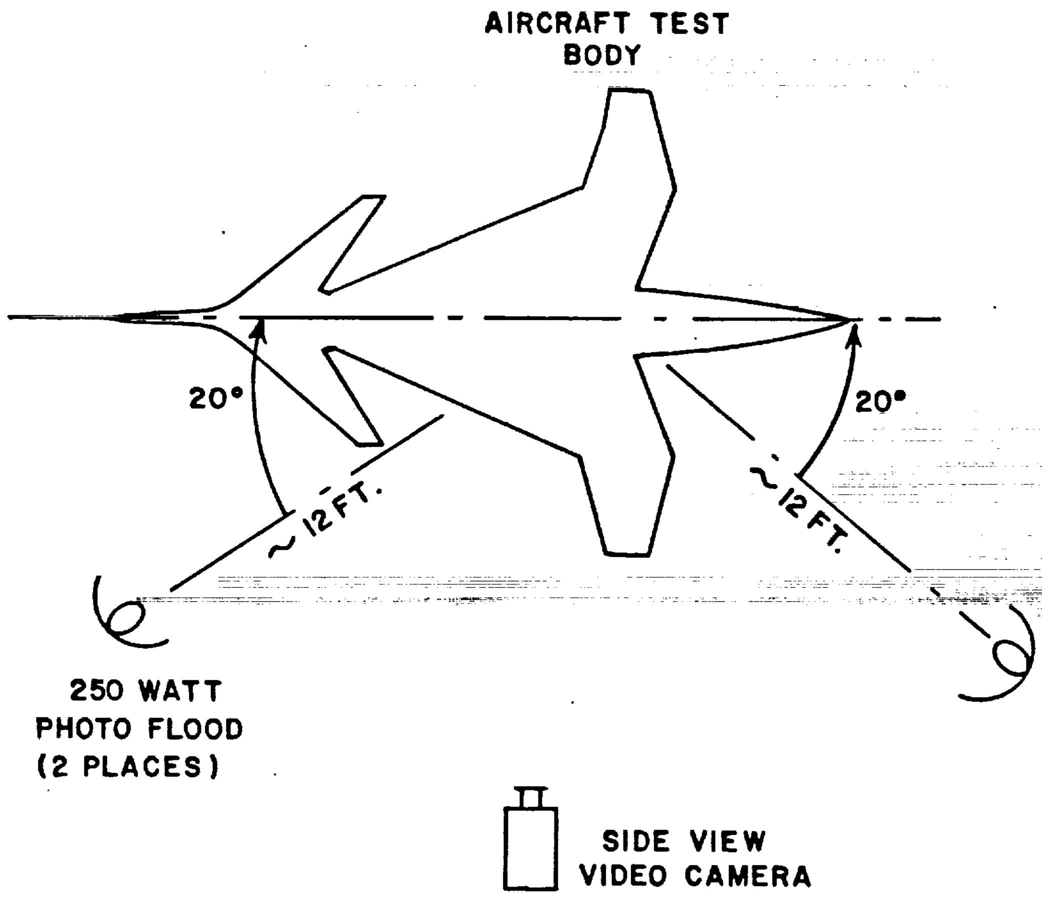


Figure 13: Lighting diagram for previous figure.

difficult to achieve in practice.

These ideas form the goals which one would like to achieve in the compact range. Again, some of these goals are difficult to achieve, but there are many ways to overcome the problems.

6.2 Lighting for the Compact Range at the ElectroScience Laboratory

While the compact range is designed to do radar imaging, it was not designed to be a photographic studio. The walls, floor and ceiling are covered with dark blue or black radar absorbent material and lighting fixtures are held to a minimum in order to reduce metallic objects within the room, which might cause unwanted scattering of the radar signal energy.

The ambient lighting in the compact range room is provided by four sodium vapor fixtures in the ceiling. These lights are very bright and are not at all consistent with the goal of diffuse lighting for the target. The specular reflections these lights cause are very strong, particularly on polished targets. One solution would be to turn these lights off during video acquisition, but due the long start-up time of these lights (approximately 30 minutes), this is not a viable solution. Unfortunately, there does not appear to be a satisfactory solution to this problem other than to alter the surface finish of the targets.

Most of the radar targets being investigated are made of polished aluminum, copper or silver-plated metal. Under ideal studio conditions, these types of objects are difficult to photograph, but under the conditions presented in the compact range room, it is a very difficult task. Several possible surface treatments which would ease the lighting problem have been considered. Probably the simplest and easiest solution is to apply a thin coating of matt paint to the target. The impact on the electromagnetic properties of such a coating have been investigated quite extensively, but the results are fairly simple : if the coating is thin, in terms of

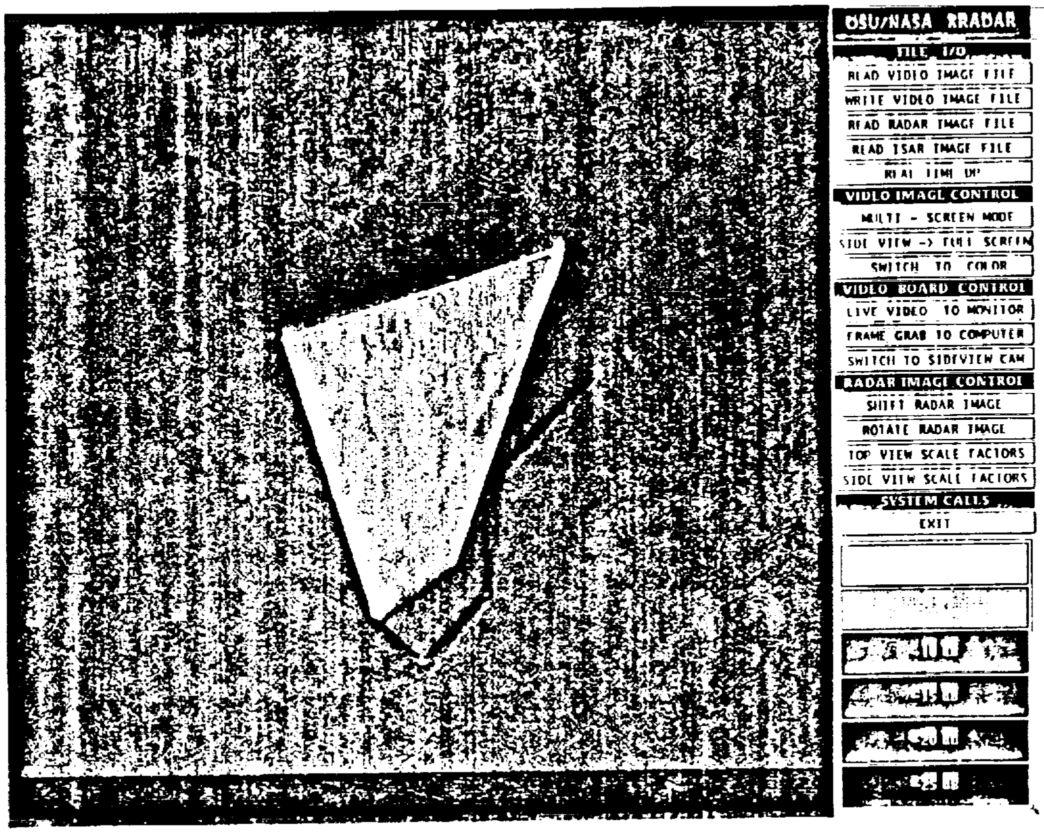


Figure 14: An unpainted aluminum test target.

wavelengths, and the target is a perfect conductor, the scattering properties will not be seriously affected. For most targets of interest in this project, the target material can be considered a perfect conductor, so a thin coating of paint should not present too much of a problem. Figures 14 and 15 show the advantages of painting a target white. In these figures, the exposure has been adjusted to show the target to best advantage.

A further problem caused by the ambient lighting in the compact range room concerns the photographic appearance of the backgrounds surrounding the target. The overhead camera looks down directly at the floor, which is covered with blue radar absorbing material. The blue paint on the absorber reflects a surprising

ORIGINAL PAGE IS
OF POOR QUALITY

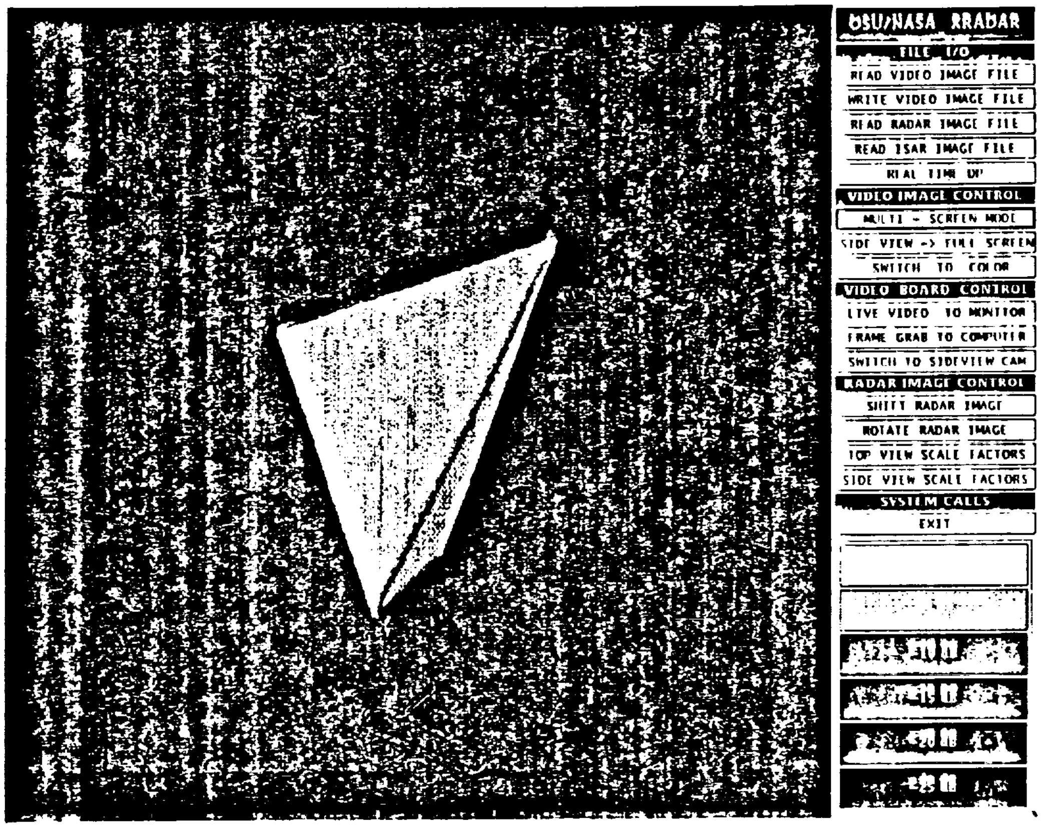


Figure 15: The same target with a light coating of white paint.

amount of light, making the floor appear much brighter in the video image than to the eye. As a result, the target does not stand out significantly from the background. The obvious solution to this problem is to create a darker background on the floor around the target. Figures 16 and 17 illustrate this effect. In Figure 16 the background is black; while in the other figure blue painted absorber was used as the background. At the ElectroScience Laboratory, the absorber on the floor in the vicinity of the target is blue, but it is obvious that the radar absorbing material on the floor in the vicinity of the target should be changed to a material that is natural black. This would provide a very dark background for the overhead camera without affecting the overall lighting of the target or the scattering characteristics of the range.

The ambient lighting does not provide enough light to acquire satisfactory video images, so additional lighting is provided by four 500 watt photographic flood lights. Two of these are located near the focus and aimed at the parabolic reflector of the compact range. Since the reflector is painted white, this provides a very high level of diffuse light in the vicinity of the target. To balance the amount of light on the target, the other two flood lights are aimed at the back wall of the range room. While the material on the rear wall of the room is not extremely reflective, it does provide enough additional diffuse light to light the target zone effectively. In addition, individual targets may require even more light than this, and up to four 250 watt clamp-on photographic flood lights are used for very small or dark targets.

6.3 Camera Placement and Lens Choice

The placement of the cameras was determined by the viewpoints required for the imaging system, one above the target and one to its side. The range is equipped with an overhead crane for moving large targets into and out of the room. This provides an ideal platform for the overhead camera. The crane can be moved

ORIGINAL PAGE IS
OF POOR QUALITY

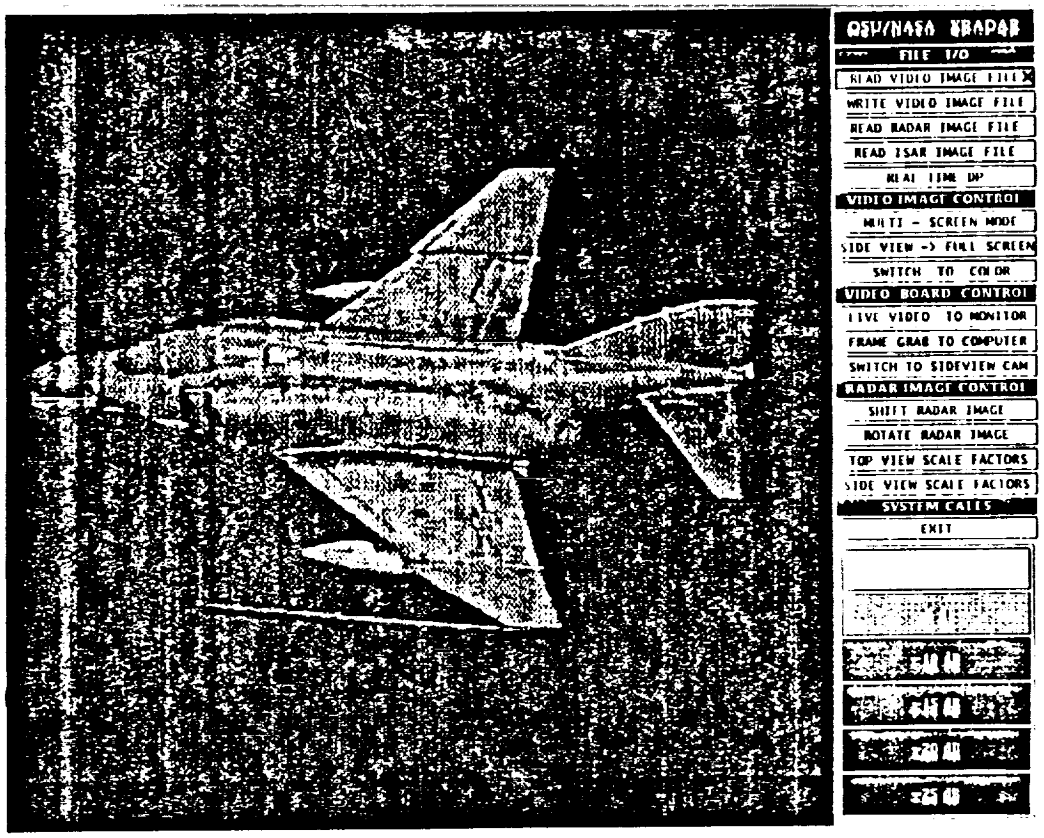


Figure 16: Aircraft model against a black background.

ORIGINAL PAGE IS
OF POOR QUALITY

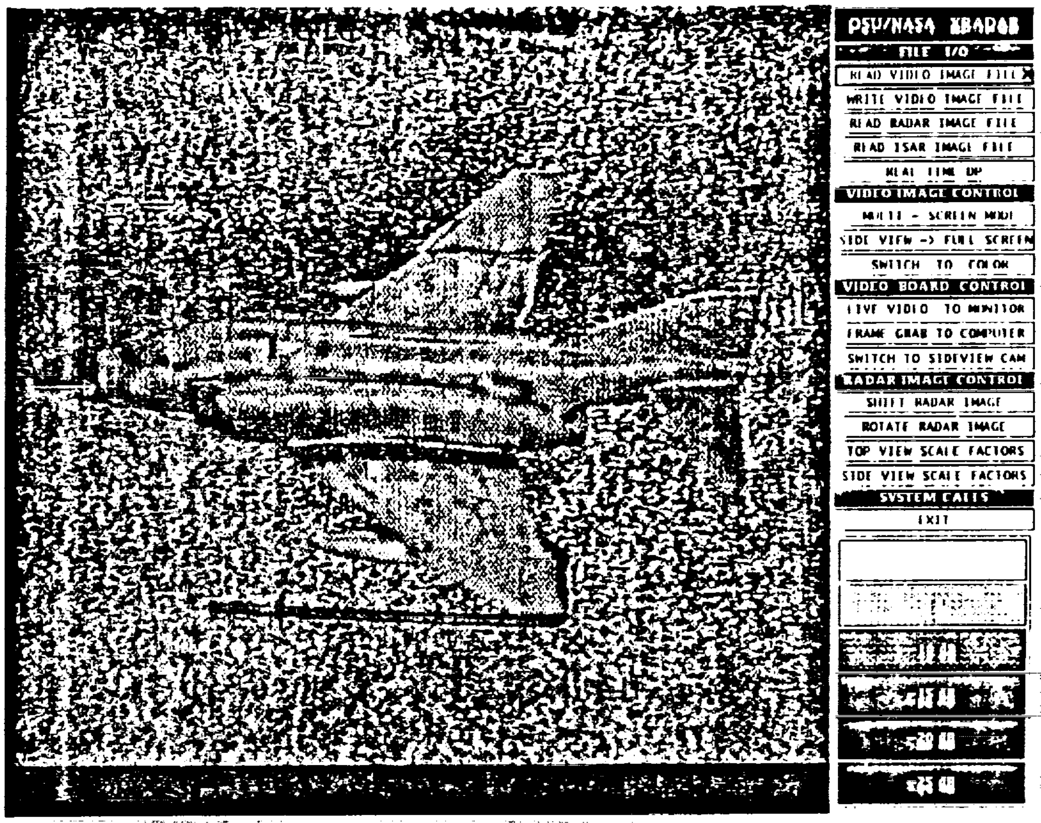


Figure 17: Aircraft test body against a blue absorber background.

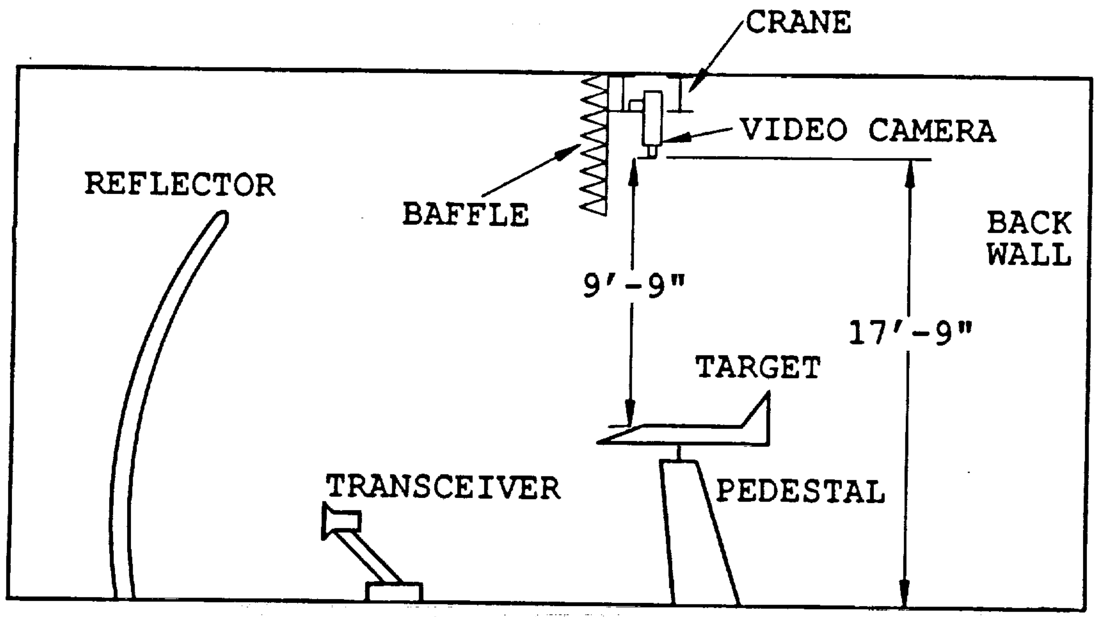


Figure 18: Side-view of compact range showing top-view camera placement.

directly over the mounted target for filming and then moved out of the way while radar measurements are taken.

The lens choice for the overhead camera was determined by the geometry of the room and the maximum target size. Figure 18 shows the geometry of the room, the crane in place over the rotation pedestal and the location of the overhead camera. A 7.5 mm focal length lens is used for most targets. The major problem encountered with this choice is the distortion introduced by such a wide angle lens. Some of the distortion is accounted for in the scaling procedure used for the radar images, but this can only correct linear distortion, not the barrel distortion introduced by a wide angle lens. A better solution would be to correct the radar image based on a distortion map of the lens.

The side view camera is mounted on a removable pedestal near the side wall of the compact range chamber. In other applications, this camera could be mounted almost anywhere in the room, as in most compact range facilities, the target can be rotated to provide a side view. In the Ohio State University compact range, the camera was mounted near a side wall so that the background would be the opposite side wall. Since the walls are covered with dark blue material and are not well lit, this provides a relatively dark background in front of which the targets are readily apparent.

Figure 19 shows a down-range view of the room, the target pedestal and the location of the side view camera. Again, the geometry of the room and the maximum target size determined the focal length of the lens. A 12.5 mm focal length is used for most work. The distortion with this lens is not as noticeable as with the overhead lens due to the increased focal length.

6.4 Lighting and Camera Placement Results

This arrangement of cameras and lights provides very satisfactory video images of a variety of targets. Shown in Figures 20 through 23 are video images taken directly from the imaging system in the compact range. The targets are described in the captions associated with each figure.

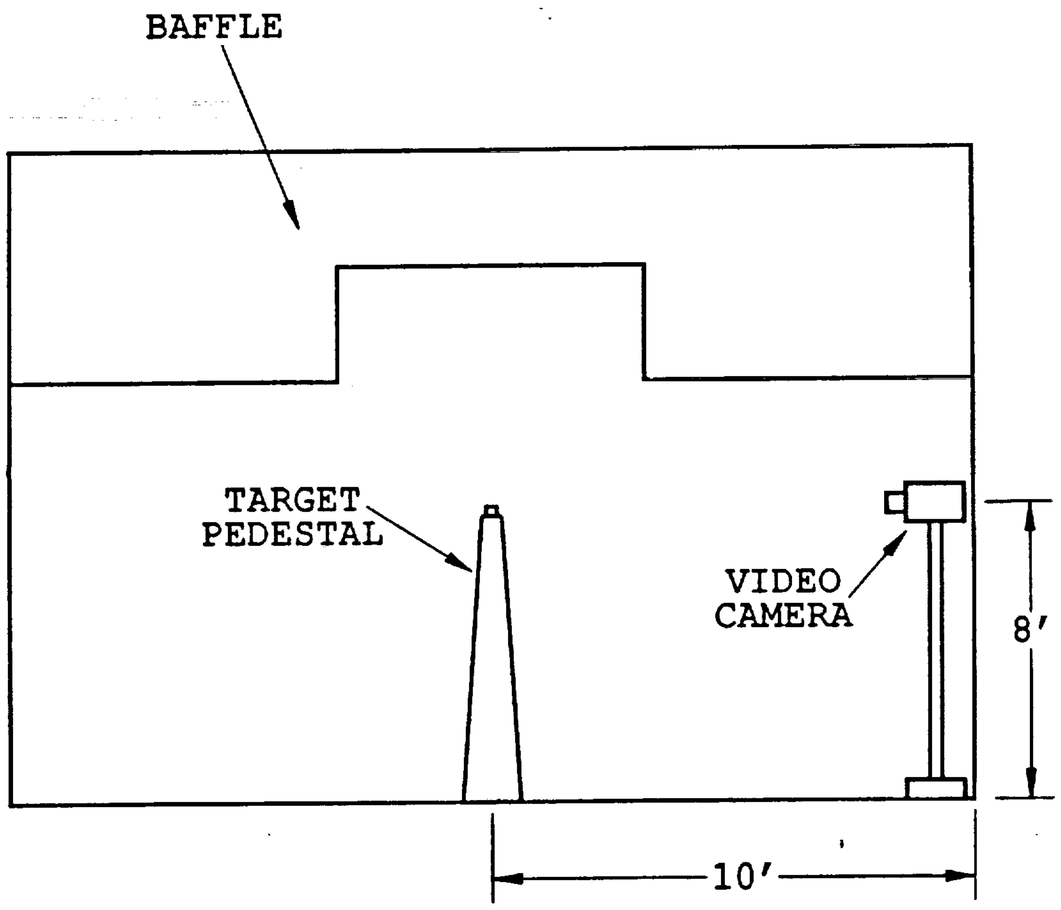


Figure 19: Down-range view of range showing side-view camera placement.

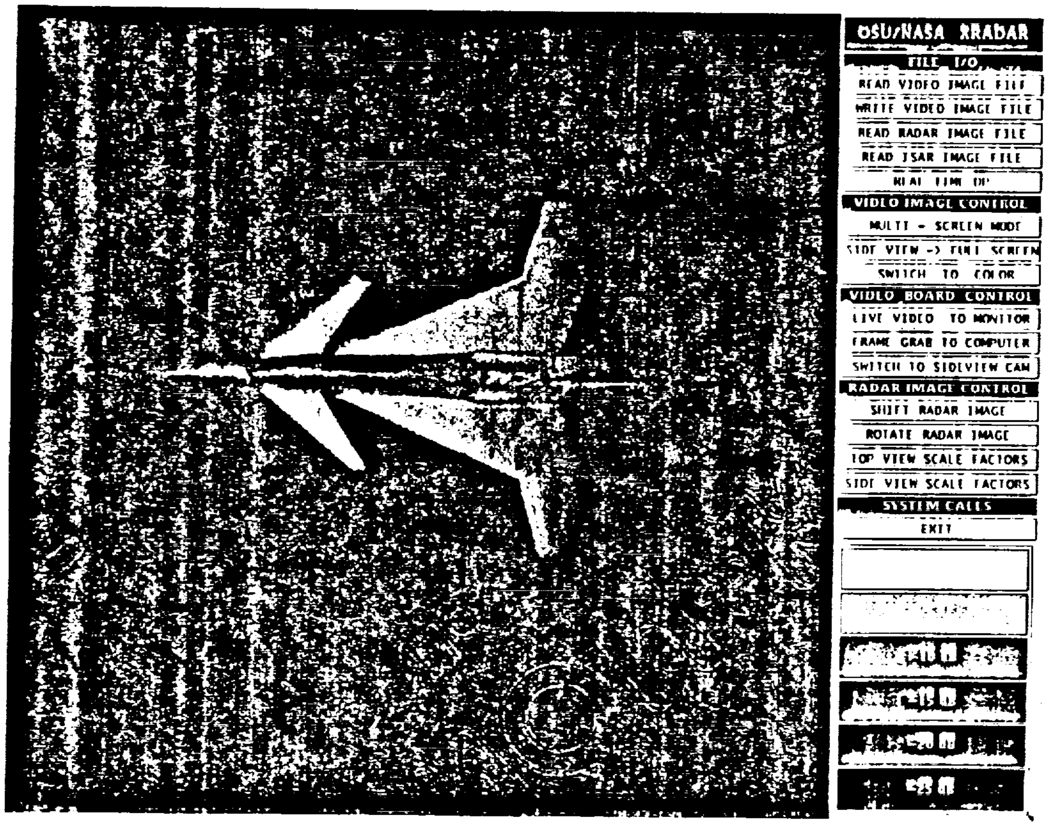


Figure 20: Top view image of an aircraft test body.

ORIGINAL PAGE IS
OF POOR QUALITY

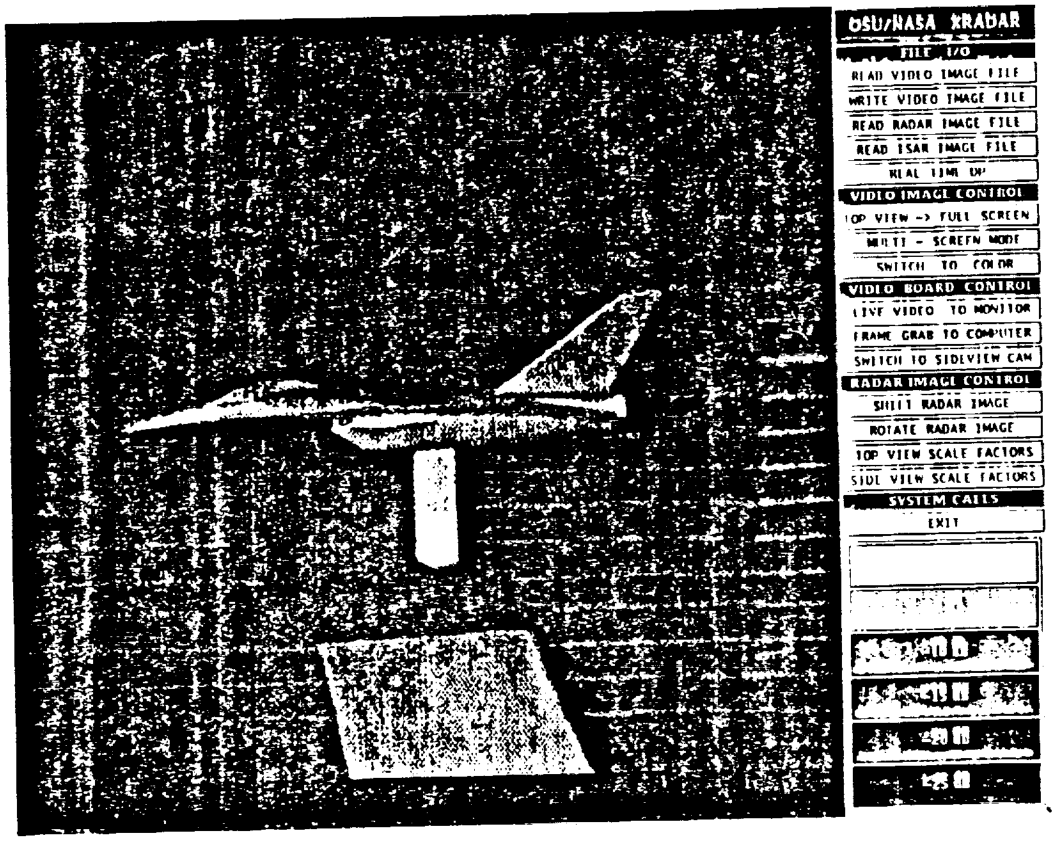


Figure 21: Side view image of an aircraft test body.

ORIGINAL PAGE IS
OF POOR QUALITY

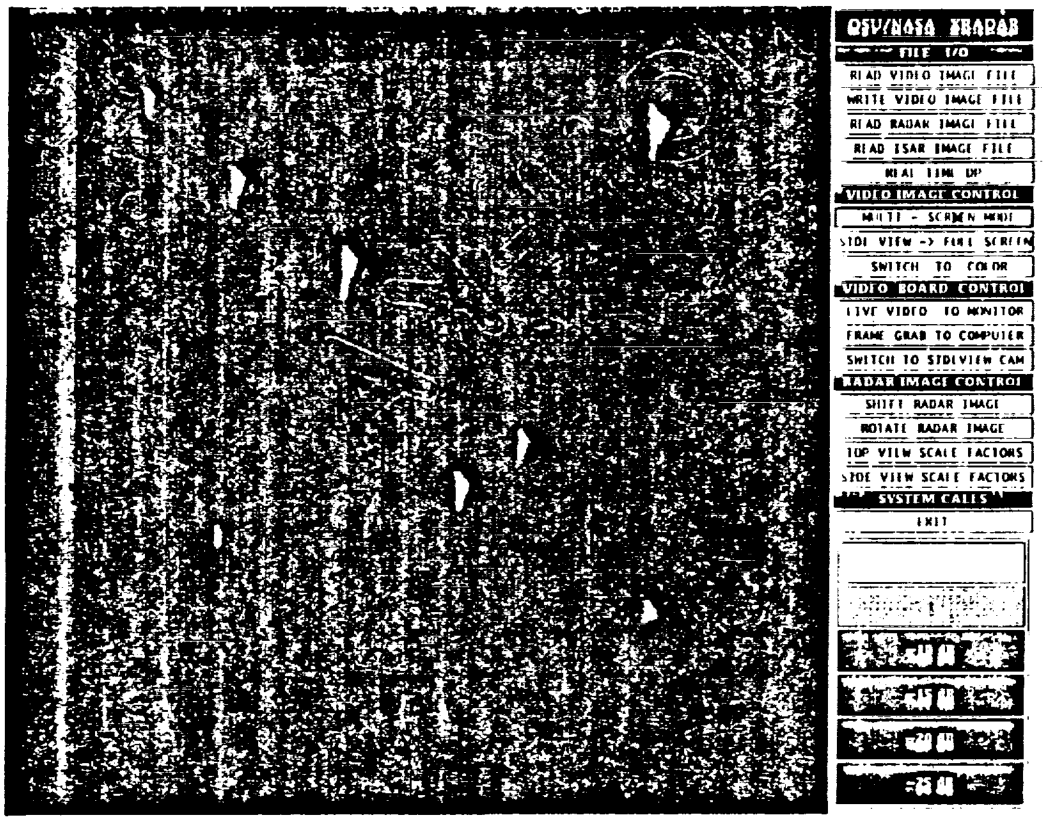


Figure 22: Top view image of dihedral test scatterers mounted on styrofoam.

ORIGINAL PAGE IS
OF POOR QUALITY

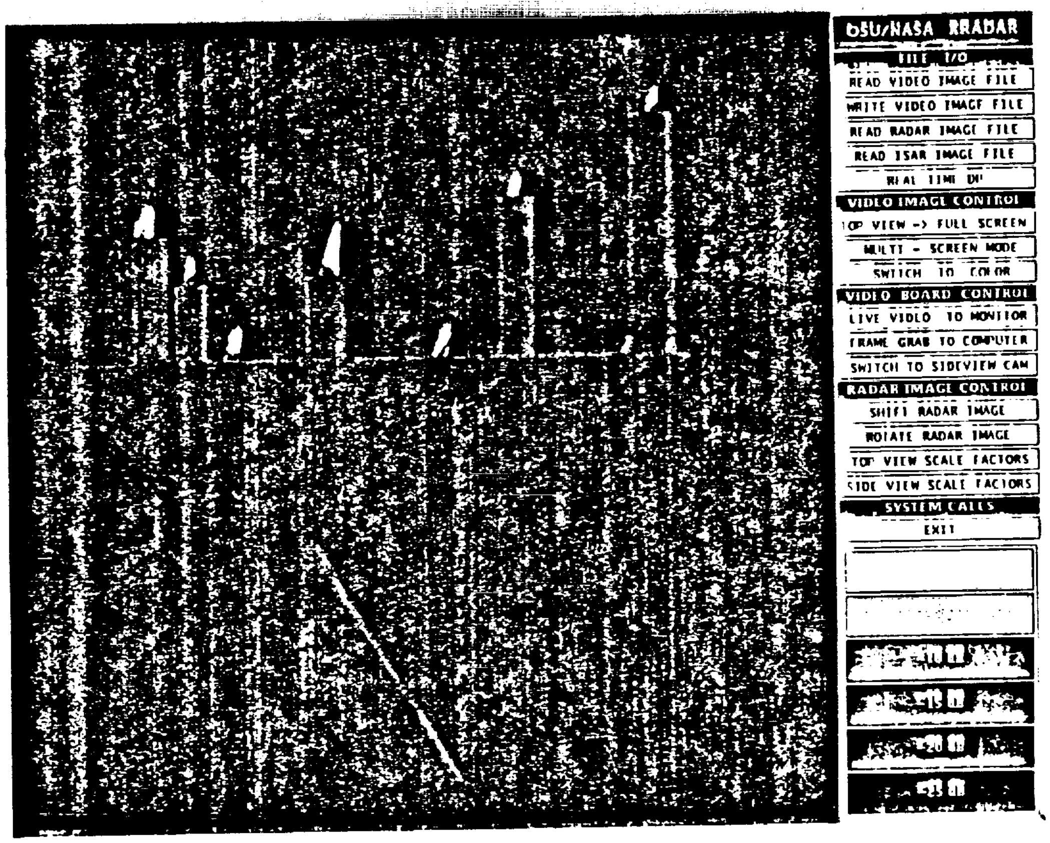


Figure 23: Side view image of dihedral test scatterers mounted on styrofoam.

ORIGINAL PAGE IS
OF POOR QUALITY

Chapter 7

An X-Radar User's Manual

One of the main goals in the design of X-Radar was to make it as user-friendly as possible. While not all problems that the user might encounter can be foreseen, the program is fairly straightforward to use and after a short initial learning period, it's use should be quite simple.

7.1 Initialization of Shared Memory Resources

Before X-Radar can be executed, the shared memory resources for the FG100 must be initialized. This is done by entering super-user mode and executing the program 'frame_init'. If 'frame_init' has been executed since the last system power-down or re-boot, 'frame_init' need not be executed again, although no harm is done by doing so.

7.2 Executing X-Radar

Next, 'xradar' is executed from a command line. It will take several seconds for the main screen and menu to appear. The program is ready to use when the command buttons respond to movement of the pointing device.

7.3 Camera and Lighting Adjustment

Before adjusting the lights and cameras, the 'LIVE VIDEO TO MONITOR' menu item should be selected so that the camera image can be viewed on the external monitor. The video image displayed on the monitor can be switched between the two cameras by selecting the 'SWITCH TO SIDEVIEW CAM' or 'SWITCH TO OVERHEAD CAM' as required. Notice that the same menu item serves both functions.

The cameras and lights are turned on from a central power switch. The lights pointed at the parabolic reflector should be aimed at the approximate center of the reflector, while the lights facing the back wall of the chamber should be aimed towards the center of the back wall.

The overhead camera (on the overhead crane) should be moved out over the target so that the back edge of the overhead baffle is in line with the white marks at the top of the chamber walls. The camera should be aimed so that the center of the video screen is coincident with the axis of the target pedestal. The side view camera and pedestal should be fitted to the mating receptical mounted at the base of the side wall of the chamber, and the camera aimed so that the vertical axis is aligned with the target pedestal axis and the horizontal axis bisects the target vertically.

If the lens aperture on either camera requires adjustment, it should be set fully open, the lens should be adjusted for the best focus condition and then the aperture adjusted so that the video image appears to have a full compliment of gray values : from pure white to black.

7.4 Acquiring a Video Image

Once the lights and cameras are on and adjusted properly, the two video images can be acquired and displayed by actuating the 'FRAME GRAB TO COMPUTER' command button. Formation of a video image takes approximately 30 seconds, at which time the image will appear in the upper left window or the upper right window of the screen, depending on which camera is activated. The other camera is then selected by clicking on the appropriate command button, and the process is repeated. Note that the external monitor video signal will respond to frame grabs by freezing the current video image, but can be un-frozen without affecting the grabbed image by selecting the 'LIVE VIDEO TO MONITOR' menu button.

Alternatively, if a video image is already available in a disk file, the file can be retrieved and displayed by selecting the 'READ VIDEO IMAGE FILE' command button. After selecting this option, a text widget will pop-up on the screen so that the user may enter a file name. Note that a particular video image file name will refer to two files, one holding the top view image (*filename*_top) and the other holding the side view image (*filename*_side). Both files will automatically be retrieved and displayed based on the root file name. For instance, if two files named 'target.img_top' and 'target.img_side' exist, both of them will be read and displayed by entering 'target.img' for the file name. When the file name has been entered in the text widget, clicking the pointing device on the 'OK' button in the text widget will pop-down the text widget and the file(s) will be retrieved.

7.5 Displaying a Radar Image

There are several ways to retrieve and display a radar image file. First, if a desired radar image file exists on the disk, the file can be read and displayed by selecting the 'READ RADAR IMAGE FILE' menu item. Again, a text widget appears on the screen so that the user may enter a file name. The radar image for both views (assuming 3-dimensional data format is present in the radar image file) is then superimposed on the current video image. If 3 dimensional data is not present in the image file, the image for the missing view will be displayed as a line of scattering centers along the target axis.

Secondly, if the radar image file is an ISAR image in raster scan format, the file can be read and displayed by selecting the 'READ ISAR IMAGE FILE' button. Note that only one ISAR image can be displayed at a time and that the ISAR image cannot be shifted or rotated for aligning with the video image.

And finally, if the RTDP program is running prior to executing xradar, radar images can be displayed in real time as the data is being collected. The interface to the RTDP program is completely transparent and selecting the 'Real Time DP' menu item will initiate this mode.

7.6 Scaling and Aligning the Radar Image

If the video and radar images are misaligned or are scaled incorrectly, the 'Radar Image Control' functions can be used to correct the problem. It should be noted that alignment or scaling of the radar image should not normally be necessary, since if the cameras are properly aligned, the video image and radar image should also be aligned and scaled properly. If alignment and/or scaling is required however, these procedures should be carried out on full screen images in order to minimize errors. It should also be noted that these procedures cannot be used on ISAR images.

If only a single radar image needs to be aligned, the process can be done at any time the radar image is displayed. Selecting the 'SHIFT RADAR IMAGE' menu item displays a full screen cross-hair cursor. The mouse is used to center the cross-hairs on the phase center of the video image of the target. Clicking the pointing device button then exits the selection and updates the position of the radar image origin. This procedure should be repeated for the both the top and side views of the target.

Rotational alignment is accomplished by selecting the 'ROTATE RADAR IMAGE' menu item. A vertical line will appear on the screen and the mouse is again used to align the line with the down-range axis of the target. This procedure should be carried out on both views of the target, and it may require several iterations to achieve a satisfactory alignment for both. Clicking a pointing device button then exits the function and updates the rotational position of the radar image.

Scaling the radar image requires knowing the actual dimensions of the extent of the target. For instance, if the target is an aircraft model, the dimensions of the wingspan, fuselage length and fuselage height are required. Scaling of the radar image is accomplished by selecting either 'TOP VIEW SCALE FACTORS' for the top view or 'SIDE VIEW SCALE FACTORS' for the side view. A small menu is popped-up to guide the user through the procedure. First, the vertical dimension item in the sub-menu should be selected by clicking on it with the pointing device. Then, using the pointing device, place the cursor at one end of the vertical dimension which is known, such as on the nose of an aircraft target. Then, push and hold the pointing device button and drag the rubber-band line to the other end of the known dimension, releasing the button when the cursor is properly positioned. Then, place the pointing device cursor in the small text window next to the dimension item just selected, and enter the actual dimension (in inches) of the line traced by the rubber band. Repeat the procedure for the other dimension and exit the procedure by clicking the pointing device on the 'OK' button of the popped-up window. The radar image is scaled and redisplayed on exit. This same procedure should be repeated for the other view of the target. It may be necessary to iterate the above procedures in order to reach a satisfactory alignment of the radar and video images.

If the system is to be used in the real-time mode, a single radar image should be used to set up alignment and scaling with the video image. This can be accomplished by selecting the 'REAL TIME DP' menu item, waiting until a radar image is displayed and then exiting this mode. The initial radar image can then be aligned with the video image using the above procedure. The real time processing mode is then be reactuated, and all future radar images should now be displayed properly.

7.7 Saving Video and Radar Images

Saving existing video and radar images files is very straightforward. Selecting the desired menu item pops-up a text widget for entering the desired file name. Clicking the pointing device on the 'OK' button writes the file and pops the widget down. If the system is in the real-time mode, the radar image data is saved in the RTDP program, but alignment and scaling information will not be saved. It should be noted that under the UNTEK V operating system, an existing file with the same name will be over-written, and this system does not check for the existence of a file before writing. The formats of all files used by X-Radar are listed in detail in Appendix D.

7.8 Changing the Color Map

An alternate monochrome color map is provided if the user wishes to display the radar image, as well as the video image, using a monochrome gray scale. This feature is provided so that the user may generate screen prints for use in publications which are limited to monochrome reproductions. Clicking the pointing device button on the 'SWITCH TO MONOCHROME' menu item will redisplay the images using this gray-scale map. Clicking the pointing device button again on the same menu item (the menu item label changes to reflect the next state) will return the system to a full color color-map.

7.9 Program Usage Examples

7.9.1 Example #1 : Capturing a video image and aligning with a radar image

1. The execution of 'frame_init' has been added to the re-boot procedure on the XD88-30 in use at the ElectroScience Laboratory, so there is no need to execute this from a command line for this installation, however in other installations, it may be necessary to do so.
2. Turn on the auxiliary lighting and adjust the position of the cameras so that they are centered on the pedestal axis in both views. If necessary, adjust the lens aperture on both cameras for correct exposure. Camera position, lighting and exposure can all be monitored in real time on the external video monitor by selecting the 'LIVE VIDEO TO MONITOR' menu item and switching between the two cameras using the 'SWITCH CAMERA' menu selection.
3. Capture video images of both views by selecting the 'FRAME GRAB TO COMPUTER' menu item. The currently selected camera will determine which view is captured and displayed, and the other view is captured by switching to the other camera using the 'SWITCH TO ... CAM' menu item. Figure 24 shows the screen after a complete video image frame grab.
4. The radar image file is read and displayed by selecting the 'READ RADAR IMAGE FILE' menu item.
5. If necessary, scale the radar image to the video image by first selecting full-screen mode for the top view and then selecting the 'SCALE RADAR IMAGE' menu item. Select the 'SCALE VERTICAL DIMENSION' item on the sub-menu and position the cursor at the bottom end of the known vertical dimension. Push and hold the mouse button while

ORIGINAL PAGE IS
OF POOR QUALITY

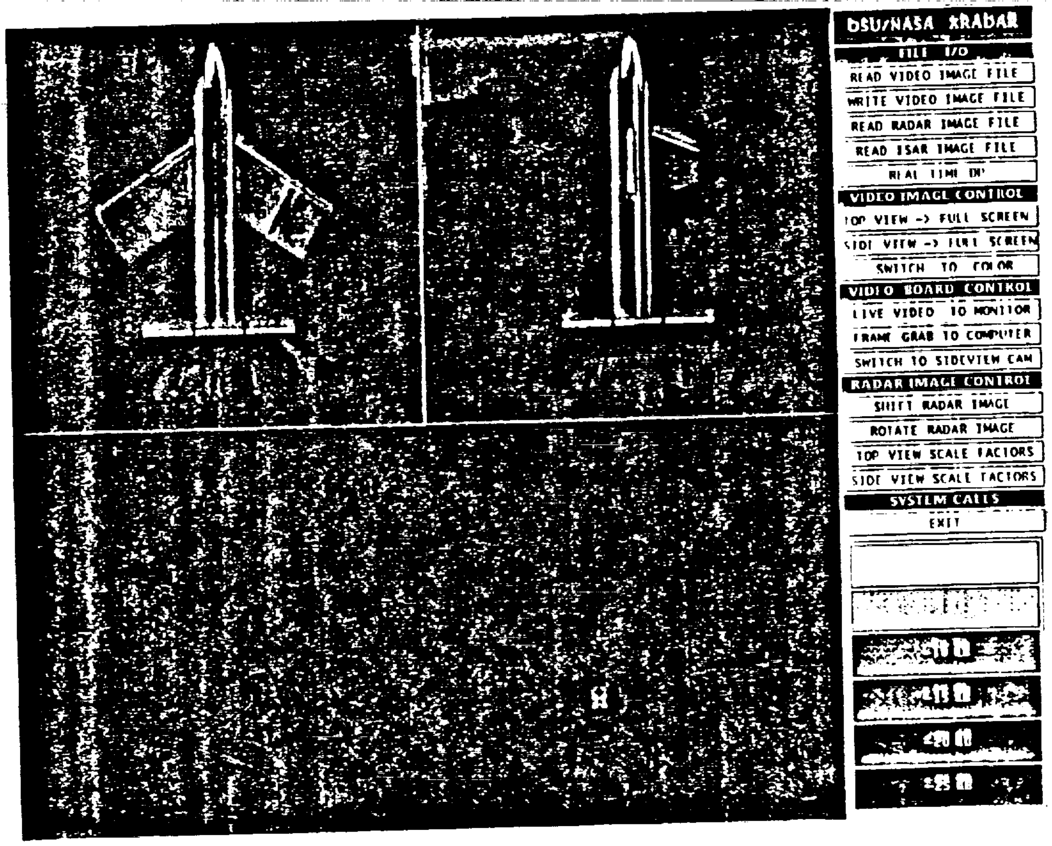
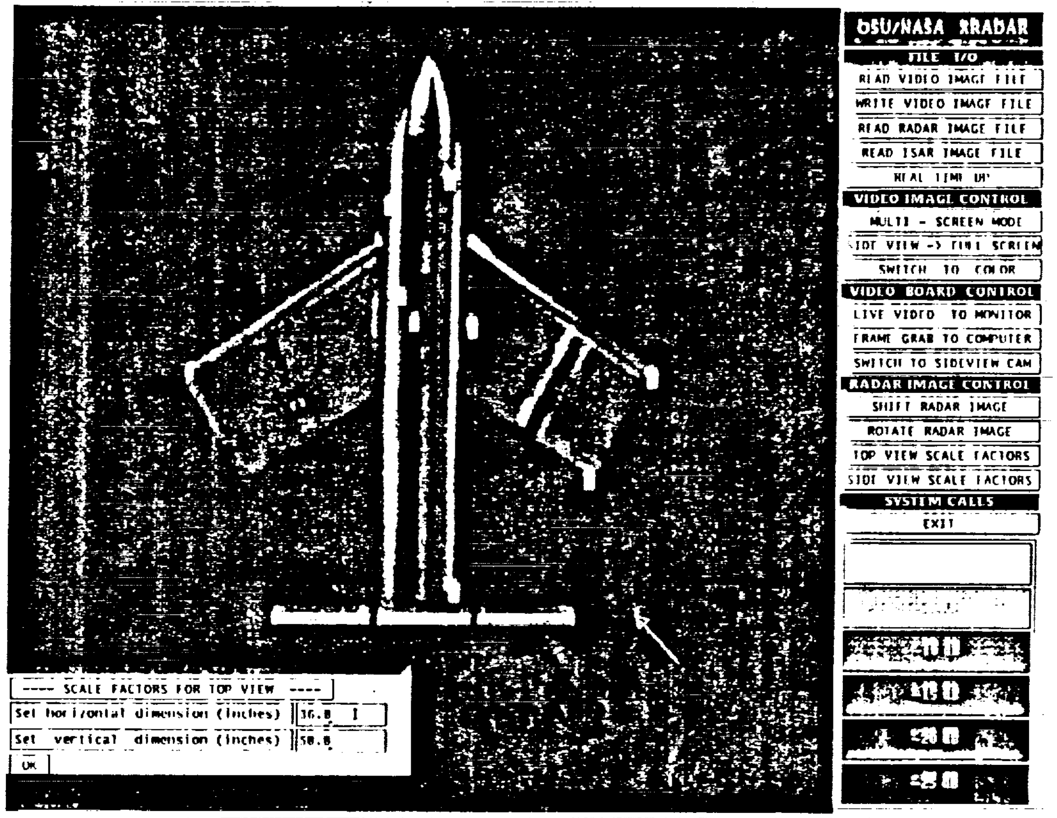


Figure 24: Main screen after video image capture of aircraft test body.

ORIGINAL PAGE IS
OF POOR QUALITY



---- SCALE FACTORS FOR TOP VIEW ----

Set horizontal dimension (inches)	36.8
Set vertical dimension (inches)	58.8

OK

Figure 25: Setting the top view scale factors.

positioning the cursor at the top end of the known vertical dimension. Release the mouse button and move the cursor to the small window to the right of the 'SCALE VERTICAL DIMENSION' sub-menu item. Enter the known vertical dimension in inches. Repeat this procedure for the known horizontal dimension, entering it in the appropriate window. Figure 25 shows the top view screen while setting the top-view scale factors. When complete, select the 'OK' item on the sub-menu. Switch to full-screen display for the other view and repeat this procedure for that view.

6. If necessary, rotate the radar image to align with the video image. While this may be difficult if the down range axis of the radar image cannot be determined, if the cameras are properly aligned, only a very slight rotation will be needed. Select the full-screen mode of the view to rotate and then select the 'ROTATE RADAR IMAGE' menu item and move the mouse cursor until the cursor line is parallel with the down range axis of the video image. Click any mouse button to accept the desired rotation and re-display the rotated radar image. Figure 26 shows the top view screen while setting the rotational offset. This procedure may be repeated as necessary. Switch to the full-screen display of the other view and repeat for that view.

7. Also if necessary, shift the radar image by selecting the 'SHIFT RADAR IMAGE' menu item and moving the cross-hairs to the phase center of the video image. Figure 27 shows the top view screen while setting the top-view offsets. Click any mouse button to accept the shift and re-display the shifted radar image. Switch to the full-screen display or the other view and repeat.

8. The above three steps may be repeated until a satisfactory alignment and scaling has been achieved.

9. The video images should be saved by selecting the 'WRITE VIDEO IMAGE FILE' menu selection. Enter the desired image file name in the sub-window and select the 'OK' item in the sub-window. Figure 28 shows the top view screen while entering the filename for the video image.

10. Exit X-Radar by selecting the 'EXIT' menu item.

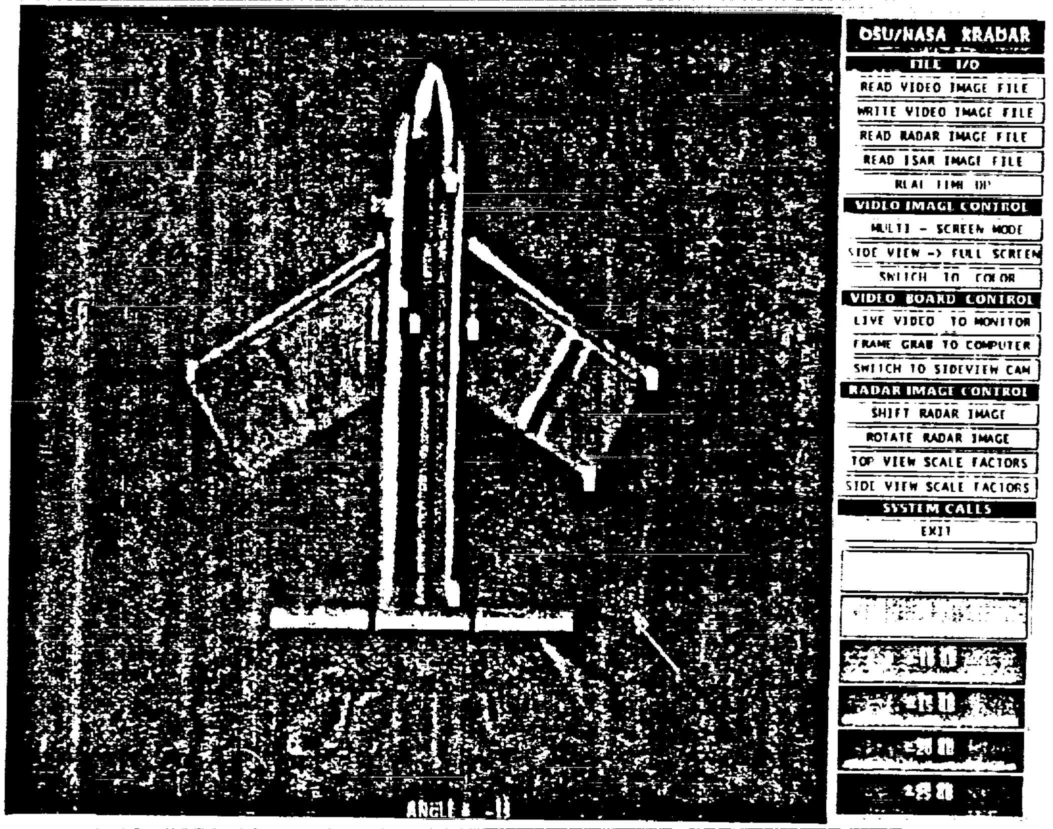
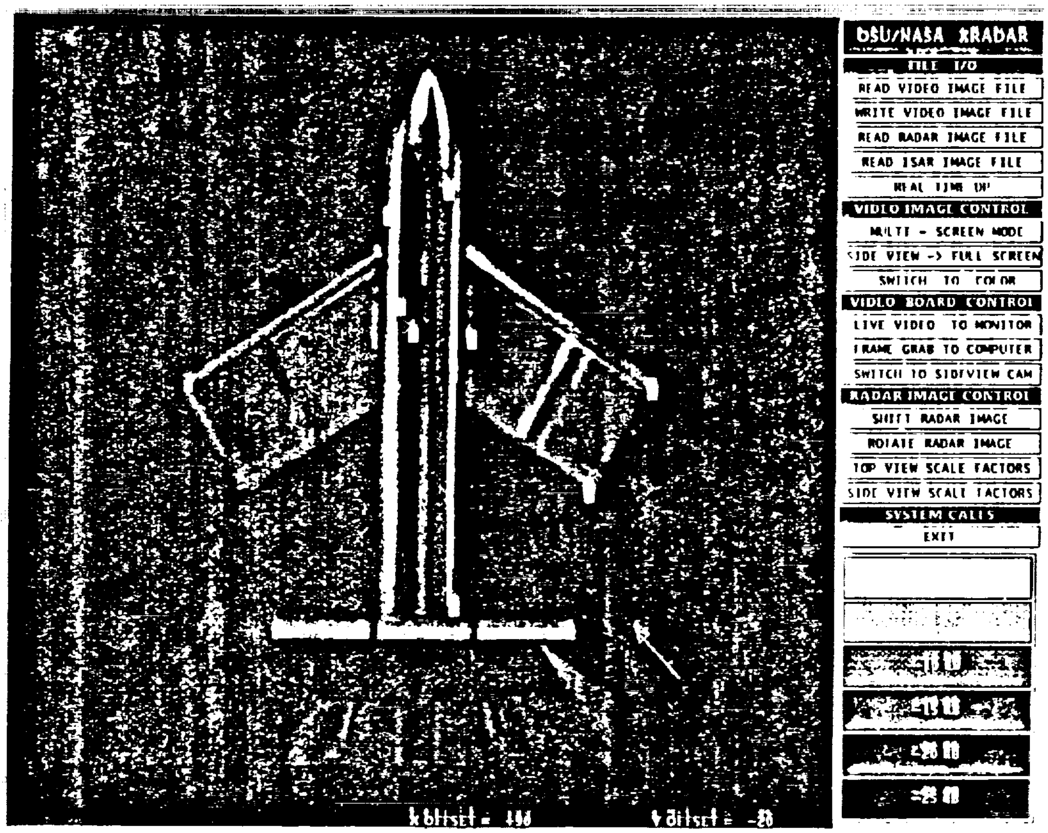


Figure 26: Setting the rotational offset.

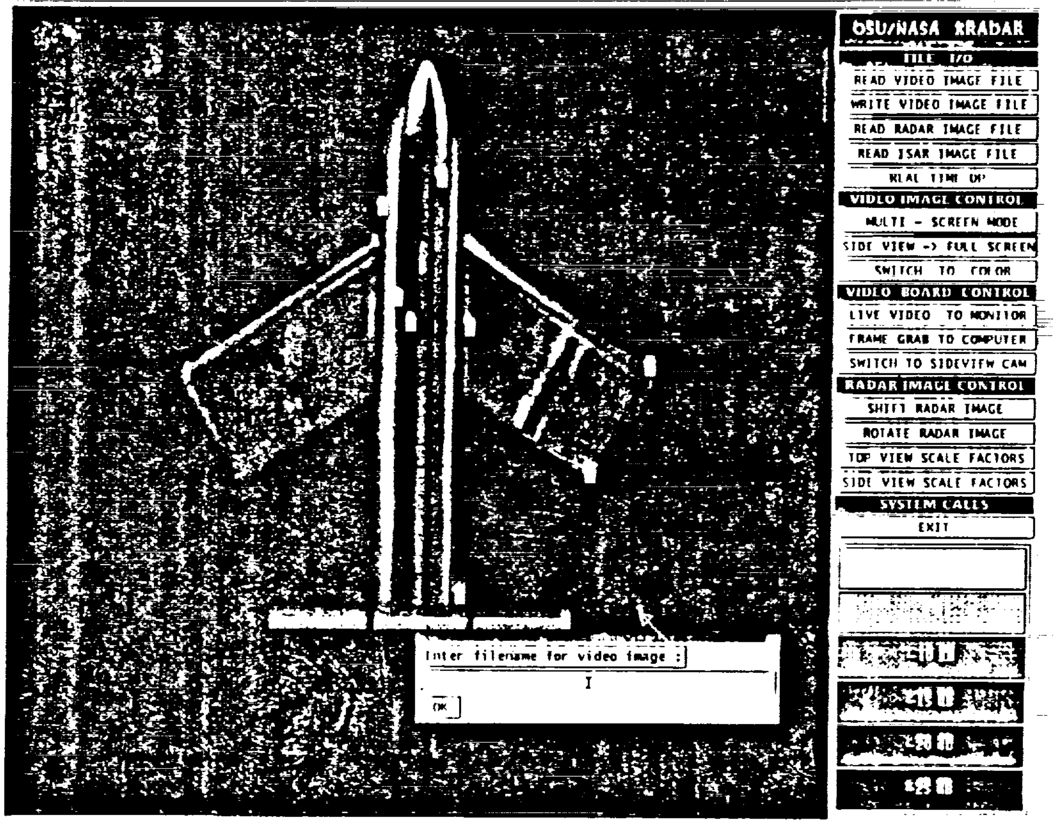
ORIGINAL PAGE IS
OF POOR QUALITY



x offset = 190   y offset = -28

Figure 27: Setting the top view offsets.

ORIGINAL PAGE IS
OF POOR QUALITY



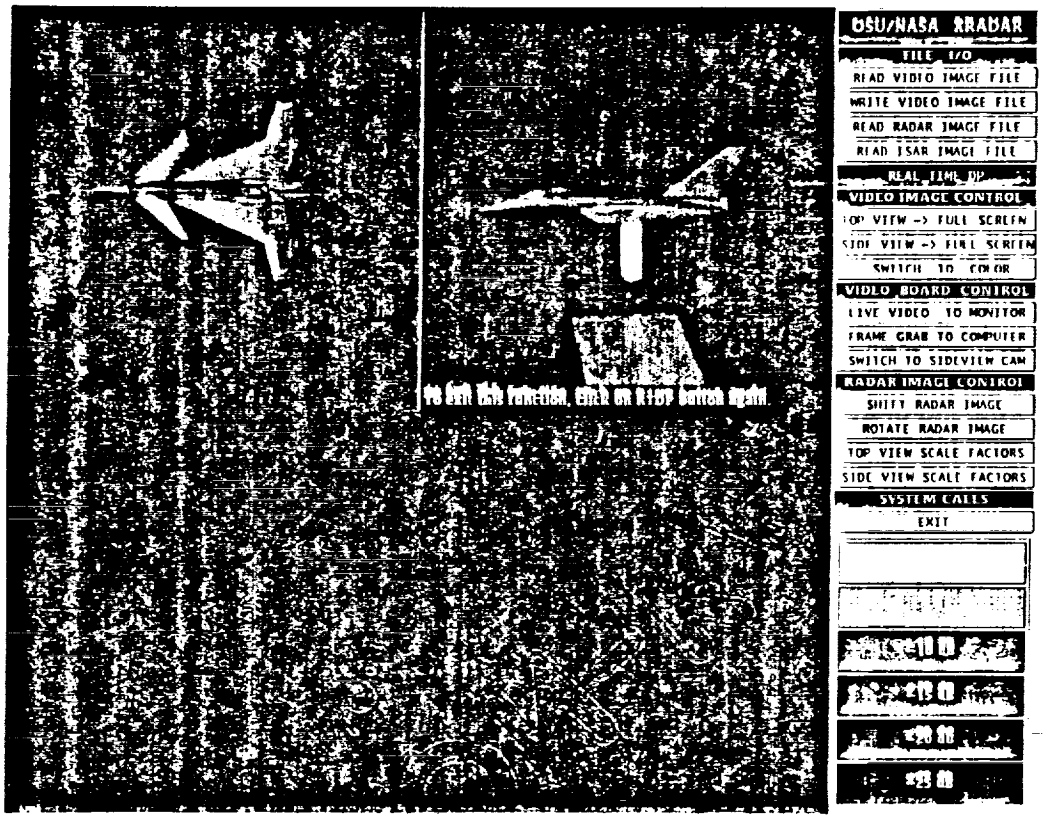
Enter filename for video image :

Figure 28: Entering the video output filename.

ORIGINAL PAGE IS
OF POOR QUALITY

7.9.2 Example #2 : Displaying Real-Time Radar Images on a Captured Video Image

1. Follow step 1 in Example #1, if necessary.
2. Execute the Real Time Data Processing Program (RTDP) [14].
3. After the RTPD program is running, and all parameters have been set, click any mouse button on an open area of the screen until a VT102 window becomes visible. Place the mouse cursor in the VT102 window and execute 'xradar'.
4. Follow steps 2 and 3 in Example #1.
5. Select the 'REAL TIME DP' menu item and wait until a radar image is displayed. Figure 29 shows the main screen after entering the real time data processing mode. If scaling and alignment are necessary, click any mouse button on the 'REAL TIME DP' item again to stop the radar image acquisition. Use the displayed radar image to scale and align the radar and video images, as described in Example #1. Select the 'REAL TIME DP' item again to continue with the real-time display.
6. During real-time display, all of the menu items, such as switching display modes and color maps, remain active and can be executed at any time. Figure 30 shows the top view screen after selecting 'TOP VIEW – > FULL SCREEN' menu item while in real time mode.
7. When radar image display is complete, exit the real-time mode by selecting the 'REAL TIME DP' menu item again.
8. If desired, save the video images by selecting the 'WRITE VIDEO IMAGE FILE' menu item and entering a file name. Select the 'OK' item on the sub-window to accept the file name and write the file.
9. Exit X-Radar by selecting the 'EXIT' menu item.



To exit this function, click on RTDP button again.

Figure 29: Main screen after entering the RTDP mode.

ORIGINAL PAGE IS
OF POOR QUALITY

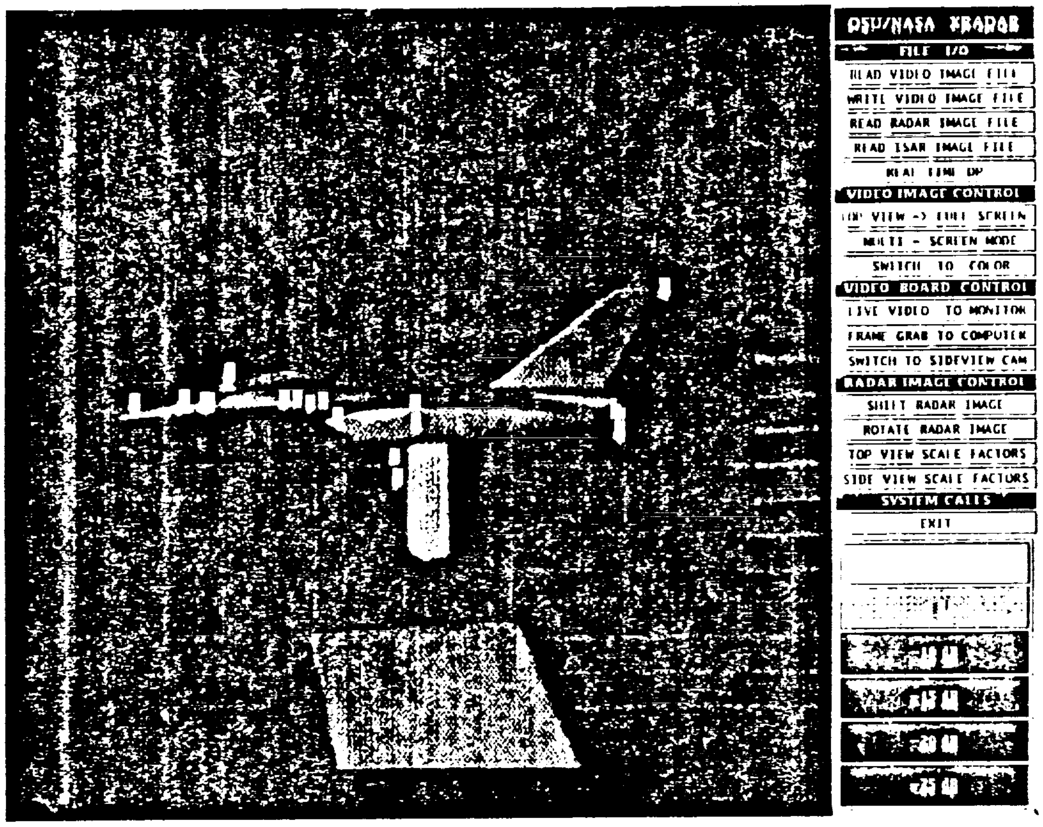
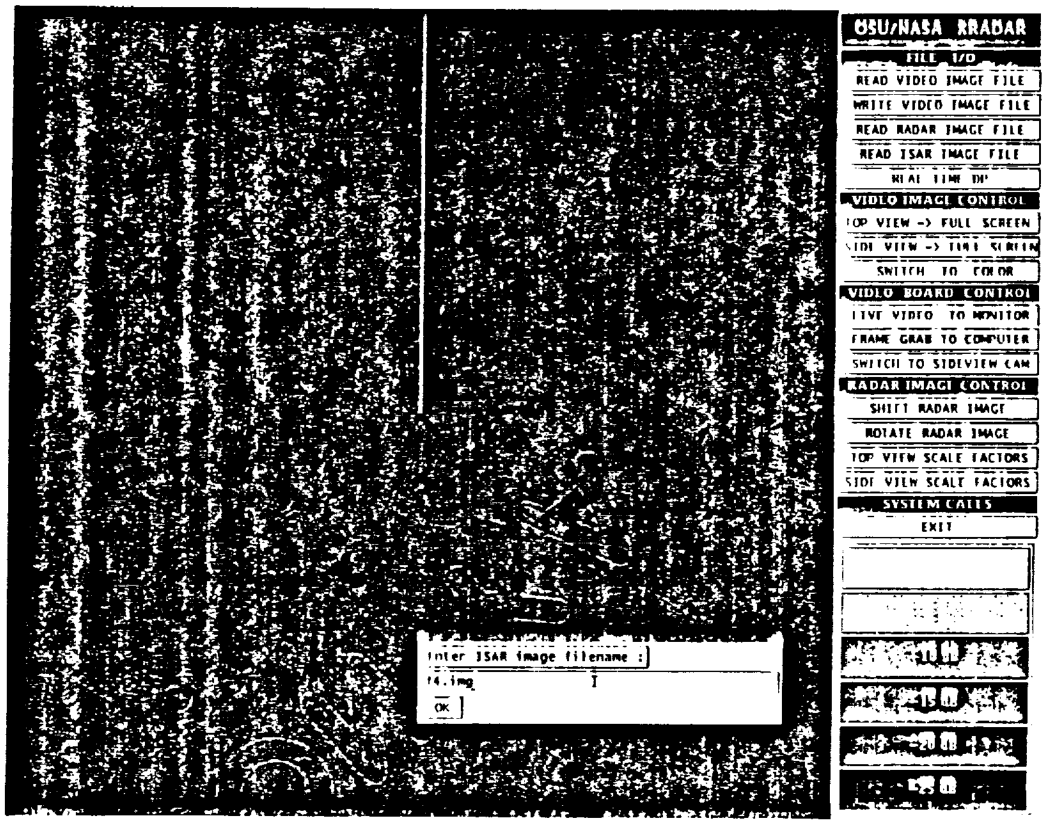


Figure 30: Side view of aircraft test body during RTDP mode.

ORIGINAL PAGE IS
OF POOR QUALITY

7.9.3 Example #3 : Displaying an ISAR image

1. Follow steps 1, 2 and 3 in Example #1.
2. Read the ISAR image file by selecting the 'READ ISAR IMAGE FILE' menu item. Figure 31 shows the main screen after selecting the 'READ ISAR IMAGE FILE' menu item, while Figure 32 shows an ISAR image of an F4 aircraft read from a file after selecting 'TOP VIEW – > FULL SCREEN' (the image data in this file has been padded to create the black border and white background).The ISAR image should conform to the file format listed in Appendix D. ISAR images cannot be scaled, rotated or shifted, so this is all that can be done.



Enter ISAR image filename :

t4.img

Figure 31: Main screen for ISAR file name entry.

ORIGINAL PAGE IS
OF POOR QUALITY

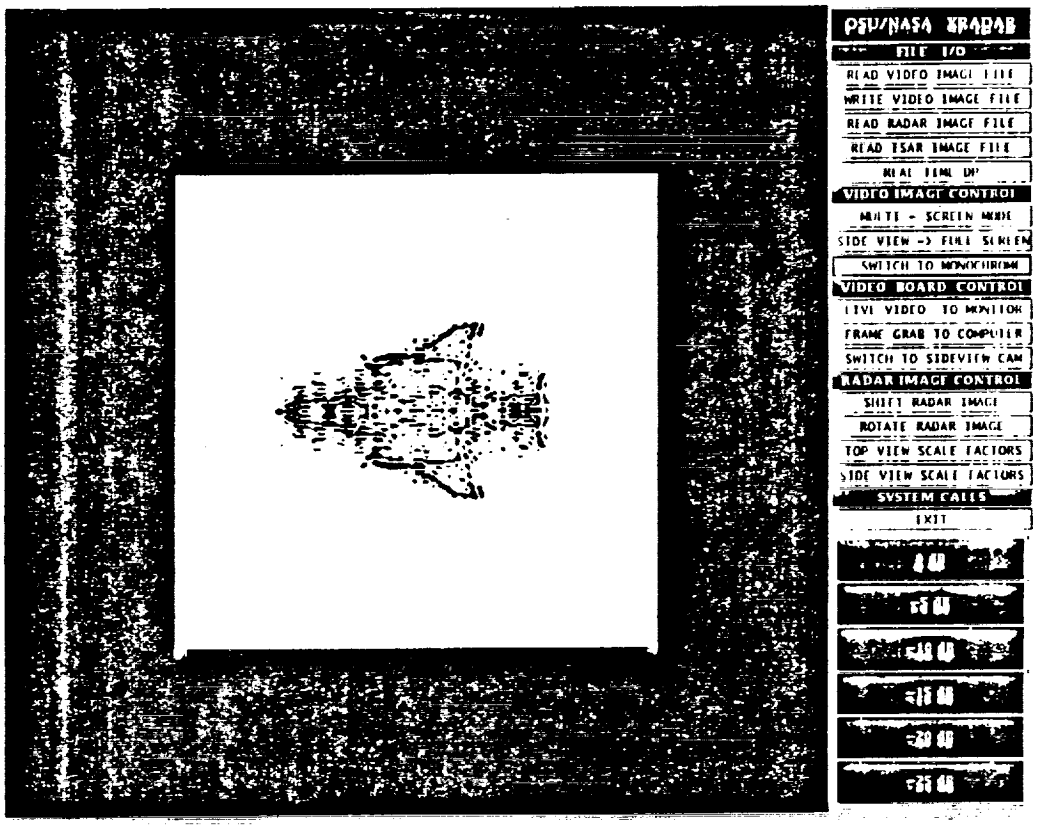


Figure 32: ISAR image of an F4 aircraft.

Chapter 8

Conclusion

A hardware and software system for the display and analysis of radar images has been described in this report. This system was designed to provide the user with a versatile and efficient way to examine different types of radar images superimposed on a video image of the actual target measured. This has been accomplished by using a menu-driven software architecture to allow the user to easily control the various hardware components of the system. In addition, since the software is event-driven, adjustments (scale, shift and rotation) to the radar images can be done interactively even in the real-time mode. This allows the user to make fine adjustments to the images as the processing is being done, rather than having to wait until processing is complete.

The capabilities of the system have surpassed the original goals. The system is capable of displaying radar images in the real time mode at a rate of approximately one every three seconds. In addition, images can be recorded on video tape, or hardcopy can be produced on a color printer. The system provides the user with complete documentation of each radar image produced and serves as a diagnostic tool for the radar system.

The system has undergone extensive testing during development and has been recently put into use in the compact range at the ElectroScience Laboratory. While the original design goals have been exceeded, new areas of application and development have become evident. Possible areas for further development are in the addition of features for interactively analyzing individual scattering centers, correlating scattering centers in one view with scattering centers in the other view and developing an isometric view of the video and radar images.

In addition, there are several parts of the software that could be improved. While both the RTDP program and X-Radar are running, there is often a need to actuate the RTDP menus. While this can currently be achieved by cycling through the various windows being displayed (by using the mouse buttons), it would be much more convenient if this could be achieved with only one mouse event. This could be achieved if both the RTDP program and X-Radar were each placed under an application shell widget; in other words, each program should have an application shell widget which covers the entire screen as the upper-most parent in the program.

The real-time image display could also be improved. As written, each new radar image requires that the entire video display be erased. A substantial improvement in speed could be achieved if only the previous radar image could be erased. This could be achieved by using the function XClearArea() for each scattering center marker, and then letting the ExposeEvent event handler fill in the missing part of the video image.

Appendix A

Programming the FG100V Video Board

Programming the FG100V video board requires a complete understanding of the way in which the XD-88 bus address space is organized, and an understanding of the UTEK V system calls available to carry out virtual address space memory mapping. The address space in the XD-88 is organized into two separate, but related address spaces : the VME address space and the Futurebus address space. The Futurebus is the internal system bus used by the system for disk I/O, CPU interconnection and processing. The VME bus is used for interfacing the system to the graphics sub-systems and 3rd-party vendor hardware [13]. The XD88 memory space is organized as shown in Figure 33.

A.1 VME Protocol Address Spaces

The VME protocol specifies three different address spaces. VME 'standard' address space requires a 24-bit address and hence can access up to 16Mbytes. VME 'short' address space only uses 16-bit addresses and so only addresses 64 Kbytes of space. There is also VME 'extended' address space, which uses 32-bit addresses. For many I/O purposes, the short address space is sufficient and the standard and extended address spaces are usually only used for devices which have large amounts of memory. Which address space is being accessed is determined by a byte known

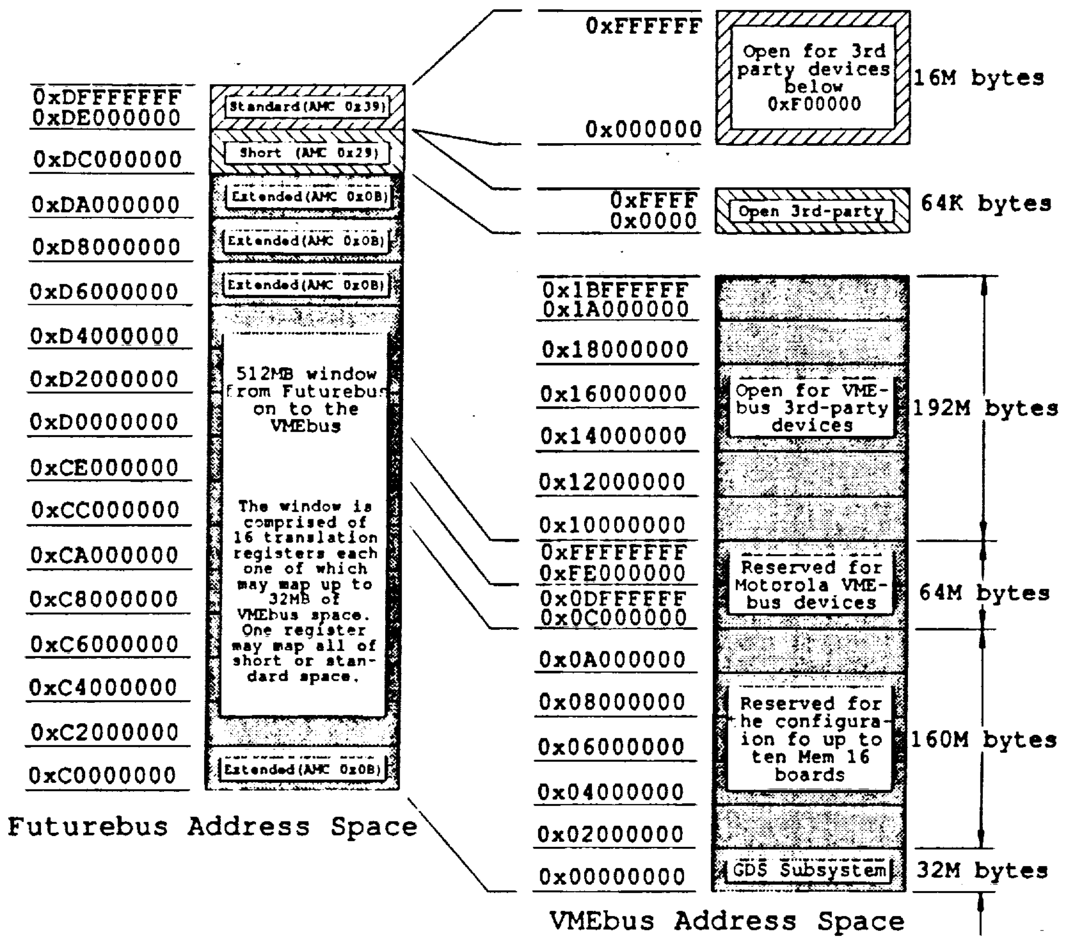


Figure 33: Memory organization in the XD88 Workstation.

as the Address Modifier Code (AMC). An AMC = 29 (hex) specifies the short address space, an AMC = 39 (hex) specifies the standard address space and an AMC = 0B (hex) specifies extended address space. The relationship between the AMC, the Futurebus and VME address space is shown in Figure 34. The details of how the user sets this byte will be described later in this chapter.

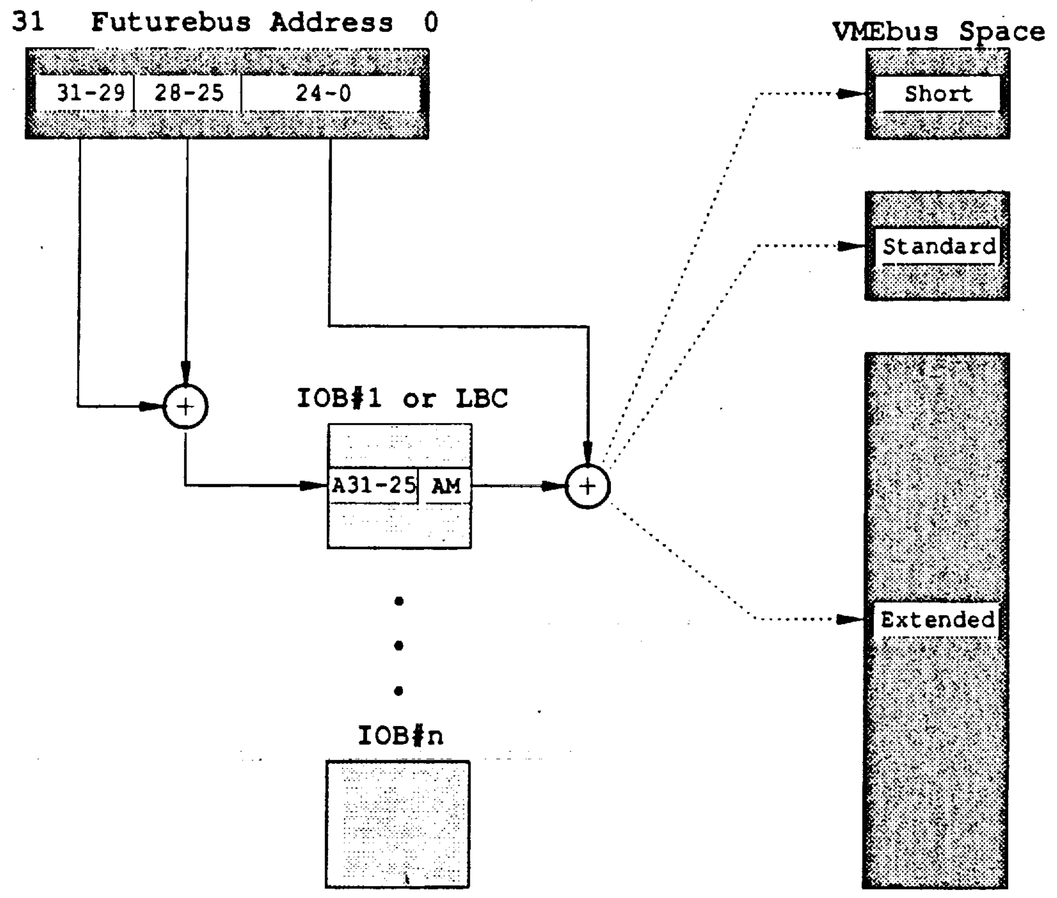


Figure 34: Usage of the AMC to calculate the physical address of a VME device.

A.2 XD88 VME Device Address Space

The VME standard and short address spaces are mapped into two windows in the Futurebus address space. VME extended address space is mapped onto the Futurebus in 32Mbyte windows. The FG100V requires mapping into both the short address space, for the control registers, and into VME standard address space, for the video buffer memory.

Figure 35 shows the control register map for the FG100V [11]. Notice that all registers are referenced as *byte* offsets from the base address. This is important since the board only supports *word* transfers. This will effect what type of pointer is used when accessing the control registers.

A.3 Shared Memory Initialization

VME address space is accessed in the XD-88 by mapping it into the real address space of the machine. This is done by allocating 'shared memory', that is, memory shared between the user and the system. The allocation of shared memory on the XD88 is reserved for superusers only. This means that a separate program must run at the superuser level to initially allocate the memory and that this program must also allow non-superusers to access the memory segment. The program 'frame_init.c' is used to initialize shared memory for the FG100V video board and is listed in Appendix B. The most important part of the code are the define statements, which are reproduced in the partial listing below.

A.3.1 Listing 1 -- SHM_INIT.C

```
#define REG_PHYS_ADDR 0x0      /* Physical address of control regs.*/
#define MEM_PHYS_ADDR 0xA00000 /* Physical address of video memory */
#define REG_REGION_SIZE 0x2000   /* size of reg. region in bytes      */
#define MEM_REGION_SIZE 0x100000 /* size of mem. region in bytes      */
#define REG_KEY 0x1000   /* key for reg shm       */
#define MEM_KEY 0x1001   /* key for mem shm       */
```

```
#define SHMFLAG   (IPC_PHYS |
                        IPC_CREAT |
                        IPC_NOCLEAR |
                        IPC_CI |
                        0777)    /* define shared memory flags */

#define REG_PHYS_SPACE (PHYS_VME |
                       PHYS_VME_SHORT |
                       PHYS_VME_DATA)     /* AMC for control regs.*/
#define MEM_PHYS_SPACE (PHYS_VME |
                       PHYS_VME_STD |
                       PHYS_VME_DATA)     /* AMC for frame buffer */
```

A.3.2 Explanation of Listing #1

A complete description of each of the define statements and the initialization code follows below :

1. REG_PHYS_ADDR and MEM_PHYS_ADDR define the physical addresses of the control registers and the frame buffer memory. As noted in the listing, there is a bug in the function shmget() which requires defining the control register physical address as 0000 (hex) and adding an offset to get the actual physical address. The addition of the offset is not done in the initialization program, but instead is handled in the actual FG100V functions. The shmget() function seems to work normally when mapping the address for the frame buffer memory, and the physical address defined for it, A00000 (hex), is correct.

2. REG_REGION_SIZE and MEM_REGION_SIZE define the amount of memory in bytes that is needed for the control registers and the frame buffer memory respectively.

3. REG_KEY and MEM_KEY are the most important defines here. These values allow other processes to access the memory areas allocated by this program, and therefore, these values must be known to other applications that wish to use the space and the FG100V video board.

4. SHMFLAG defines a set of bit flags passed to the shmget() function. The exact details of each flag will not be explained, so unless the actions of a specific flag are in question, the values listed here should be used. The values of the flags are defined in the UTEK system header file 'ipc.h' .

5. REG_PHYS_SPACE and MEM_PHYS_SPACE define a set flags which define the complete AMC used for each mapped area. The flags are defined in the header file 'shmphys.h', but are fairly self-explanatory. The define for REG_PHYS_SPACE should be used as shown when mapping an I/O device

to VME short address space and the define for MEM_PHYS_SPACE should be used as shown when mapping a device to standard address space.

The remainder of the initialization program is fairly simple. After turning off signaling for bus errors (a good idea while debugging code for VME bus devices), address space for the control registers is allocated. The shmget() function returns an ID number for the memory space, if an error doesn't occur. Errors are trapped to a fatal exit after printing a description of the error.

Next, the memory area for the control registers is set to allow access by non-superuser applications. The structure of type shmBusTable is defined in the UTEK V system header file 'shm.h' and is used to pass information about the shared memory area to the function shmctl(). This function performs a number of different functions and is outlined fairly well in Reference [12]. Here it is used to set the flag SHMPHYS_ALLOW so that the memory space can be accessed by any application. Again errors are trapped to a fatal exit after printing an error message and freeing the memory allocated by the previous shmget().

The previous instructions are repeated for the frame buffer memory space. Again note that before exiting due to an error, previously allocated space is freed. This is done because the allocations are permanent, and are not freed just because an error occurred. These allocations are only deleted from the memory management table by a free() instruction or at power-down.

This completes the initial allocation of shared memory space for a VME device. The next step is to attach a process to the memory space which has been allocated.

A.4 Shared Memory Attachment in X-Radar

The application is 'attached' to the previously allocated virtual memory space through the same system calls used to allocate the memory. However, since the same allocation 'key' is used, the system does not re-allocate the space, but simply returns an identification number for use by the application. The function, ITI_init(), used to attach the allocated virtual memory space to X-Radar is listed in Appendix B, module 'xframe.c'.

Again, the define statements are very important, and are listed in Appendix B, module 'radar.h'. They are virtually identical to the define statements in the initialization program in Appendix B. It is extremely important that the values of REG_KEY and MEM_KEY are the same in both, as this is the mechanism used to associate the previously allocated memory space with the application. Also, it is important that the amount of memory requested in the application program be the same or less than the amount allocated in the initialization program.

The main part of the attachment function is very similar to the initialization program. Following the declarations of variables and again turning off bus error signalling, the shmget() function is called to get an ID number. However, since this portion of memory has already been allocated, the ID number returned is the same one returned by shmget() in the initialization program. Now, the function shmat() is used to attach the ID number to this application. As in the initialization program, any time a fatal error occurs, the application must 'clean up' the environment before exiting. In this case however, the function shmdt() is used to detach the application from the allocated memory. Note that this does not free the memory, it just releases the application from the memory management table.

The value returned by shmat() is a pointer to the beginning of the shared memory space. In the case of the control register memory space, this pointer points to the address of the first control register, and other control registers can be accessed by adding the appropriate offset to the pointer. The same is true for the frame buffer. The pointer returned by the corresponding call to shmat() points to the beginning of the frame buffer.

One of the problems encountered during the development of this code was the type declarations for the various pointers. Since the board only supports word addressing, the pointers must be declared as 'shorts', which are 16-bit words.

However, the offsets for the control registers are specified in bytes, so all the offsets listed in the FG100V user's manual [11] must be divided by two to get the offset in words.

A.5 FG100V Control Software

With the required memory mapping established, the FG100V can be controlled. Control of the board entails setting particular control registers in a particular sequence to accomplish the required process. The functions written to control the FG100V are listed in Appendix B, module 'xframe.c', and are outlined here.

1. write_register() : A function to write a data value to a FG100 control register. Notice that the register offset is divided by 2. As mentioned previously, this is because the register offsets are given in bytes, but the FG100 only accepts word addresses.
2. read_register() : A function to read the value of an FG100 control register. Again, the register offset is divided by 2 for the same reason as mentioned above.
3. wait_vb() : This function waits for a vertical blanking period in the video signal. This is necessary in order to insure that the FG100 grabs a complete frame.
4. ITI_init() : As discussed previously, this function is used to attach X-Radar to the previously allocated shared memory spaces. It also sets the FG100 control registers to default values. A series of write_register() calls at the end of the function set these default values and are explained in the comments.
5. ITI_lut() : This function initializes the FG100 hardware color look-up tables. The FG100 has up to 16 different look-up tables, but for this simple application, only four are used : one each for RED, GREEN, and BLUE and one for the intermediate look-up table. In this application, all the look-up tables are configured as linear ramps, and the intermediate look-up table is installed between the camera and the video A/D converter.
6. ITI_frame() : This function grabs a frame of video and displays it on the screen. There are two things that the function must do : first, it must wait for a vertical blanking period so that a complete frame is digitized, and secondly, it must wait for any previous command to terminate. The wait_vb() function accomplishes the first, the second being accomplished by the while() statement in line 6. At this point, the camera selected is used to

set the data to write to the FG100 in order to initiate the frame grab. Finally, the function waits for the frame grab to complete. The remaining part of the code reads the video data from the FG100V and forms and displays the video image on the XD88. The code checks whether X-Radar is displaying both the top and side view or just one or the other and then displays the video image in appropriate window on the screen.

7. ITI_cont() : This function simply puts the FG100 in the continuous acquisition mode so that the video signal from the cameras can be viewed in real-time on the external system monitor. This is useful for aligning the cameras and monitoring the target in the compact range chamber.

8. ITI_cam() : This function switches between the two cameras as the current video source.

9. ITI_close() : This function simply releases the shared memory attachment of X-Radar. This function is only called on exiting the application.

REGISTER BASE ADDRESS +

0	MEMORY ACCESS CONTROL
2	HOST MASK
4	VIDEO ACQUISITION MASK
6	PIXEL BUFFER REGISTER
8	X POINTER
A	Y POINTER
C	POINTER CONTROL
E	CPU ADDRESS CONTROL
10	X SPIN CONSTANT
12	Y SPIN CONSTANT
14	PAN A
16	LOOKUP TABLE CONTROL
18	SCROLL A
1A	BOARD STATUS/CONTROL
1C	ZOOM CONTROL
1E	FRAME MEMORY DATA PORT

Figure 35: FG100V control register map.

Appendix B

Listing of X-Radar

B.1 Program ITI_init.c

```
/*
 * This program sets up shared memory management for FG100 frame grabber.
 */
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/io/shmphys.h>
#define REG_PHYS_ADDR 0x0 /* Physical address of control regs. */
#define MEM_PHYS_ADDR 0xA00000 /* Physical address of video memory  */
#define REG_REGION_SIZE 0x2000   /* size of reg. region in bytes      */
#define MEM_REGION_SIZE 0x100000 /* size of mem. region               */
#define REG_KEY 0x1000 /* key for reg shm */
#define MEM_KEY 0x1001 /* key for mem shm */
#define SHMFLAG   (IPC_PHYS | IPC_CREAT | IPC_NOCLEAR | IPC_CI | 0777)
#define REG_PHYS_SPACE (PHYS_VME|PHYS_VME_SHORT|PHYS_VME_DATA)
#define MEM_PHYS_SPACE (PHYS_VME|PHYS_VME_STD|PHYS_VME_DATA)
extern int errno;
extern char *sys_errlist[];
main()
{
int shmid_reg,shmid_mem;
struct shmBusTable addr;
char *shmat();
signal(SIGBUS,SIG_IGN);
/* set up shared address space for control registers */
if ((shmid_reg = shmget(REG_KEY,
REG_REGION_SIZE,
SHMFLAG,
REG_PHYS_ADDR,
REG_PHYS_SPACE)) < 0)
  {
```

```
  fprintf(stderr,
       "Error allocating shared memory for control registers\n");
fprintf(stderr,"shmget: errno: %d,%s\n",
errno,sys_errlist[errno]);
exit(1);
  }
addr.start = (caddr_t)REG_PHYS_ADDR;
addr.length = REG_REGION_SIZE;
addr.space = REG_PHYS_SPACE;
addr.uid = 0;
addr.gid = 0;
addr.perm = 0777;
/*
 *  Allow non-superuser applications to use the space.
 */
if(shmctl(shmid_reg,SHMPHYS_ALLOW, &addr) != 0)
{
fprintf(stderr,"shmctl: %d %s\n",errno,sys_errlist[errno]);
exit( 1 );
}
/*
 *  Open VME standard space for frame buffer
 */
if ((shmid_mem = shmget(MEM_KEY,
MEM_REGION_SIZE,
SHMFLAG,
MEM_PHYS_ADDR,
MEM_PHYS_SPACE)) < 0)
  {
  fprintf(stderr,
       "Error allocating shared memory for video memory\n");
fprintf(stderr,"shmget: %d -- %s\n",
errno,sys_errlist[errno]);
exit(1);
  }
addr.start = (caddr_t)MEM_PHYS_ADDR;
addr.length = MEM_REGION_SIZE;
addr.space = MEM_PHYS_SPACE;
addr.uid = 0;
addr.gid = 0;
addr.perm = 0777;
/*
 *  Allow non-superuser applications to use the space.
 */
if( shmctl( shmid_mem, SHMPHYS_ALLOW, &addr) != 0)
{
fprintf(stderr,"shmctl: %d %s\n",errno,sys_errlist[errno]);
exit( 1 );
}
exit( 0 );
}
```

B.2 Program xradar

B.2.1 Module xrdr.c

```
/*
 *
 * This is the main source file for the compact range image
 * display program.
 *
 *
 */
#include "radar.h" /* includes all X includes */
extern void RePaint();
extern void ITI_init();
extern int ClearRadarData();
/*** ALL Global variables are defined in this module ***/
/* GLOBAL X STUFF */
XImage          *im1, *im2,*im3, *im4;   /* image vars. for video images */
Display         *display;
Colormap        cmap;
GC              gc1, /* GC for all image display */
gc2; /* GC for cross hairs */
XFontStruct     *font,font2;
Widget toplevel, top2, shell1,
shell2, shell3,
shell4,
winwidget1, winwidget2,
winwidget3, winwidget4,
popfile1, popfile1a, popfile2,popfile3,
error_popup, popmove, exit_message,
scalepop1, scalepop2,
button1, button2, button3,
button4, button5, button6,
button7, button8, button9,
button10,button11,button12,
button13,button14,button15,
button16,button17,
box, color_scale, mono_scale,
lev1;
/** Globals for positions and sizes of graphics window widgets **/
int             gxpos1,gxpos2,gxpos3,gxpos4;
int             gypos1,gypos2,gypos3,gypos4;
int             xsize,ysize;
unsigned long   cols[256];
unsigned long   monocols[256];
unsigned long   fore, back;
unsigned long   red,white,blue,black;
/*** Globals for multiple screen manipulation ***/
int scrn1_flag = 0; /* flag set if full screen side view */
int scrn2_flag = 0; /* flag set if full screen top view */
Boolean isar_flag = False; /* flag set if ISAR image display */
Boolean mono = False; /* flag set if using mono colormap */
widget_struct       w_rec1, w_rec2;
/** Globals for radar images ***/
/*
 * array to hold radar image data.
 * structure as follows:
 * radar_data[i][0] = x coord. of ith scattering center
 * radar_data[i][1] = y coord. of ith scattering center
 * radar_data[i][2] = z coord. of ith scattering center
 * radar_data[i][3] = magnitude of ith scattering center
 * radar_data[i][4] = spare
 * radar_data[i][5] = spare
 */
float radar_data[IMAGE_PNTS][6];
int num_pnts    = 0;              /* number of points in current radar image */
float in_angle = 0.0;  /* incident angle of radar signal */
Boolean vid1_flag = False;  /* true if top view in system */
Boolean vid2_flag = False;  /* true if side view in system */
rubber_band_data rb_data;       /* structure for rubber band lines */
cursor_data cur_data; /* structure for full screen cursor */
line_data l_data; /* structure for rotationn cursor */
char xscale_str[10],              /* strings to hold scale dimensions */
     yscale1_str[10],
     yscale2_str[10],
     zscale_str[10];
float x_scale = 512./96.;  /* default scale factors */
float y1_scale = 512./96.;
float y2_scale = 512./96;
float z_scale = 512./96.;
int x_dim, y1_dim,
    y2_dim, z_dim;    /* # of pixels for scaling */
int x_offset = 0;   /* offsets for adjusting radar image */
int y1_offset = 0;
int y2_offset = 0;
int z_offset = 0;
float xy_angle = 0.0;   /* rotation angles for radar image */
float yz_angle = 0.0;
int horizontal_flag;              /* flag : True = setting horiz. scaling */
/**  GLOBAL FILE STUFF **/
char buffer1[80], buffer1a[80], buffer2[80];
FILE *fd_mapfile;
unsigned char *vidray1, *vidray2, *isar1, *isar2;  /* pointers to raw data */
unsigned char *Image1 , *Image2;     /* pointers to image data */
unsigned char *BigImage1, *BigImage2;    /* pointers to image data */
/*************************************************************************/
main(argc,argv)
int argc;
char *argv[];
{
    /* set default image sizes and locations */
    xsize = 512;
    ysize = 480;
    gxpos1 = 0;
    gypos1 = 0;
    gxpos2 = 512;
    gypos2 = 0;
    gxpos3 = 0;
    gypos3 = 512;
    gxpos4 = 0;
    gypos4 = 0;
    /* prepare everything in startup routine */
    initialize();
    /* setup user interaction */
    xinteract();
    /* loop for events */
    XtMainLoop();
} /*end main()*/
/*****************************************************************************/
/*  initialize -- does general initialization      */
initialize()
{
Arg arg[10];
int mapchars, bitmap_pad;
unsigned int depth;
mapchars = 512 * 512;
/*
 *  Allocate memory for raw data arrays.
 */
if( !(vidray1  = malloc(mapchars))   ||
    !(vidray2  = malloc(mapchars))   ||
    !(isar1    = malloc(mapchars))   ||
    !(isar2    = malloc(mapchars)))
{
printf("Unable to allocate memory for data arrays...\n");
exit(-1);
}
/* allocate memory for small image data arrays */
if( !(Image1   = malloc(2*mapchars)) ||
    !(Image2   = malloc(2*mapchars)) )   /* 2x for 12 bit depth */
{
printf("Unable to allocate memory for small image arrays...\n");
exit(-1);
}
/* allocate memory for big image data arrays */
if( !(BigImage1 = malloc(2*4*mapchars)) ||
    !(BigImage2 = malloc(2*4*mapchars)) )
{
printf("Unable to allocate memory for big image arrays...\n");
exit(0);
}
clear_array(vidray1,mapchars);    /* clear arrays */
clear_array(vidray2,mapchars);
clear_array(isar1,mapchars);
clear_array(isar2,mapchars);
clear_array(Image1,2*mapchars);
clear_array(Image2,2*mapchars);
clear_array(BigImage1,2*4*mapchars);
clear_array(BigImage2,2*4*mapchars);
ClearRadarData(); /* clear radar data array */
/*
 *  create toplevel shell for the main menu and an application shell
 *   for graphics widgets
 */
toplevel = XtInitialize(NULL,"X-Radar",
NULL, 0,
NULL, NULL);
top2 = XtCreateApplicationShell("Windows",topLevelShellWidgetClass,
   NULL,0);
XtSetArg(arg[0], XtNx, 0);
XtSetArg(arg[1], XtNy, 0);
XtSetArg(arg[2], XtNwidth, 1024);
XtSetArg(arg[3], XtNheight,1024);
XtSetArg(arg[4], XtNtransient, True);
XtSetArg(arg[5], XtNallowShellResize, False);
XtSetValues(top2,arg,6);
XtRealizeWidget(top2);
/*
 *  Now that top2 is realized, create colormap and a graphic
 *  context based on it.
 */
display = XtDisplay(top2);
cmap = XCreateColormap(display,DefaultRootWindow(display),
DefaultVisual(display, DefaultScreen(display)),
AllocAll);
cmap = XDefaultColormap(display, DefaultScreen(display));
XInstallColormap(display,cmap);
BuildColorMap2();
BuildMonoColormap();
XSetWindowColormap(XtDisplay(top2),XtWindow(top2),cmap);
gc1 = XCreateGC(display,DefaultRootWindow(display),0,0);
fore = XBlackPixel(display,DefaultScreen(display));
back = XWhitePixel(display,DefaultScreen(display));
XSetBackground(display,gc1,fore);
XSetForeground(display,gc1,back);
/* Now, let's define some useful colors */
red = cols[245];
blue = cols[130];
white = back;
black = fore;
/*
 *  create popup shells for graphics widgets
 */
XtSetArg(arg[0], XtNx, gxpos1);
XtSetArg(arg[1], XtNy, gypos1);
XtSetArg(arg[2], XtNallowShellResize, False);
XtSetArg(arg[3], XtNgeometry, "512x512");
shell1 = XtCreatePopupShell("Window1",transientShellWidgetClass,
top2,arg,4);
XtSetArg(arg[0], XtNx, gxpos2);
XtSetArg(arg[1], XtNy, gypos2);
XtSetArg(arg[2], XtNallowShellResize, False);
XtSetArg(arg[3], XtNgeometry, "512x512");
shell2 = XtCreatePopupShell("Window2",transientShellWidgetClass,
top2,arg,4);
XtSetArg(arg[0], XtNx, gxpos3);
XtSetArg(arg[1], XtNy, gypos3);
XtSetArg(arg[2], XtNallowShellResize, False);
XtSetArg(arg[3], XtNgeometry, "1024x512");
shell3 = XtCreatePopupShell("Window3",transientShellWidgetClass,
top2,arg,4);
XtSetArg(arg[0], XtNx, gxpos4);
XtSetArg(arg[1], XtNy, gypos4);
XtSetArg(arg[2], XtNallowShellResize, False);
XtSetArg(arg[3], XtNgeometry, "1024x1024");
shell4 = XtCreatePopupShell("Window4",transientShellWidgetClass,
top2,arg,4);
/*
 *  create core widgets to go in the popup shells.
 *  Note that foreground and backgound colors are reversed so
 *  that the normal state of the windows is black.
 */
/* window1 and window2 are 512 x 512 */
XtSetArg(arg[0], XtNx, 0);
XtSetArg(arg[1], XtNy, 0);
XtSetArg(arg[2], XtNwidth, xsize);
XtSetArg(arg[3], XtNheight, ysize);
XtSetArg(arg[4], XtNbackground, fore);
XtSetArg(arg[5], XtNforeground, back);
winwidget1 = XtCreateManagedWidget("Window1",widgetClass,
shell1,arg,6);
XtAddEventHandler(winwidget1, ExposureMask, FALSE, RePaint,NULL );
XtSetArg(arg[0], XtNx, 0);
XtSetArg(arg[1], XtNy, 0);
XtSetArg(arg[2], XtNwidth, xsize);
XtSetArg(arg[3], XtNheight, ysize);
XtSetArg(arg[4], XtNbackground, fore);
XtSetArg(arg[5], XtNforeground, back);
winwidget2 = XtCreateManagedWidget("Window2",widgetClass,
shell2,arg,6);
XtAddEventHandler(winwidget2, ExposureMask, FALSE, RePaint, NULL);
/* window3 is 1024 x 512 */
XtSetArg(arg[0], XtNx, 0);
XtSetArg(arg[1], XtNy, 0);
XtSetArg(arg[2], XtNwidth, 2*xsize);
XtSetArg(arg[3], XtNheight, ysize);
XtSetArg(arg[4], XtNbackground, fore);
XtSetArg(arg[5], XtNforeground, back);
winwidget3 = XtCreateManagedWidget("Window3",widgetClass,
shell3,arg,6);
XtAddEventHandler(winwidget3, ExposureMask, FALSE, RePaint, NULL );
/* window4 is 1024 x 1024 */
XtSetArg(arg[0], XtNx, 0);
XtSetArg(arg[1], XtNy, 0);
XtSetArg(arg[2], XtNwidth, 2*xsize);
XtSetArg(arg[3], XtNheight, 2*ysize);
XtSetArg(arg[4], XtNbackground, fore);
XtSetArg(arg[5], XtNforeground, back );
winwidget4 = XtCreateManagedWidget("Window4",widgetClass,
shell4,arg,6);
XtAddEventHandler(winwidget4, ExposureMask, FALSE,
  RePaint, NULL);
/*
 * allocate structures to hold images ;
 *  im1, im2 for multi-screen display;
 *  im3 and im4 for single-screen display.
 *
 */
depth = XDefaultDepth(display,DefaultScreen(display));
bitmap_pad = 8;
im1 = XCreateImage(display, DefaultVisual(display,
DefaultScreen(display)),
depth, ZPixmap, 0, Image1, xsize,
ysize, bitmap_pad, xsize*2 );
if(im1==0)
{
printf("Allocation of structure for image #1 failed.\n");
exit(0);
}
im2 = XCreateImage(display, DefaultVisual(display,
DefaultScreen(display)),
depth, ZPixmap, 0, Image2, xsize,
ysize, bitmap_pad, xsize*2);
if(im2==0)
{
printf("Allocation of structure for image #2 failed.\n");
exit(0);
}
im3 = XCreateImage(display, DefaultVisual(display,
DefaultScreen(display)),
depth, ZPixmap, 0, BigImage1, 2*xsize,
2*ysize, bitmap_pad, xsize*4);
if(im3==0)
{
printf("Allocation of structure for image #3 failed.\n");
exit(0);
```

```
}
im4 = XCreateImage(display, DefaultVisual(display,
DefaultScreen(display)),
depth, ZPixmap, 0, BigImage2, 2*xsize,
2*ysize, bitmap_pad, xsize*4);
if(im4==0)
{
printf("Allocation of structure for image #4 failed.\n");
exit(0);
}
ITI_init();   /* init. video board */
}  /*end initialize()*/
/***************************************************************************/
BuildColorMap2()
{
int numcolors;
XColor defs[MAXCOLORS];
int i,j;
numcolors = XDisplayCells(display,DefaultScreen(display));
for( i=0; i<MAXCOLORS/2; i++)
{
defs[i].red =
defs[i].green =
defs[i].blue = (i+1)*512 - 1;
defs[i].flags = DoRed | DoGreen | DoBlue;
if( !XAllocColor(display,cmap,&defs[i]))
{
printf("Unable to allocate color #%d\n",i);
exit(-1);   /* exit */
}
cols[i] = defs[i].pixel;
}
for(i=MAXCOLORS/2+1;i<170;i++)
{
defs[i].red = 0;
defs[i].green = 0;
defs[i].blue = (i+85)*512;
defs[i].flags = DoRed | DoGreen | DoBlue;
if(!XAllocColor(display,cmap,&defs[i]))
{
printf("Unable to allocate color #%d\n",i);
exit(-1);
}
cols[i] = defs[i].pixel;
}
for(i=170;i<212;i++)
{
defs[i].red = 0;
defs[i].green = (i+43)*512;
defs[i].blue = 0;
defs[i].flags = DoRed | DoGreen | DoBlue;
```

```
if(!XAllocColor(display,cmap,&defs[i]))
{
printf("Unable to allocate color #%d\n",i);
exit(-1);
}
cols[i] = defs[i].pixel;
}
for(i=212;i<256;i++)
{
defs[i].red = i*512;
defs[i].green = 0;
defs[i].blue = 0;
defs[i].flags = DoRed | DoGreen | DoBlue;
if(!XAllocColor(display,cmap,&defs[i]))
{
printf("Unable to allocate color #%d\n",i);
exit(-1);
}
cols[i] = defs[i].pixel;
}
}
/*
 * function to build monochrome color map for images
 */
BuildMonoColormap()
{
int numcolors;
XColor defs[MAXCOLORS];
int i,j;
numcolors = XDisplayCells(display,DefaultScreen(display));
for( i=0; i<MAXCOLORS/2; i++)
{
defs[i].red =
defs[i].green =
defs[i].blue = (i+1)*512;
defs[i].flags = DoRed | DoGreen | DoBlue;
if( !XAllocColor(display,cmap,&defs[i]))
{
printf("Unable to allocate color #%d\n",i);
exit(-1);   /* exit */
}
monocols[i] = defs[i].pixel;
}
for( i=MAXCOLORS/2+1; i<MAXCOLORS; i++)
{
defs[i].red =
defs[i].green =
defs[i].blue = 512*(i-127) - 1;
defs[i].flags = DoRed | DoGreen | DoBlue;
if( !XAllocColor(display,cmap,&defs[i]))
{
```

```
printf("Unable to allocate color #%d\n",i);
exit(-1);   /* exit */
}
monocols[i] = defs[i].pixel;
}
}
```

B.2.2 Module xinterface.c

```
/*    xinterface.c
 *    File contains source code for initializing the widgets and defining
 *    callback functions for the X-RADAR program.
 *
 */
#include "radar.h" /* inlcudes all X includes */
Widget create_mag_scale();
/* external function declarations */
extern void exit();
extern void pop_down();
extern void multi_screen();
extern void single_screen();
extern Widget CreateTextWidget();
extern Widget CreateErrorWidget();
extern Widget CreateMsgWidget();
extern Widget CreateScaleWidget();
extern void BuildColorMap();
extern int read_video_file();
extern int read_radar_file();
extern void create_rubber_gc();
extern void start_rubber_band();
extern void track_rubber_band();
extern void end_rubber_band();
extern void init_cursor();
extern void track_cursor();
extern void end_cursor();
extern void init_line();
extern void track_line();
extern void end_line();
extern void ITI_cont();
extern void ITI_frame();
extern void ITI_camera();
extern void image_loop();
/* external global variable declarations */
extern XImage           *im1, *im2, *im3, *im4;
extern Display          *display;
extern Colormap         cmap;
extern GC          gc1;
extern XFontStruct  *font,*font2;
extern Widget toplevel, top2, shell1,
shell2, shell3,
shell4,
```

C-2

```
 winwidget1, winwidget2,
 winwidget3, winwidget4,
 popfile1,popfile1a,popfile2,
 popfile3,popmove,
 scalepop1, scalepop2,
 box,error_popup, exit_message,
 image_message,
 button1,button2,
 button3,button4,
 button5,button6,
 button7,button8,
 button9,button10,
 button11,button12,
 button13,button14,
 button15,button16,
 button17,
 color_scale, mono_scale;
extern float          radar_data[IMAGE_PNTS][6];
extern int            gxpos1,gxpos2,gxpos3,gxpos4;
extern int            gypos1,gypos2,gypos3,gypos4;
extern int            xsize,ysize;
extern long           cols[];
extern long       monocols[];
extern int            fore,back;
extern unsigned int   red,white,blue,black;
extern int            scrn1_flag, scrn2_flag;
extern Boolean       vid1_flag,vid2_flag;
extern Boolean        isar_flag;
extern Boolean       mono;
extern                rubber_band_data rb_data;
extern      cursor_data cur_data;
extern      line_data l_data;
extern widget_struct w_rec1, w_rec2;
extern char           xscale_str[10],yscale1_str[10],
                      yscale2_str[10],zscale_str[10];
extern float        , x_scale,y1_scale,y2_scale,z_scale;
extern float        xy_angle, yz_angle;
extern int            x_dim, y1_dim, y2_dim, z_dim;
extern int            x_offset,y_offset,z_offset;
extern char           buffer1[80], buffer1a[80], buffer2[80];
extern FILE           *fd_mapfile;
extern char           *vidray1, *vidray2, *isar1, *isar2, *Image1, *Image2;
extern char           *BigImage1, *BigImage2;
/* CALLBACK FUNCTIONS */
/*********************************************************************************/
void VideoFileIn(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
char *file_name;
char top_file[80], side_file[80];
```

```
file_name = (char *)client_data; /* get base file name */
strcpy(top_file,file_name); /* save to array */
strcpy(side_file,file_name);
strcat(top_file,"_top"); /* append view */
strcat(side_file,"_side"); /* to file name */
if(read_video_file(top_file,vidray1)) /* read top view */
{
vid1_flag = True;
CreateImage(vidray1,im1,xsize,ysize);
CreateImage(vidray1,im3,2*xsize,2*ysize);
if(scrn1_flag)    /* in single screen mode */
{
   XPutImage(XtDisplay(winwidget4),XtWindow(winwidget4),
  gc1,im3,0,0,0,0,2*xsize,2*ysize);
   XFlush(XtDisplay(winwidget4));
}
else if(!scrn2_flag)      /* in multi screen mode */
{
   XPutImage(XtDisplay(winwidget1),XtWindow(winwidget1),
  gc1,im1,0,0,0,0,xsize,ysize);
   XFlush(XtDisplay(winwidget1));
}
}
if(read_video_file(side_file,vidray2)) /* read side view */
{
vid2_flag = True;
CreateImage(vidray2,im2,xsize,ysize);
CreateImage(vidray2,im4,2*xsize,2*ysize);
if(scrn2_flag)   /* in single screen mode showing #2 */
{
   XPutImage(XtDisplay(winwidget4),XtWindow(winwidget4),
 gc1,im4,0,0,0,0,2*xsize,2*ysize);
   XFlush(XtDisplay(winwidget4));
}
else if(!scrn1_flag)       /* in multi screen mode */
{
   XPutImage(XtDisplay(winwidget2),XtWindow(winwidget2),
  gc1,im2,0,0,0,0,xsize,ysize);
   XFlush(XtDisplay(winwidget2));
}
}
}
/******************************************************************************/
void VideoFileOut(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
char *file_name;
char top_file[80], side_file[80];
file_name = (char *)client_data;
/*
```

```
  * build filenames for both views
  */
 strcpy(top_file,file_name);
 strcpy(side_file,file_name);
 strcat(top_file,"_top");
 strcat(side_file,"_side");
 /*
  * decide if two views exist and write the ones that do
  */
 if(vid1_flag)
    write_video_file(top_file,vidray1);
 if(vid2_flag)
    write_video_file(side_file,vidray2);
 }
/*******************************************************************************/
void RadarFileIn(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
char *file_name;
isar_flag = False;
file_name = (char *)client_data;
if(read_radar_file(file_name,radar_data))
{
AddRadarImage(im1);
AddRadarImage(im2);
AddRadarImage(im3);
AddRadarImage(im4);
}
}
/*******************************************************************************/
void IsarFileIn(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
char *filename;
isar_flag = True;
filename = (char *)client_data;
if(read_video_file(filename,isar1))
AddIsarImage(vidray1,isar1,xsize,ysize);
}
/*******************************************************************************/
void switch_screen( w, closure, call_data )
    Widget w;
    caddr_t closure, call_data;
{
static Arg args[10];
if(!scrn1_flag && !scrn2_flag)   /* in multi screen mode */
{
if(w == button1)  /* screen 1 to full screen */
{
```

```
XtSetArg(args[0],XtNlabel,"   MULTI - SCREEN MODE  ");
XtSetValues(w,args,1);
single_screen(im3);
AddRadarImage(im3);
scrn1_flag = TRUE;
}
if(w == button2) /* screen 2 to full screen */
{
XtSetArg(args[0],XtNlabel,"   MULTI - SCREEN MODE  ");
XtSetValues(w,args,1);
single_screen(im4);
AddRadarImage(im4);
scrn2_flag = TRUE;
}
}
else if(scrn1_flag)   /* screen1 is full screen */
{
if(w == button1)   /* going to multiscreen mode */
{
XtSetArg(args[0],XtNlabel," WINDOW1 -> FULL SCREEN ");
XtSetValues(w,args,1);
multi_screen();
AddRadarImage(im1);
AddRadarImage(im2);
scrn2_flag = FALSE;
}
if(w == button2)   /* going to screen2 full screen */
{
XtSetArg(args[0],XtNlabel,"   MULTI - SCREEN MODE  ");
XtSetValues(w,args,1);
XtSetArg(args[0],XtNlabel," WINDOW1 -> FULL SCREEN ");
XtSetValues(button1,args,1);
single_screen(im4);
AddRadarImage(im4);
scrn2_flag = TRUE;
}
scrn1_flag = FALSE;
}
else if(scrn2_flag)   /* screen2 is full screen */
{
if(w == button1)  /* going to screen1 to full screen */
{
XtSetArg(args[0],XtNlabel,"   MULTI - SCREEN MODE  ");
XtSetValues(w,args,1);
XtSetArg(args[0],XtNlabel," WINDOW2 -> FULL SCREEN ");
XtSetValues(button2,args,1);
single_screen(im3);
AddRadarImage(im3);
scrn1_flag = TRUE;
}
if(w == button2)    /* going to multi-screen mode */
```

```
{
XtSetArg(args[0],XtNlabel," WINDOW2 -> FULL SCREEN ");
XtSetValues(w,args,1);
multi_screen();
AddRadarImage(im1);
AddRadarImage(im2);
scrn1_flag = FALSE;
}
scrn2_flag = FALSE;
}
}
/*****************************************************************************/
void switch_maps(w,client_data,call_data)
Widget w;
caddr_t client_data,call_data;
{
static Arg args[10];
if(mono) /* change flag and menu button */
{
mono = False;
XtSetArg(args[0],XtNlabel,"   SWITCH TO MONOCHROME   ");
XtSetArg(args[1],XtNwidth,245);
XtSetValues(w,args,2);
XtPopdown(mono_scale);
XtPopup(color_scale,XtGrabNone);
}
else
{
mono = True;
XtSetArg(args[0],XtNwidth,245);
XtSetArg(args[1],XtNlabel,"    SWITCH  TO  COLOR    ");
XtSetValues(w,args,2);
XtPopdown(color_scale);
XtPopup(mono_scale,XtGrabNone);
}
XSync(display,0);
/*
 * Now, build and display new images
 */
if(!scrn1_flag && !scrn2_flag)    /* in single screen mode */
                WriteMessage("Please wait, this will take awhile...",
                                XtWindow(winwidget1),5,505,white);
else
                WriteMessage("Please wait, this will take awhile...",
                                XtWindow(winwidget4),300,1020,white);
CreateImage(vidray1,im1,xsize,ysize);
CreateImage(vidray1,im3,2*xsize,2*ysize);
CreateImage(vidray2,im2,xsize,ysize);
CreateImage(vidray2,im4,2*xsize,2*ysize);
if(!scrn1_flag && !scrn2_flag)   /* in multi-screen mode */
{
    SubRadarImage(im1); /* clear top window */
    XPutImage(XtDisplay(winwidget1),XtWindow(winwidget1),
   gc1,im1,0,0,0,0,xsize,ysize);
    AddRadarImage(im1); /* add radar image to new image */
    SubRadarImage(im2);  /* clear other window */
    XPutImage(XtDisplay(winwidget2),XtWindow(winwidget2),
   gc1,im2,0,0,0,0,xsize,ysize);
    AddRadarImage(im2);
    XFlush(XtDisplay(winwidget2));
}
if(scrn1_flag)    /* in single screen mode */
{
    SubRadarImage(im3);
    XPutImage(XtDisplay(winwidget4),XtWindow(winwidget4),
   gc1,im3,0,0,0,0,2*xsize,2*ysize);
    AddRadarImage(im3);
    XFlush(XtDisplay(winwidget4));
}
if(scrn2_flag)
{
    SubRadarImage(im4);
    XPutImage(display,XtWindow(winwidget4),
  gc1,im4,0,0,0,0,2*xsize,2*ysize);
    AddRadarImage(im4);
    XFlush(XtDisplay(winwidget4));
}
if(!scrn1_flag && !scrn2_flag)    /* in single screen mode */
               WriteMessage("Please wait, this will take awhile...",
                               XtWindow(winwidget1),5,505,black);
else
               WriteMessage("Please wait, this will take awhile...",
                               XtWindow(winwidget4),300,1020,black);
}
/*******************************************************************************/
void ShiftImage( w, closure, call_data )
    Widget w;
    caddr_t closure, call_data;
{
Time time;
/*
 * can't shift ISAR image
 */
if(isar_flag)
{
Error_window(5,0,NULL);
return;
}
/*
 * must be in single screen mode to set offsets : do error window
 * and return.
 */
```

```
if((!scrn1_flag) && (!scrn2_flag))
{
Error_window(4,0,NULL);
return;
}
  /* set up cursor stuff */
XtAddEventHandler(winwidget4, PointerMotionMask,
False, track_cursor, &cur_data);
XtAddEventHandler(winwidget4, ButtonPressMask,
False, end_cursor, &cur_data);
XGrabPointer(XtDisplay(winwidget4),
XtWindow(winwidget4),True,
PointerMotionMask | ButtonPressMask ,
GrabModeAsync, GrabModeAsync, XtWindow(winwidget4),
XCreateFontCursor(XtDisplay(winwidget4),XC_crosshair),
time);
init_cursor(w,&cur_data);
}
/*********************************************************************************/
void RotateImage(w,client_data,call_data)
Widget w;
caddr_t call_data,client_data;
{
Time time;
/*
 * can't rotate ISAR image
 */
if(isar_flag)
{
Error_window(5,0,NULL);
return;
}
/*
 * must be in single screen mode to do rotation : do error window
 * and return.
 */
if((!scrn1_flag) && (!scrn2_flag))
{
Error_window(6,0,NULL);
return;
}
  /* set up cursor stuff */
XtAddEventHandler(winwidget4, PointerMotionMask,
False, track_line, &l_data);
XtAddEventHandler(winwidget4, ButtonPressMask,
False, end_line, &l_data);
XGrabPointer(XtDisplay(winwidget4),
XtWindow(winwidget4),True,
PointerMotionMask | ButtonPressMask ,
GrabModeAsync, GrabModeAsync, XtWindow(winwidget4),
XCreateFontCursor(XtDisplay(winwidget4),XC_crosshair),
time);
init_line(winwidget4,&l_data);
}
/*****************************************************************************/
void PopupScale1(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
if(isar_flag) /* can't scale isar image */
{
/* pop-up error message and exit function */
Error_window(5,0,NULL);
return;
}
/*
 * check to see if in window #1 single screen mode
 */
if(scrn1_flag)
{
  /* set up rubber band stuff */
XtAddEventHandler(winwidget4, ButtonPressMask,
FALSE, start_rubber_band, &rb_data);
XtAddEventHandler(winwidget4, ButtonMotionMask,
FALSE, track_rubber_band, &rb_data);
XtAddEventHandler(winwidget4, ButtonReleaseMask,
FALSE, end_rubber_band, &rb_data);
XGrabButton(XtDisplay(winwidget4), AnyButton, AnyModifier,
XtWindow(winwidget4), TRUE,
ButtonPressMask | ButtonMotionMask | ButtonReleaseMask,
GrabModeAsync, GrabModeAsync, XtWindow(winwidget4),
XCreateFontCursor(XtDisplay(winwidget4),XC_crosshair));
XtPopup(scalepop1, XtGrabNone);
}
else
   Error_window(3,0,NULL);
}
/****************************************************************************/
void PopupScale2(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
if(isar_flag)
{
/* pop-up error message and exit function */

return;
}
/*
 * check to see if in window #2 single screen mode
 */
if(scrn2_flag)
```

```
{
  /* set up rubber band stuff */
XtAddEventHandler(winwidget4, ButtonPressMask,
FALSE, start_rubber_band, &rb_data);
XtAddEventHandler(winwidget4, ButtonMotionMask,
FALSE, track_rubber_band, &rb_data);
XtAddEventHandler(winwidget4, ButtonReleaseMask,
FALSE, end_rubber_band, &rb_data);
XGrabButton(XtDisplay(winwidget4), AnyButton, AnyModifier,
XtWindow(winwidget4), TRUE,
ButtonPressMask | ButtonMotionMask | ButtonReleaseMask,
GrabModeAsync, GrabModeAsync, XtWindow(winwidget4),
XCreateFontCursor(XtDisplay(winwidget4),XC_crosshair));
XtPopup(scalepop2, XtGrabNone);
}
else
   Error_window(3,0,NULL);
}
/**********************************************************************************/
void SetScale1(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
char **dummy;
char *temp;
/* remove event handlers and button grab */
XtRemoveEventHandler(winwidget4, ButtonPressMask,
FALSE, start_rubber_band, &rb_data);
XtRemoveEventHandler(winwidget4, ButtonReleaseMask,
FALSE, end_rubber_band, &rb_data);
XtRemoveEventHandler(winwidget4, ButtonMotionMask,
FALSE, track_rubber_band, &rb_data);
XUngrabButton(XtDisplay(winwidget4), AnyButton,AnyModifier,
XtWindow(winwidget4));
x_scale = (float)(x_dim)/2./atof(xscale_str);
y1_scale = (float)(y1_dim)/2./atof(yscale1_str);
}
/**********************************************************************************/
void SetScale2(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
/* remove event handlers and button grab */
XtRemoveEventHandler(winwidget4, ButtonPressMask,
FALSE, start_rubber_band, &rb_data);
XtRemoveEventHandler(winwidget4, ButtonReleaseMask,
FALSE, end_rubber_band, &rb_data);
XtRemoveEventHandler(winwidget4, ButtonMotionMask,
FALSE, track_rubber_band, &rb_data);
XUngrabButton(XtDisplay(winwidget4), AnyButton,AnyModifier,
XtWindow(winwidget4));
```

```
y2_scale = (float)(y2_dim)/2./atof(yscale2_str);
z_scale = (float)(z_dim)/2./atof(zscale_str);
}
/*****************************************************************************/
void RadarFilePopup(w, closure, call_data )
    Widget w;
    caddr_t closure, call_data;
{
  XtPopup(popfile2,XtGrabNone);
}

void VideoFilePopup(w, closure, call_data )
    Widget w;
    caddr_t closure, call_data;
{
XtPopup(popfile1,XtGrabNone);
}

void VideoFilePopup1a(w, client_data, call_data)
     Widget w;
     caddr_t client_data, call_data;
{
XtPopup(popfile1a,XtGrabNone);
}

void IsarFilePopup(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
XtPopup(popfile3,XtGrabNone);
}
/*****************************************************************************/
/* video board callbacks  */
void vid_cont(w, client_data, call_data)
    Widget w;
    caddr_t client_data, call_data;
{
ITI_cont();
}
void vid_single(w, client_data, call_data)
    Widget w;
    caddr_t client_data, call_data;
{
unsigned char *camera;
camera = (unsigned char *)client_data;
ITI_frame(*camera);
}
void switch_camera(w, client_data, call_data)
    Widget w;
    caddr_t client_data, call_data;
{
```

```
static char label_string[30];
static Arg arg[] = { {XtNlabel, (XtArgVal)label_string} };
unsigned char *camera;
camera = (unsigned char *)client_data;
*camera = ~(*camera);          /* alternative is camera #1 (side) */
if(*camera)
{
  sprintf(label_string, " SWITCH TO OVERHEAD CAM ");
  XtSetValues(w,arg,1);
}
else
{
  sprintf(label_string, " SWITCH TO SIDEVIEW CAM ");
  XtSetValues(w,arg,1);
}
ITI_camera(*camera);
}
/*****************************************************************************/
void Bye( w, closure, call_data )
    Widget w;
    caddr_t closure, call_data;
{
printf("Bye\n");
ITI_close();                   /* close VME allocations */
XtDestroyWidget(toplevel);     /* destroy all widgets */
XtDestroyWidget(top2);
XFreeColormap(XtDisplay(XtParent(w)),cmap); /* free colormap alloc. */
fflush(stdout);                /* clean I/O channel */
exit(0);
}

/*****************************************************************************/
/* setup main user interface with Widgets */
void xinteract()
{
    Arg  args[10];
    Cardinal i;
    char name[100];
    static unsigned char cam = 0;        /* init. to overhead camera */
    static XtCallbackRec callback[2];
    static XtPopdownIDRec  pdrec1,pdrec2;
    font=XLoadQueryFont(display,"PellucidaSerif14B");
    font2=XLoadQueryFont(display,"PellucidaSerif12B");
    /*
     *  Setup arguments for toplevel shell for menu
     */
    XtSetArg(args[1], XtNwidth, 256);
    XtSetArg(args[2], XtNheight, 1024);
    XtSetArg(args[3], XtNtransient, TRUE);
    XtSetArg(args[4], XtNx, 1024);
    XtSetArg(args[5], XtNy, 0);
    XtSetArg(args[6], XtNforeground,fore);
    XtSetArg(args[7], XtNbackground,back);
    XtSetValues(toplevel,args,8);
    /*
     *  use same arguments for box for menu
     */
    box = XtCreateManagedWidget ("BOX", boxWidgetClass,
       toplevel, args, EIGHT);
    /*
     *   create menu items to go in box
     */
    XtSetArg(args[0], XtNfont, font);
    XtSetArg(args[1], XtNforeground,white);
    XtSetArg(args[2], XtNbackground,blue);
    XtSetArg(args[3], XtNwidth,245);
    XtSetArg(args[4], XtNheight,40);
    XtCreateManagedWidget( "OSU/NASA   XRADAR", labelWidgetClass, box, args, 5);
    XtSetArg(args[0], XtNfont, font2);
    XtSetArg(args[1], XtNforeground,white);
    XtSetArg(args[2], XtNbackground,blue);
    XtSetArg(args[3],XtNwidth,245);
    XtCreateManagedWidget( " FILE   I/O ",
                            labelWidgetClass, box, args, 4);
    XtSetArg(args[0], XtNforeground,blue);
    XtSetArg(args[1], XtNbackground,white);
    XtSetArg(args[2],XtNwidth,245);
    button6 = XtCreateManagedWidget(" READ VIDEO IMAGE FILE ",
   commandWidgetClass, box, args, 3);
    XtSetArg(args[0], XtNforeground,blue);
    XtSetArg(args[1], XtNbackground,white);
    XtSetArg(args[2],XtNwidth,245);
    button9 = XtCreateManagedWidget(" WRITE VIDEO IMAGE FILE ",
commandWidgetClass, box, args, 3);
    XtSetArg(args[0], XtNforeground,blue);
    XtSetArg(args[1], XtNbackground,white);
    XtSetArg(args[2],XtNwidth,245);
    button7 = XtCreateManagedWidget(" READ RADAR IMAGE FILE ",
   commandWidgetClass, box, args, 3);
    XtSetArg(args[0], XtNforeground,blue);
    XtSetArg(args[1], XtNbackground,white);
    XtSetArg(args[2], XtNwidth,245);
    button16 = XtCreateManagedWidget("READ ISAR IMAGE FILE",
   commandWidgetClass,box,args,3);
    XtSetArg(args[0], XtNforeground,blue);
    XtSetArg(args[1], XtNbackground,white);
    XtSetArg(args[2], XtNwidth, 245);
    button15 = XtCreateManagedWidget("REAL TIME DP",
   commandWidgetClass, box, args,3);
    XtAddCallback(button15,XtNcallback,image_loop,NULL);
    XtSetArg(args[0], XtNfont, font2);
    XtSetArg(args[1], XtNforeground,white);
    XtSetArg(args[2], XtNbackground,blue);
    XtSetArg(args[3],XtNwidth,245);
    XtCreateManagedWidget( " VIDEO IMAGE CONTROL ",
labelWidgetClass, box, args, 4);
    callback[0].callback = switch_screen;
    XtSetArg(args[0], XtNcallback, callback );
    XtSetArg(args[1], XtNforeground,blue);
    XtSetArg(args[2], XtNbackground,white);
    XtSetArg(args[3],XtNwidth,245);
    button1 = XtCreateManagedWidget(" WINDOW1 -> FULL SCREEN ",
      commandWidgetClass, box, args, 4);
    callback[0].callback = switch_screen;
    XtSetArg(args[0], XtNcallback, callback );
    XtSetArg(args[1], XtNforeground,blue);
    XtSetArg(args[2], XtNbackground,white);
    XtSetArg(args[3],XtNwidth,245);
    button2 = XtCreateManagedWidget(" WINDOW2 -> FULL SCREEN ",
      commandWidgetClass, box, args, 4);
    callback[0].callback = switch_maps;
    XtSetArg(args[0], XtNcallback, callback);
    XtSetArg(args[1], XtNforeground, blue);
    XtSetArg(args[2], XtNbackground, white);
    XtSetArg(args[3], XtNwidth,245);
    button17 = XtCreateManagedWidget(" SWITCH TO MONO  MAP ",
commandWidgetClass,box,args,4);
    XtSetArg(args[0], XtNfont, font2);
    XtSetArg(args[1], XtNforeground, white);
    XtSetArg(args[2], XtNbackground, blue);
    XtSetArg(args[3],XtNwidth,245);
    XtCreateManagedWidget( " VIDEO  BOARD  CONTROL ",
labelWidgetClass, box, args, 4);
    XtSetArg(args[0], XtNforeground, blue);
    XtSetArg(args[1], XtNbackground, white);
    XtSetArg(args[2],XtNwidth,245);
    button11 = XtCreateManagedWidget(" LIVE VIDEO  TO MONITOR ",
   commandWidgetClass, box, args, 3);
    XtAddCallback(button11,XtNcallback,vid_cont,NULL);
    XtSetArg(args[0], XtNforeground, blue);
    XtSetArg(args[1], XtNbackground, white);
    XtSetArg(args[2],XtNwidth,245);
    button12 = XtCreateManagedWidget(" FRAME GRAB TO COMPUTER ",
   commandWidgetClass, box, args, 3);
    XtAddCallback(button12,XtNcallback,vid_single,&cam);
    XtSetArg(args[0], XtNforeground, blue);
    XtSetArg(args[1], XtNbackground, white);
    XtSetArg(args[2],XtNwidth,245);
    button13 = XtCreateManagedWidget( " SWITCH TO SIDEVIEW CAM ",
   commandWidgetClass, box, args, 3);
    XtAddCallback(button13,XtNcallback,switch_camera,&cam);
    XtSetArg(args[0], XtNfont, font2);
    XtSetArg(args[1], XtNforeground, white);
    XtSetArg(args[2], XtNbackground, blue);
    XtSetArg(args[3],XtNwidth,245);
    XtCreateManagedWidget( " RADAR IMAGE CONTROL ",
labelWidgetClass, box, args, 4);
    callback[0].callback = ShiftImage;
    XtSetArg( args[0], XtNcallback, callback );
    XtSetArg(args[1], XtNforeground, blue);
    XtSetArg(args[2], XtNbackground, white);
    XtSetArg(args[3],XtNwidth,245);
    button8 = XtCreateManagedWidget(" SHIFT RADAR IMAGE ",
   commandWidgetClass, box, args,4);
    callback[0].callback = RotateImage;
    XtSetArg( args[0], XtNcallback, callback );
    XtSetArg(args[1], XtNforeground, blue);
    XtSetArg(args[2], XtNbackground, white);
    XtSetArg(args[3],XtNwidth,245);
    button8 = XtCreateManagedWidget(" ROTATE RADAR IMAGE ",
   commandWidgetClass, box, args,4);
    XtSetArg(args[0], XtNforeground, blue);
    XtSetArg(args[1], XtNbackground, white);
    XtSetArg(args[2],XtNwidth,245);
    button3 = XtCreateManagedWidget(" WINDOW1 SCALE FACTORS ",
   commandWidgetClass, box, args, 3);
    XtSetArg(args[0], XtNforeground, blue);
    XtSetArg(args[1], XtNbackground, white);
    XtSetArg(args[2],XtNwidth,245);
    button4 = XtCreateManagedWidget(" WINDOW2 SCALE FACTORS ",
   commandWidgetClass, box, args, 3);
    XtSetArg(args[0], XtNfont, font2);
    XtSetArg(args[1], XtNforeground, white);
    XtSetArg(args[2], XtNbackground, blue);
    XtSetArg(args[3],XtNwidth,245);
    XtCreateManagedWidget( " SYSTEM CALLS ",
labelWidgetClass, box, args, 4);
    callback[0].callback = Bye;
    XtSetArg( args[0], XtNcallback, callback );
    XtSetArg(args[1], XtNforeground, blue);
    XtSetArg(args[2], XtNbackground, white);
    XtSetArg(args[3],XtNwidth,245);
    button8 = XtCreateManagedWidget(" EXIT ",
   commandWidgetClass, box, args,4);
    /*
     * create 2 popup magnitude color bars, one for color, the other for mono
     */
    color_scale = create_mag_scale(toplevel,cols);
    mono_scale  = create_mag_scale(toplevel,monocols);
    /*
     *  create popup text windows for file name entry
     */
    popfile1 = CreateTextWidget(toplevel,"Enter video image filename :",
                                VideoFileIn,buffer1,512,800);
    popfile1a = CreateTextWidget(toplevel,"Enter filename for video image :",
       VideoFileOut,buffer1a,512,800);
    popfile2 = CreateTextWidget(toplevel,"Enter radar image filename :",
RadarFileIn,buffer2,512,800);
    popfile3 = CreateTextWidget(toplevel,"Enter ISAR image filename :",
IsarFileIn,buffer2,512,800);
    /*
     * add callbacks to appropriate buttons to activate popups;
     *  this works better than installing at creation.
     */
    XtAddCallback(button6,XtNcallback,VideoFilePopup,NULL);
    XtAddCallback(button7,XtNcallback,RadarFilePopup,NULL);
    XtAddCallback(button9,XtNcallback,VideoFilePopup1a,NULL);
    XtAddCallback(button16,XtNcallback,IsarFilePopup,NULL);
    /*
     * create a popup shell for displaying error messages
     */
     error_popup = CreateErrorWidget(toplevel);
    /*
     *  create two popups for scaling the radar image and register
     *  callbcks to pop them up.
     */
    scalepop1 = CreateScaleWidget(toplevel,
" ---- SCALE FACTORS FOR WINDOW #1 ---- ",
"Set horizontal dimension (inches) ",
"Set  vertical  dimension (inches) ",
SetScale1,xscale_str,yscale1_str,
&w_rec1,0,850);
    scalepop2 = CreateScaleWidget(toplevel,
" ---- SCALE FACTORS FOR WINDOW #2 ---- ",
"Set horizontal dimension (inches) ",
"Set  vertical  dimension (inches) ",
SetScale2,yscale2_str,zscale_str,
&w_rec2,512,850);
      XtAddCallback(button3, XtNcallback,PopupScale1,NULL);
      XtAddCallback(button4, XtNcallback,PopupScale2,NULL);
    /*
     * set up gc for rubber band lines
     */
    create_rubber_gc(winwidget4,&rb_data);
    /*
     * realize toplevel widget and all of its children
     */
    XtRealizeWidget(toplevel);
    XSetWindowColormap(XtDisplay(toplevel),XtWindow(toplevel),cmap);
    XtPopup(shell1, XtGrabNone);
    XSetWindowColormap(XtDisplay(shell1),XtWindow(shell1),cmap);
    XtPopup(shell2, XtGrabNone);
    XSetWindowColormap(XtDisplay(shell2),XtWindow(shell2),cmap);
    XtPopup(shell3, XtGrabNone);
    XSetWindowColormap(XtDisplay(shell3),XtWindow(shell3),cmap);
    XtPopup(color_scale,XtGrabNone);
}
/*********************************************************************/
Widget create_mag_scale(parent,colormap)
Widget parent;
unsigned long *colormap;
{
Arg args[10];
Widget popup,box;
char geom_str[20];
int x_size,y_size;
XtTranslations trans_table;
/*
 * create a popup shell
 */
x_size = 230;
y_size = 300;
sprintf(geom_str,"%1ix%1i",x_size,y_size);
XtSetArg(args[0], XtNy, 675);
XtSetArg(args[1], XtNx, 1030);
XtSetArg(args[2], XtNgeometry,geom_str);
XtSetArg(args[3], XtNallowShellResize,False);
popup = XtCreatePopupShell("popup",transientShellWidgetClass,
parent,args,4);
XtSetArg(args[0],XtNheight,300);
XtSetArg(args[1],XtNwidth,230);
XtSetArg(args[2],XtNforeground,fore);
XtSetArg(args[3],XtNbackground,back);
box = XtCreateManagedWidget("box",boxWidgetClass,popup,args,4);
/* create a simple color bar for magnitude scale */
XtSetArg(args[0],XtNbackground,colormap[255]);
XtSetArg(args[1],XtNforeground,white);
XtSetArg(args[2],XtNfont,font2);
XtSetArg(args[3],XtNheight,50);
XtSetArg(args[4],XtNwidth,230);
XtCreateManagedWidget(" 0 dB ",labelWidgetClass,
box,args,5);
XtSetArg(args[0],XtNbackground,colormap[230]);
XtCreateManagedWidget(" -5 dB ",labelWidgetClass,
box,args,5);
XtSetArg(args[0],XtNbackground,colormap[205]);
XtCreateManagedWidget(" -10 dB ",labelWidgetClass,
box,args,5);
XtSetArg(args[0],XtNbackground,colormap[180]);
XtCreateManagedWidget(" -15 dB ",labelWidgetClass,
box,args,5);
XtSetArg(args[0],XtNbackground,colormap[155]);
XtCreateManagedWidget(" -20 dB ",labelWidgetClass,
box,args,5);
XtSetArg(args[0],XtNbackground,colormap[130]);
XtCreateManagedWidget(" -25 dB ",labelWidgetClass,
box,args,5);
return(popup);
}
```

B.2.3 Module ximage.c

```
/*  ximage.c
 *
 *  file of routines for manipulating video and radar images
 */
#include "radar.h" /* includes all X includes */
/*** Function prototypes for this module ***/
void start_rubber_band();
void end_rubber_band();
void track_rubber_band();
void init_cursor();
void track_cursor();
void end_cursor();
void init_line();
void track_line();
void end_line();
void multi_screen();
int  AddRadarImage();
int  SubRadarImage();
int  ClearRadarData();
int  single_screen();
extern double square();
extern int rndf();
extern rotate_point();
extern XImage *im1,*im2,*im3,*im4;
extern Display *display;
extern Colormap cmap;
extern GC gc1,gc2;
extern Widget toplevel, top2,
shell1, shell2,
shell3, shell4,
winwidget1, winwidget2,
winwidget3, winwidget4,
popfile1,popfile2,error_popup,popmove,
image_message,
box,button1,button2,
button3,button4,
button5,button6,
button7,button8,
button9,button10,
button11,button12,
button13,button14;
extern int scrn1_flag,scrn2_flag;  /* flags set if in single screen mode */
extern Boolean mono;    /* flag set if using mono colormap */
extern int num_pnts;    /* # of points in current radar image */
extern float in_angle;    /* incident angle of plane wave */
extern Boolean vid1_flag,vid2_flag;
extern Boolean isar_flag;
extern widget_struct w_rec1, w_rec2;
extern rubber_band_data rb_data;
extern cursor_data cur_data;
extern line_data l_data;
extern unsigned long cols[256];
extern unsigned long monocols[256];
extern char buffer1[80], buffer2[80];
extern int      gxpos1, gypos1,
                gxpos2, gypos2,
                gxpos3, gypos3,
                gxpos4, gypos4,
                xsize, ysize,
                screen;
extern unsigned fore,back,red,white,blue,black;
extern float radar_data[IMAGE_PNTS][6];
extern char *vidray1, *vidray2, *isar1, *isar2;
extern char *Image1, *Image2, *BigImage1, *BigImage2;
extern float x_scale, y1_scale, y2_scale, z_scale;
extern float xy_angle, yz_angle;
extern int   x_dim, y1_dim, y2_dim, z_dim;
extern       horizontal_flag;
extern int   x_offset,y1_offset,y2_offset,z_offset;
/*****************************************************************************/
CreateImage(raw_image,image,xsize,ysize)
char *raw_image;
XImage *image;
int xsize,ysize;
{
int x,y,x2,y2,x21,y21;
int xlimit,ylimit;
unsigned char dat;
unsigned long *colormap;
unsigned long color;
if(mono) colormap = monocols;
else     colormap = cols;
if(xsize == 512) /* doing small image */
{
   for(y=0; y<ysize; y++)
       for(x=0; x<xsize; x++)
       {
    XPutPixel(image,x,y,colormap[*(raw_image + x + y*512)]);
       }
}
else /* doing big image */
{
   xlimit = xsize/2; /* for speed */
   ylimit = ysize/2;
   for(y=0;y<ylimit;y++) /* loop thru raw data */
   {
```

```
y2 = 2*y; /* for speed */
y21 = y2+1;
     for(x=0;x<xlimit;x++) /* dup. pixels into big image */
        {
x2 = 2*x; /* for speed */
x21 = x2+1;
  color = colormap[*(raw_image + x + 512*y)];
  XPutPixel(image,x2,y2,color);
  XPutPixel(image,x21,y2,color);
  XPutPixel(image,x2,y21,color);
  XPutPixel(image,x21,y21,color);
}
   }
}
}
/*****************************************************************************/
AddRadarImage(image)
XImage *image;
{
unsigned int width,height; /* size of image point */
unsigned int px,py; /* screen coords. of point */
unsigned int magt;
int i;
float x,y,z; /* temp. vars. */
float rin_angle;
unsigned long *colormap;
if(mono) colormap = monocols; /* set color map pointer */
else  colormap = cols;
width = height = 6;
rin_angle  = -PI/180.*in_angle;
if( image == im1 )
{
for(i=0;i<num_pnts;i++)
{
   x = radar_data[i][0];
   y = radar_data[i][1];
   rotate_point(&x,&y,xy_angle);
   px = rndf(x_scale*y - width/2. + x_offset + 256);
   py = rndf(y1_scale*x - height/2. + y1_offset + 240);
   magt = rndf(255+80*log10((float)(radar_data[i][3])));
   XSetForeground(display,gc1,colormap[magt]);
   XFillRectangle(display,XtWindow(winwidget1),
gc1,px,py,width,height);
}
if(!isar_flag)
    draw_arrow(white,rin_angle+xy_angle); /* draw arrow */
}
if( image == im2 )
{
for(i=0;i<num_pnts;i++)
{
    y = radar_data[i][1];
    z = radar_data[i][2];
    rotate_point(&x,&y,yz_angle);
    px = rndf(fabs(y2_scale)*y - width/2. + y2_offset + 256);
    py = rndf(-fabs(z_scale)*z - height/2. + z_offset + 240);
    magt = rndf(255+80*log10((float)(radar_data[i][3])));
    XSetForeground(display,gc1,colormap[magt]);
    XFillRectangle(display,XtWindow(winwidget2),
gc1,px,py,width,height);
}
}
if( image == im3 )
{
for(i=0;i<num_pnts;i++)
{
    x = radar_data[i][0];
    y = radar_data[i][1];
    rotate_point(&x,&y,xy_angle);
    px = rndf(2*(fabs(x_scale)*y - width/2. + x_offset + 256));
    py = rndf(2*(fabs(y1_scale)*x - height/2. +
y1_offset + 240));
    magt = rndf(255+80*log10((float)(radar_data[i][3])));
    XSetForeground(display,gc1,colormap[magt]);
    XFillRectangle(display,XtWindow(winwidget4),
gc1,px,py,2*width,2*height);
}
if(!isar_flag)
    draw_arrow(white,rin_angle+xy_angle);
}
if( image == im4 )
{
for(i=0;i<num_pnts;i++)
{
    y = radar_data[i][1];
    z = radar_data[i][2];
    rotate_point(&x,&y,yz_angle);
    px = rndf(2*(fabs(y2_scale)*y - width/2. +
y2_offset + 256));
    py = rndf(2*(-fabs(z_scale)*z - height/2. +
z_offset + 240));
    magt = rndf(255+80*log10((float)(radar_data[i][3])));
    XSetForeground(display,gc1,colormap[magt]);
    XFillRectangle(display,XtWindow(winwidget4),
gc1,px,py,2*width,2*height);
}
}
}
/*******************************************************************************/
ClearRadarData()
{
int i,j;
```

```
for(i=0;i<IMAGE_PNTS;i++)
   for(j=0;j<6;j++)
radar_data[i][j] = 0;
num_pnts = 0;
}
/*****************************************************************************/
SubRadarImage(image)
XImage *image;
{
if(image == im1)
{
    XClearArea(display,XtWindow(winwidget1),0,0,0,0,False);
}
if(image == im2)
{
    XClearArea(display,XtWindow(winwidget2),0,0,0,0,False);
}
if((image == im3) | (image == im4))
{
    XClearArea(display,XtWindow(winwidget4),0,0,0,0,False);
}
}
/*****************************************************************************/
AddIsarImage(video,isar)
char *video, *isar;
{
int x,y,xs,ys;
unsigned char dat;
unsigned long *colormap;
int x2,y2,x21,y21;
if(mono) colormap = monocols;
else colormap = cols;
xs = ys = 512;
        if(!scrn1_flag && !scrn2_flag)    /* in single screen mode */
               WriteMessage("Please wait, this will take awhile...",
                                XtWindow(winwidget1),5,505,white);
        else
               WriteMessage("Please wait, this will take awhile...",
                                XtWindow(winwidget4),300,1020,white);
for( y=0; y<ys; y++ )
for(x=0; x<xs; x++)
{
if(*(isar + x + 512*y))
dat = *(isar + x + 512*y);
else
dat = *(video + x + 512*y);
XPutPixel(im1,x,y,colormap[dat]);
}
/* Now, do big image */
for(y=0;y<ys;y++)         /* loop thru raw data */
{
```

```
y2 = 2*y; /* for speed */
y21 = y2+1;
     for(x=0;x<xs;x++) /* dup. pixels into big image */
        {
if(*(isar + x + 512*y))
       dat = *(isar + x + 512*y);
else
     dat = *(video + x + 512*y);
x2 = 2*x;
x21 = x2+1;
  XPutPixel(im3,x2,y2,colormap[dat]);
  XPutPixel(im3,x21,y2,colormap[dat]);
  XPutPixel(im3,x2,y21,colormap[dat]);
  XPutPixel(im3,x21,y21,colormap[dat]);
}
}
/*
 * Now, need to display appropriate image: im1 or im3
 * and erase message.
 */
if(!scrn1_flag && !scrn2_flag)
{
XPutImage(display,XtWindow(winwidget1),gc1,im1,0,0,0,0,
xs,ys);
                WriteMessage("Please wait, this will take awhile...",
                              XtWindow(winwidget1),5,505,black);
}
else
{
XPutImage(display,XtWindow(winwidget4),gc1,im3,0,0,0,0,
2*xs,2*ys);
                WriteMessage("Please wait, this will take awhile...",
                              XtWindow(winwidget4),300,1020,black);
}
}
/*************************************************************************/
draw_arrow(color,angle)
unsigned long color;
float angle;
{
int re,rp, /* radius of end and head of arrow */
    ra, /* length of arrow head */
    xe,ye, /* coords. of end of arrow */
    xp,yp, /* coords of point of arrow */
            x1,x2, /* x coords of arrow head lines */
            y1,y2; /* y coords of arrow head lines */
float phi; /* angle of arrow head lines with shaft */
re = 239;
rp = 199;
ra = 8;
phi = PI/6.;
```

```
xe = rndf(255 - re * cos(angle));
ye = rndf(239 + re * sin(angle));
xp = rndf(255 - rp * cos(angle));
yp = rndf(239 + rp * sin(angle));
x1 = rndf(xp - ra * cos(angle - phi));
y1 = rndf(yp + ra * sin(angle - phi));
x2 = rndf(xp - ra * cos(angle + phi));
y2 = rndf(yp + ra * sin(angle + phi));
if((!scrn1_flag) && (!scrn2_flag)) /* mult. screens */
{
XSetForeground(display,gc1,color);
XDrawLine(display,XtWindow(winwidget1),
gc1,xe,ye,xp,yp);
XDrawLine(display,XtWindow(winwidget1),
gc1,x1,y1,xp,yp);
XDrawLine(display,XtWindow(winwidget1),
gc1,x2,y2,xp,yp);
XDrawLine(display,XtWindow(winwidget1),
gc1,x1,y1,x2,y2);
}
if(scrn1_flag) /* single screen on top view */
{
XSetForeground(display,gc1,color);
XDrawLine(XtDisplay(winwidget4),XtWindow(winwidget4),
gc1,2*xe,2*ye,2*xp,2*yp);
XDrawLine(XtDisplay(winwidget4),XtWindow(winwidget4),
gc1,2*x1,2*y1,2*xp,2*yp);
XDrawLine(XtDisplay(winwidget4),XtWindow(winwidget4),
gc1,2*x2,2*y2,2*xp,2*yp);
XDrawLine(XtDisplay(winwidget4),XtWindow(winwidget4),
gc1,2*x1,2*y1,2*x2,2*y2);
}
}
/*****************************************************************************/
single_screen(image)
XImage *image;
{
XtPopdown(shell1);
XtPopdown(shell2);
XtPopdown(shell3);
XtPopup(shell4,XtGrabNone);
XClearWindow(XtDisplay(winwidget4),XtWindow(winwidget4));
XPutImage(XtDisplay(winwidget4),XtWindow(winwidget4),gc1,image,0,0,0,0,
2*xsize,2*ysize);
XFlush(display);
}
/*****************************************************************************/
void multi_screen()
{
XtPopdown(shell4);
XtPopup(shell1,XtGrabNone);
XtPopup(shell2,XtGrabNone);
XtPopup(shell3,XtGrabNone);
XFlush(display);
}
/*****************************************************************************/
/*           FUNCTIONS FOR RUBBER BAND LINES , ETC.      */
/*****************************************************************************/
create_rubber_gc(w, data)
Widget w;
rubber_band_data *data;
{
XGCValues values;
Arg arg[2];
XtSetArg(arg[0], XtNbackground, &values.foreground);
XtSetArg(arg[1], XtNforeground, &values.background);
XtGetValues(w, arg,2);
/*
 * set the FG to the XOR of the FG and BG . This creates inverse
 * video.
 */
values.foreground = 250;
values.line_style = LineOnOffDash;
values.function = GXxor;
data->gc = XtGetGC(w, GCForeground | GCBackground |
GCFunction | GCLineStyle, &values);
}
/*****************************************************************************/
void start_rubber_band(w, data, event)
Widget w;
rubber_band_data *data;
XEvent *event;
{
data->last_x = data->start_x = event->xbutton.x;
data->last_y = data->start_y = event->xbutton.y;
XDrawLine(XtDisplay(w), XtWindow(w),
 data->gc, data->start_x,
 data->start_y,data->last_x,data->last_y);
}
/*****************************************************************************/
void track_rubber_band(w, data, event)
Widget w;
rubber_band_data *data;
XEvent *event;
{
XDrawLine(XtDisplay(w), XtWindow(w), data->gc,
  data->start_x, data->start_y,
  data->last_x, data->last_y);
data->last_x = event->xbutton.x;
data->last_y = event->xbutton.y;
XDrawLine(XtDisplay(w), XtWindow(w), data->gc,
  data->start_x, data->start_y,
  data->last_x, data->last_y);
}
/*********************************************************************************/
void end_rubber_band(w, data, event)
Widget w;
rubber_band_data *data;
XEvent *event;
{
Arg arg[1];
int delta_x, delta_y;
double sum;
XDrawLine(XtDisplay(w), XtWindow(w), data->gc,
  data->start_x, data->start_y,
  data->last_x, data->last_y);
data->last_x = event->xbutton.x;
data->last_y = event->xbutton.y;
XFlush(XtDisplay(winwidget4));
delta_x = data->last_x - data->start_x;
delta_y = data->last_y - data->start_y;
sum = square((double)delta_x) + square((double)delta_y);
/* now resensitize the buttons in the widget, except the "done"
 * button , unless really done !
 */
XtSetArg(arg[0], XtNsensitive, True);
if(scrn1_flag)
{
   if(horizontal_flag)
   {
x_dim = rndf(sqrt(sum));
   }
   else
   {
y1_dim = rndf(sqrt(sum));
   }
   XtSetValues(w_rec1.widget1,arg,1);
   XtSetValues(w_rec1.widget2,arg,1);
   XtSetValues(w_rec1.widget3,arg,1);
   XtSetValues(w_rec1.widget4,arg,1);
   XtSetValues(w_rec1.widget5,arg,1);
}
else
{
   if(horizontal_flag)
   {
y2_dim = rndf(sqrt(sum));
   }
   else
   {
z_dim = rndf(sqrt(sum));
   }
   XtSetValues(w_rec2.widget1,arg,1);
   XtSetValues(w_rec2.widget2,arg,1);
   XtSetValues(w_rec2.widget3,arg,1);
   XtSetValues(w_rec2.widget4,arg,1);
   XtSetValues(w_rec2.widget5,arg,1);
}
}
/*****************************************************************************/
/*
 * functions to draw cross hairs  for aligning the
 * radar image with the video image.
 *
 *
 * start full screen cursor from button callback
 */
void init_cursor(w,data)
Widget w;
cursor_data *data;
{
char xmessage[40], ymessage[40];
data->last_x = 511;
data->last_y = 479;
  /* force cursor to center of image */
XWarpPointer(XtDisplay(winwidget4),None,XtWindow(winwidget4),
0,0,0,0,data->last_x,data->last_y);
/* draw initial cursor */
XDrawLine(XtDisplay(winwidget4),XtWindow(winwidget4),rb_data.gc,0,
data->last_y,1023,data->last_y);
XDrawLine(XtDisplay(winwidget4),XtWindow(winwidget4),rb_data.gc,
data->last_x,0,data->last_x,959);
sprintf(ymessage,"Y OFFSET = %+5d ",(data->last_x - 511));
sprintf(xmessage,"X OFFSET = %+5d ",(479 - data->last_y));
WriteMessage(xmessage,XtWindow(winwidget4),500,1020,white);
WriteMessage(ymessage,XtWindow(winwidget4),800,1020,white);
}
/*
 * track cursor within the window
 */
void track_cursor(w,data,event)
Widget w;
cursor_data *data;
XEvent *event;
{
int i,x,y;
char xmessage[40], ymessage[40];
/* erase previous lines */
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,0,
data->last_y,1023,data->last_y);
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,
data->last_x,0,data->last_x,959);
data->last_x = event->xmotion.x;
data->last_y = event->xmotion.y;
```

```
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,0,
data->last_y,1023,data->last_y);
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,
data->last_x,0,data->last_x,959);
sprintf(ymessage,"Y OFFSET = %+5d ",(data->last_x - 511));
sprintf(xmessage,"X OFFSET = %+5d ",(479 - data->last_y));
WriteMessage(xmessage,XtWindow(winwidget4),500,1020,white);
WriteMessage(ymessage,XtWindow(winwidget4),800,1020,white);
}
/*
 * process button event at the end of cursor motion, and set shifts
 */
void end_cursor(w,data,event)
Widget w;
cursor_data *data;
XEvent *event;
{
Time time;
char xmessage[40], ymessage[40];
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,0,
data->last_y,1023,data->last_y);
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,
data->last_x,0,data->last_x,959);
data->last_x = event->xmotion.x;
data->last_y = event->xmotion.y;
/*
 * remove event handlers and ungrb pointer
 */
XtRemoveEventHandler(winwidget4,PointerMotionMask,
False, track_cursor,&cur_data);
XtRemoveEventHandler(winwidget4,ButtonPressMask,
False,end_cursor,&cur_data);
XUngrabPointer(XtDisplay(winwidget4),time);
/*
 * clear message area
 */
XClearArea(XtDisplay(winwidget4),XtWindow(winwidget4),
0,960,0,63,False);
/*
 * set offset values
 */
if(scrn1_flag)
{
x_offset = rndf((data->last_x - 511)/2.);
y1_offset = rndf((data->last_y - 479)/2.);
SubRadarImage(im3);
XPutImage(display,XtWindow(winwidget4),gc1,im3,0,0,0,0,
2*xsize,2*ysize);
AddRadarImage(im3);
}
if(scrn2_flag)
```

```
{
y2_offset = rndf((data->last_x - 511)/2.);
z_offset = rndf((data->last_y - 479)/2.);
SubRadarImage(im4);
XPutImage(display,XtWindow(winwidget4),gc1,im4,0,0,0,0,
2*xsize,2*ysize);
AddRadarImage(im4);
}
}
/*****************************************************************************/
void init_line(w,data)
Widget w;
line_data *data;
{
char message[40];
data->x1 = 511;
data->x2 = 511;
data->y1 = 0;
data->y2 = 960;
XWarpPointer(XtDisplay(winwidget4),None,XtWindow(winwidget4),
0,0,0,0,511,240);
/*
 * draw initial line
 */
XDrawLine(XtDisplay(winwidget4),XtWindow(winwidget4),rb_data.gc,
data->x1,data->y1,data->x2,data->y2);
/*
 * initial message
 */
sprintf(message,"ANGLE = %+4d ",0);
WriteMessage(message,XtWindow(winwidget4),500,1020,white);
}
void track_line(w,data,event)
Widget w;
line_data *data;
XEvent *event;
{
char message[40];
double deltax, deltay;
double angle;
/*
 * erase previous line
 */
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,
data->x1,data->y1,data->x2,data->y2);
deltax = (double)(event->xmotion.x - 511 );
deltay = (double)( 479 - event->xmotion.y );
angle = atan2(-deltax,deltay);
data->x1 = rndf(511 * (1. - sin(angle)));
data->x2 = rndf(511 * (1. + sin(angle)));
data->y1 = rndf(479 * (1. - cos(angle)));
```

```
data->y2 = rndf(479 * (1. + cos(angle)));
/*
 * draw new line
 */
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,
data->x1,data->y1,data->x2,data->y2);
sprintf(message,"ANGLE = %+5d ",rndf(-angle*180./PI));
WriteMessage(message,XtWindow(winwidget4),500,1020,white);
}
void end_line(w,data,event)
Widget w;
line_data *data;
XEvent *event;
{
double deltax, deltay;
double angle;
Time time;
/*
 * erase previous line
 */
XDrawLine(XtDisplay(w),XtWindow(w),rb_data.gc,
data->x1,data->y1,data->x2,data->y2);
deltax = (double)(event->xmotion.x - 511);
deltay = (double)(479 - event->xmotion.y );
angle = atan2(-deltax,deltay);
/*
 * remove event handlers and ungrab pointer
 */
XtRemoveEventHandler(winwidget4,PointerMotionMask,
False, track_line,&l_data);
XtRemoveEventHandler(winwidget4,ButtonPressMask,
False,end_line,&l_data);
XUngrabPointer(XtDisplay(winwidget4),time);
/*
 * clear message area
 */
XClearArea(XtDisplay(winwidget4),XtWindow(winwidget4),
0,960,0,63,False);
/*
 * set rotation angles and redraw images
 */
if(scrn1_flag)
{
xy_angle = angle;
SubRadarImage(im3);
XPutImage(display,XtWindow(winwidget4),gc1,im3,0,0,0,0,
2*xsize,2*ysize);
AddRadarImage(im3);
}
if(scrn2_flag)
{
```

```
yz_angle = angle;
SubRadarImage(im4);
XPutImage(display,XtWindow(winwidget4),gc1,im4,0,0,0,0,
2*xsize,2*ysize);
AddRadarImage(im4);
}
}
```

B.2.4 Module xframe.c

```
/*
 *  This is a file of routines for talking to the FG100 video board
 */
#include "radar.h" /* includes all X includes */
#include "frame.h"
/*
 *  Externals
 */
extern void clear_array();
extern char *vidray1, *vidray2;
extern int scrn1_flag, scrn2_flag;
extern int xsize,ysize;
extern Widget winwidget1,winwidget2,winwidget4,top2;
extern XImage *im1,*im2,*im3,*im4;
extern Display *display;
extern Colormap cmap;
extern GC gc1;
extern XFontStruct *font,*font2;
extern int fore,back;
extern Boolean vid1_flag,vid2_flag;
/*
 * globals defined in this module
 */
unsigned short *reg_addr;    /* word pointer to reg[0] */
unsigned short *mem_addr;     /* byte pointer to mem[0] */
int shmid_reg, shmid_mem;     /* shared memory id's */
/******************************************************************************/
/*
 * function to initialize shared memory and ITI board
 */
ITI_init()
{
int size, shmflag;
unsigned int p_addr, p_base, reg_p_space, mem_p_space;
unsigned short *v_base;
unsigned char *mem_base;
unsigned short temp;
char *shmaddr;
extern int errno;
extern char *sys_errlist[];
key_t key;
```

```
int i;
char *shmat();
unsigned short read_register();
signal(SIGBUS,SIG_IGN);
/* set up shared address space for control registers */
if ((shmid_reg = shmget(REG_KEY,
REG_REGION_SIZE,
SHMFLAG,
REG_PHYS_ADDR,
REG_PHYS_SPACE)) < 0)
  {
  fprintf(stderr,
       "Error allocating shared memory for control registers\n");
fprintf(stderr,"shmget: errno: %d,%s\n",
errno,sys_errlist[errno]);
exit(1);
  }
if ((v_base =
(unsigned short *)shmat(shmid_reg, 0, 0)) < 0)
  {
  fprintf(stderr,
          "Error allocating memory for control registers\n");
  fprintf(stderr, "shmat: %s\n", sys_errlist[errno]);
  exit(1);
  }
reg_addr = v_base + (0x1000/sizeof(temp));
/* now set up shared memory for video memory */
if ((shmid_mem = shmget(MEM_KEY,
MEM_REGION_SIZE,
SHMFLAG,
MEM_PHYS_ADDR,
MEM_PHYS_SPACE)) < 0)
  {
  fprintf(stderr,
       "Error allocating shared memory for video memory\n");
fprintf(stderr,"shmget: %d -- %s\n",
errno,sys_errlist[errno]);
exit(1);
  }
if ((mem_addr =
(unsigned short *)shmat(shmid_mem, 0, 0)) < 0)
  {
  fprintf(stderr,
          "Error allocating memory for video memory\n");
  fprintf(stderr, "shmat: %d -- %s\n",
errno,sys_errlist[errno]);
  exit(1);
  }
        /*
         * Actually do something with device here .....
         */
```

```
/* do a soft reset and set up default controls */
write_register(LUT_CONTROL,0xffff); /* reset board */
write_register(LUT_CONTROL,0x3000); /* select board */
write_register(ZOOM,0); /* zoom & regmux = 0 */
write_register(MEMORY_CONTROL,0x4040); /* Z-mode & pixel buf*/
write_register(PROCESSOR_MASK,0); /* no protection */
write_register(VIDEO_MASK,0); /* no protection */
write_register(PIXEL_BUFFER,0); /* clear pix buffer reg. */
write_register(X_POINTER,0); /* set x-pointer */
write_register(Y_POINTER,0); /* set y-pointer */
write_register(POINTER_CONTROL,0); /* no pointers */
write_register(CPU_ADR_CONTROL,0x0407); /* enable fb add's. */
write_register(X_SPIN,0x10); /* try this, or 10h */
write_register(Y_SPIN,0);  /* no y-spin */
write_register(PAN,0); /* no pan */
write_register(SCROLL,0);  /* no scroll */
write_register(STATUS_CONTROL,0x3040);
ITI_lut();  /* set up LUT's */
}
/*
 * Function to initialize the LUT's
 */
ITI_lut()
{
unsigned short i;
write_register(LUT_CONTROL,0x2000); /* select LUT memory */
for(i=0;i<256;i++)
{
mem_addr[RED_BASE/2 + i] = i;
mem_addr[BLUE_BASE/2 + i] = i;
mem_addr[GREEN_BASE/2 + i] = i;
}
for(i=0;i<256;i++)
{
mem_addr[ILUT_BASE/2 + i] = i;
}
write_register(LUT_CONTROL,0x3000); /* select video mem. */
}
/*
 *  Function to do a single frame grab and display it in the correct
 *  window.  NOTE : This function deletes any current radar image.
 */
ITI_frame(cam)
unsigned char cam;
{
/* do an acquisition */
unsigned short data;
int size,i;
size = 512*480;
wait_vb();
data = (unsigned short)read_register(STATUS_CONTROL);
```

```
if(!cam) data = data & 0x0040;
if(cam) data = data & 0x0048;
write_register(STATUS_CONTROL,data);
while(data=((unsigned short)read_register(STATUS_CONTROL)&0x3000));
if(!cam) data = data | 0x2040;
if(cam) data = data | 0x2048;
write_register(STATUS_CONTROL,data);
while(data=((unsigned short)read_register(STATUS_CONTROL)&0x3000));
sleep(1);
if(!scrn1_flag && !scrn2_flag)    /* in multiscreen mode */
{
    if(!cam)                        /* overhead camera */
    {
for(i=0;i<size;i++)
  vidray1[i] = (unsigned char)((mem_addr[i] & 0x00ff)/2);
vid1_flag = True;
/* this will take some time so tell'em */
WriteMessage("Please wait, this will take awhile...",
XtWindow(winwidget1),5,505,back);
CreateImage(vidray1,im1,xsize,ysize);
CreateImage(vidray1,im3,2*xsize,2*ysize);
XPutImage(XtDisplay(winwidget1),XtWindow(winwidget1),
   gc1,im1,0,0,0,0,xsize,ysize);
/* erase message by writing same one in black */
        WriteMessage("Please wait, this will take awhile...",
XtWindow(winwidget1),5,505,fore);
    }
    else
    {
for(i=0;i<size;i++)
  vidray2[i] = (unsigned char)((mem_addr[i] & 0x00ff)/2);
vid2_flag = True;
WriteMessage("Please wait, this will take awhile...",
XtWindow(winwidget2),5,505,back);
CreateImage(vidray2,im2,xsize,ysize);
CreateImage(vidray2,im4,2*xsize,2*ysize);
XPutImage(XtDisplay(winwidget2),XtWindow(winwidget2),
gc1,im2,0,0,0,0,xsize,ysize);
WriteMessage("Please wait, this will take awhile...",
XtWindow(winwidget2),5,505,fore);
    }
}
if(scrn1_flag)
{
WriteMessage("Please wait, this will take awhile...",
XtWindow(winwidget4),300,1020,back);
if(!cam)
{
     for(i=0;i<size;i++)
       vidray1[i] = (unsigned char)((mem_addr[i] & 0x00ff)/2);
   vid1_flag = True;
   CreateImage(vidray1,im1,xsize,ysize);
   CreateImage(vidray1,im3,2*xsize,2*ysize);
   XPutImage(XtDisplay(winwidget4),XtWindow(winwidget4),
gc1,im3,0,0,0,0,2*xsize,2*ysize);
}
else
{
     for(i=0;i<size;i++)
      vidray2[i] = (unsigned char)((mem_addr[i] & 0x00ff)/2);
   vid2_flag = True;
   /* create the images but don't display it */
   CreateImage(vidray2,im2,xsize,ysize);
   CreateImage(vidray2,im4,2*xsize,2*ysize);
}
WriteMessage("Please wait, this will take awhile...",
XtWindow(winwidget4),300,1020,fore);
}
if(scrn2_flag)
{
WriteMessage("Please wait, this will take awhile...",
XtWindow(winwidget4),300,1020,back);
if(!cam)
{
     for(i=0;i<size;i++)
      vidray1[i] = (unsigned char)((mem_addr[i] & 0x00ff)/2);
   vid1_flag = True;
   /* create the images but don't display */
   CreateImage(vidray1,im1,xsize,ysize);
   CreateImage(vidray1,im3,2*xsize,2*ysize);
}
else
{
     for(i=0;i<size;i++)
      vidray2[i] = (unsigned char)((mem_addr[i] & 0x00ff)/2);
   vid2_flag = True;
   CreateImage(vidray2,im2,xsize,ysize);
   CreateImage(vidray2,im4,2*xsize,2*ysize);
   XPutImage(XtDisplay(winwidget4),XtWindow(winwidget4),
gc1,im4,0,0,0,0,2*xsize,2*ysize);
}
WriteMessage("Please wait, this will take awhile...",
XtWindow(winwidget4),300,1020,fore);
}
}
/*
 *  function to put in continuous acquisition mode
 */
ITI_cont()
{
unsigned short data;
while(data=((unsigned short)read_register(STATUS_CONTROL)&0x3000));
```

```
data = read_register(STATUS_CONTROL);
data = data | 0x3040;
write_register(STATUS_CONTROL,data);
}
/*
 *  function to switch cameras
 */
ITI_camera(cam)
unsigned char cam;
{
unsigned short data;
data = read_register(STATUS_CONTROL);
data = data & 0x0fff;
write_register(STATUS_CONTROL,data);
while(data=((unsigned short)read_register(STATUS_CONTROL)&0x3000));
if(cam)
{
  data = read_register(STATUS_CONTROL);
  data = data & 0xff40;
  data = data | 0x0008;
  write_register(STATUS_CONTROL,data);
}
else
{
  data = read_register(STATUS_CONTROL);
  data = data & 0xff40;
  write_register(STATUS_CONTROL,data);
}
}
/*
 * Function to release shared memory mapping for ITI board
 */
ITI_close()
{
/*
 * Now that we're done playing with the device, free the region.
 */
shmdt(0);
shmdt(0);
}
/*
 * Write a FG100 control register
 */
write_register(r,data)
unsigned short r,data;
{
reg_addr[r/2] = data;
}
/*
 * Read a FG100 control register
 */
```

```
unsigned short read_register(r)
unsigned short r;
{
unsigned short data;
data = (unsigned short)reg_addr[r/2];
return data;
}
/*
 * Wait for a vertical blanking period
 */
wait_vb()
{
/* wait out pending vertical blank */
while(!(read_register(STATUS_CONTROL)&0x0400));
/* wait for next vertical blank */
while(read_register(STATUS_CONTROL)&0x0400);
/* wait it out */
while(!(read_register(STATUS_CONTROL)&0x0400));
}
```

B.2.5 Module xcom.c

```
/*
 *
 *    XCOM.C :
 *
 *    This is a file of functions for handling data transfer between
 *    XRADAR and TTT.
 *
 *
 */
#include "radar.h" /* includes all X includes */
extern XImage *im1,*im2,*im3,*im4;
extern Display *display;
extern Colormap cmap;
extern GC gc1,gc2;
extern Widget   toplevel, top2,
                shell1, shell2,
                shell3, shell4,
                winwidget1, winwidget2,
                winwidget3, winwidget4,
button15,exit_message;
extern int      gxpos1, gypos1,
                gxpos2, gypos2,
                gxpos3, gypos3,
                gxpos4, gypos4,
                xsize, ysize,
                screen;
extern unsigned int red,white,blue,black;
extern int scrn1_flag,scrn2_flag;
extern int num_pnts;
extern float in_angle;
extern float radar_data[IMAGE_PNTS][6];
/*****************************************************************************/
/*
 * Callback for radar imaging button
 *
 * NOTICE : This function has its own event processing loop for handling
 *      events while performing continuous radar image display. Only two
 *     events are recognized by this handler :
 *
 * 1) Property change events on the property being used to
 *    pass data from TTT to XRADAR.
 * 2) A button press to exit the event handling loop and
 *    exit this callback routine.
 *
 *     This function also mixes Xlib calls with Xt Intrinsics and
 *     Xaw widgets. Care is needed when attempting to modify this
 *     function.
 */
void image_loop(w, call_data, client_data)
Widget w;
caddr_t call_data, client_data;
{
Window root,window;
XEvent event;
int i,x,y;
if(!scrn1_flag && !scrn2_flag)
{
    window = XtWindow(winwidget2);
    x = 5;
    y = 505;
    WriteMessage("To exit this function, click on RTDP button again.",
window,x,y,white);
}
else
{
    window = XtWindow(winwidget4);
    x = 400;
            y = 1020;
    WriteMessage("To exit this function, click on RTDP button again.",
window,x,y,white);
}
root = DefaultRootWindow(XtDisplay(toplevel));
RDR_DATA = XInternAtom(XtDisplay(toplevel),"Data",0);
RDR_DATA_TYPE = XInternAtom(XtDisplay(toplevel),"Data Type",0);
/*
 * write message to tell user how to exit this function
 */
XSelectInput(XtDisplay(toplevel),root,
PropertyChangeMask);
while(TRUE) /* loop until user exits via button press */
```

```
{
XtNextEvent(&event);  /* get next event */
switch(event.type)    /* check event type */
{
case PropertyNotify :
if(event.xproperty.window == root &&
   event.xproperty.atom == RDR_DATA)
{
   printf("Got the data !\n");
   display_data();    /* display new image */
}
else
   XtDispatchEvent(&event);
break;
case ButtonPress :
if(event.xbutton.window == XtWindow(button15))
{
         XDeleteProperty(XtDisplay(toplevel),
root, RDR_DATA);
    XDeleteProperty(XtDisplay(toplevel),
root,RDR_DATA_TYPE);
    /* erase message */
         WriteMessage("To exit this function, click\
 on RTDP button again.", window,
 x,y,black);
    return;
}
default : XtDispatchEvent(&event);
}
}
}
display_data()
{
int i,j, type, format, nitems, left;
char *retdata;
float *fretdata;
XGetWindowProperty(XtDisplay(toplevel),
DefaultRootWindow(XtDisplay(toplevel)),
RDR_DATA,0,4096,FALSE,
RDR_DATA_TYPE, &type, &format,
&nitems, &left, &retdata);
if(type == RDR_DATA_TYPE)
{
fretdata = retdata;
num_pnts = (int)(*fretdata);
in_angle = *(fretdata+1);
for(i=0;i<num_pnts;i++)
   for(j=0;j<6;j++)
   {
radar_data[i][j] = *(fretdata+6*(i+1)+j);
   }
if(!scrn1_flag && !scrn2_flag)
{
    SubRadarImage(im1); /* clear screens */
    SubRadarImage(im2);
    XPutImage(display,XtWindow(winwidget1),gc1,im1,0,0,0,0,
xsize,ysize);  /* put video image back */
    AddRadarImage(im1);        /* add new radar image */
    XPutImage(display,XtWindow(winwidget2),gc1,im2,0,0,0,0,
xsize,ysize);
    AddRadarImage(im2);
}
if(scrn1_flag)
{
    SubRadarImage(im3);
    XPutImage(display,XtWindow(winwidget4),gc1,im3,0,0,0,0,
2*xsize,2*ysize);
    AddRadarImage(im3);
}
if(scrn2_flag)
{
    SubRadarImage(im4);
    XPutImage(display,XtWindow(winwidget4),gc1,im4,0,0,0,0,
2*xsize,2*ysize);
    AddRadarImage(im4);
}
}
}
```

B.2.6 Module xutil.c

```
/*
 * util.c -- This file contains support routines for the xrdr.c
 *   main program.
 */
#include "radar.h" /* includes all X includes */
extern XImage *im1, *im2, *im3, *im4;
extern Display *display;
extern Colormap cmap;
extern GC  gc1;
extern XFontStruct *font,*font2;
extern unsigned long cols[];
extern unsigned int fore, back;
extern Widget  toplevel,  top2,
shell1, shell2,
shell3, shell4,
winwidget1,winwidget2,
winwidget3,winwidget4,
popfile1, popfile2,
error_popup, msg_popup;
extern float x_scale, y1_scale, y2_scale, z_scale;
extern int x_offset, y1_offset, y2_offset, z_offset;
extern int horizontal_flag,scrn1_flag,scrn2_flag;
extern int xsize,ysize;
extern float radar_data[IMAGE_PNTS][6];
extern int num_pnts;       /* number of points in current radar image */
extern float in_angle;
extern char  *vidray1, *vidray2, *Image1, *Image2;
extern char *BigImage1, *BigImage2;
void Null_func();
double square();
static XtActionsRec actionsTable [] = {
{"Null_func",Null_func}, };
static char defaultTranslations[] = "Ctrl<Key>J: Null_func() \n\
     Ctrl<Key>O: Null_func() \n\
     Ctrl<Key>M: Null_func() \n\
     <Key>Return: Null_func()";
Widget error_button;      /* global widgets in this module */
FILE *video_fp, *radar_fp;
/******************************************************************************/
/*
 *  a null function which does nothing but which is needed for intercepting
 *  <RET> typed in text widgets, ie) do nothing if a <RET> is typed.
 */
void Null_func()
{
}
/******************************************************************************/
/*
 *  a math round function
 */
int rndf(number)
double number;
{
double trash;
if(modf(number,&trash) > 0.5) return ceil(number);
else return floor(number);
}
/******************************************************************************/
double square(x)
double x;
{
double result;
result = x*x;
return result;
}
/******************************************************************************/
void rotate_point(x,y,angle)
float *x,*y;
float angle;
{
double tx,ty;
double r,theta;
```

```
tx = *x;
ty = *y;
/*
 * convert to polar
 */
r = sqrt(tx*tx + ty*ty);
if((tx == 0.0)&&(ty==0.0))
   theta = 0.0;
else
   theta = atan2(ty,tx);
/*
 * rotate point
 */
*x = (float)(r * cos(theta + angle));
*y = (float)(r * sin(theta + angle));
}
/*******************************************************************************/
clear_array(array,size)
char *array;
int size;
{
int i;
for(i=0;i<size;i++) *(array + i ) = 0;
}
/*******************************************************************************/
/*
 *  callback proc to popdown the popup windows
 */
void pop_down(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
XtPopdown((Widget)client_data);
}
/*******************************************************************************/
/*
 *  a function to activate a popup error window and display the
 *  appropriate error message.
 */
void Error_window(err_code,err_no,str)
int err_code,err_no;
char *str;
{
static char error_string[100];
static Arg arg[] = { {XtNlabel, (XtArgVal)error_string} };
char temp[100];
switch(err_code)
{
    case 1 : /* file i/o error */
     sprintf(error_string, "  X-Radar I/O Error : ");
                    sprintf(temp,"File '%s' Not Found ... ",str);
```

```
         strcat(error_string,temp);
         break;
        case 2 : /* scale factors not set yet */
         sprintf(error_string," X-RADAR Protocol Error : ");
         sprintf(temp,"Radar image scale factors not set ... ");
         strcat(error_string,temp);
         break;
        case 3 : /* trying to set scale factor in wrong mode */
         sprintf(error_string," X-RADAR Protocol Error : ");
         strcat(error_string,"Must be in single screen ");
         strcat(error_string,"mode to set scale factors.");
         break;
        case 4 : /* trying to set offsets in wrong mode */
         sprintf(error_string," X-RADAR Protocol Error : ");
         strcat(error_string,"Must be in single screen ");
         strcat(error_string,"mode to set offsets.");
         break;
        case 5 : /* trying to do shift,etc. to ISAR image */
         sprintf(error_string," X-RADAR Protocol Error : ");
         strcat(error_string,"Function not available for ");
         strcat(error_string,"ISAR images.");
         break;
        case 6 : /* trying to rotate in wrong mode */
         sprintf(error_string," X-RADAR Protocol Error : ");
         strcat(error_string,"Must be in single screen ");
         strcat(error_string," mode to rotate image.");
         break;
        case 7 : /* bad filename : unable to open file */
         sprintf(error_string,"X-RADAR I/O Error : ");
         strcat(error_string,"Bad file name , unable ");
         strcat(error_string,"to create or open file.");
         break;
}
XtSetValues(error_button,arg,XtNumber(arg));
XBell(XtDisplay(toplevel), 100);
XtPopup(error_popup,XtGrabNone);
}
/******************************************************************************/
/*
 * write_video_file() -- writes a video image to disk in binary format
 */
write_video_file(fname,array)
char fname[];
char *array;
{
/* open file for writing */
if((video_fp = fopen(fname,"wb")) == NULL)
{
    Error_window(7,0,fname);
    return(0);
}
```

```
/* write video file to disk */
fwrite(array,sizeof(*array),(512*512),video_fp);
fclose(video_fp);
} /* end write_video_file() */
/*******************************************************************************/
/*
 * read_video_file() -- reads video image file into named array memory,
 * initializes globals: mapchars, and video_fp  (file descriptor)
 */
read_video_file(fname,array)
char fname[];
char *array;
{
    long i,j;
    /* open video image file */
    if((video_fp = fopen(fname, "rb")) == NULL )
    {
Error_window(1,0,fname);
return(0);
    }
    /* read video file into video array */
    fread(array, sizeof(*array),(512*512),video_fp);
    fclose(video_fp);
    return(1);
}  /*end read_video_file()*/
/*******************************************************************************/
/*
 * read_sif_file() -- reads sif image file into named array memory,
 * initializes globals: mapchars, and video_fp  (file descriptor).
 * Included to provide upward compatibility for SIF files (video image
 * files from MetraByte MV-1 software). This function is not currently used.
 */
read_sif_file(fname,array)
char fname[];
char *array;
{
    long i,j;
    char t1[256*240];
    char t2[256*240];
    char t3[256*240];
    char t4[256*240];
    /* open video image file */
    if((video_fp = fopen(fname, "rb")) == NULL )
    {
Error_window(1,0,fname);
return(0);
    }
    /* read video file into video array */
    fread(t1,sizeof(char),(256*240),video_fp);
    fread(t2,sizeof(char),(256*240),video_fp);
    fread(t3,sizeof(char),(256*240),video_fp);
    fread(t4,sizeof(char),(256*240),video_fp);
    for(j=0;j<240*512;j+=512)
    {
        for(i=0;i<256;i++) array[i+j] = t1[i + j/2];
for(i=0;i<256;i++) array[i+256+j] = t2[i + j/2];
    }
    for(j=0;j<240*512;j+=512)
    {
        for(i=0;i<256;i++) array[i+j + 240*512] = t3[i + j/2];
for(i=0;i<256;i++) array[i+256+j + 240*512] = t4[i + j/2];
    }
    fclose(video_fp);
    return(1);
}  /*end read_file()*/
/*****************************************************************************/
/*  a function to read in x,y,z,mag format radar data files.
 *
 */
read_radar_file(fname)
char fname[];
{
float x,y,z,mag;
int i;
FILE *radar_fp;
if((radar_fp = fopen(fname,"r")) == NULL)
{
Error_window(1,0,fname);
return(0);
}
fscanf(radar_fp,"%d %f\n",&num_pnts,&in_angle);
for(i=0;i<num_pnts;i++)
{
   fscanf(radar_fp,"%f %f %f %d\n",&x,&y,&z,&mag);
   radar_data[i][0] = x;
   radar_data[i][1] = y;
   radar_data[i][2] = z;
   radar_data[i][3] = mag;
}
fclose(radar_fp);
return(1);   /* successful read of data file */
}
/*****************************************************************************/
/* a function to create a "composite" widget which is like a dialog Widget,  */
/* but which is easier to control.      */
Widget CreateTextWidget(parent,label_str,cb_func,buffer,
                                        x_pos,y_pos)
Widget parent;           /* parent of composite widget */
char *label_str;         /* prompt string to display */
void (*cb_func)();       /* pointer to callback function */
char *buffer;            /* pointer to buffer to hold string */
int x_pos, y_pos;        /* x,y position of popup */
```

```
{
Widget popup, button, box, label, w;
Arg  args[10];
int x_size, y_size;
char geom_str[20];
XtCallbackRec callback[2];
XtTranslations trans_table;
/*
 *   register new actions and compile the new translation table
 */
XtAddActions(actionsTable,XtNumber(actionsTable));
trans_table =
XtParseTranslationTable(defaultTranslations);
/*
 *  create a popup shell for the text widget
 */
/* first set up geometry string */
x_size = 450;
y_size = 100;
sprintf(geom_str,"%1ix%1i",x_size,y_size);
XtSetArg(args[0], XtNx, x_pos);
XtSetArg(args[1], XtNy, y_pos);
XtSetArg(args[2], XtNgeometry,geom_str);
XtSetArg(args[3], XtNallowShellResize,False);
popup = XtCreatePopupShell("popup",transientShellWidgetClass,
   parent,args,4);
XtSetArg(args[0], XtNheight, y_size);
XtSetArg(args[1], XtNwidth, x_size);
XtSetArg(args[2], XtNforeground,fore);
XtSetArg(args[3], XtNbackground,back);
box = XtCreateManagedWidget("box",boxWidgetClass,popup,args,4);
XtSetArg(args[0], XtNlabel, label_str);
XtSetArg(args[1], XtNforeground,fore);
XtSetArg(args[2], XtNbackground,back);
label = XtCreateManagedWidget("label",labelWidgetClass,box,args,3);
/*
 *  create an ascii string widget.
 */
XtSetArg(args[0], XtNeditType, (XtArgVal)XttextEdit);
XtSetArg(args[1], XtNstring, buffer);
XtSetArg(args[2], XtNwidth, (x_size-10));
XtSetArg(args[3], XtNlength, 40);
XtSetArg(args[4], XtNforeground,fore);
XtSetArg(args[5], XtNbackground,back);
w = XtCreateManagedWidget("string",asciiStringWidgetClass,box,
  args,6);
/*
 *  create a button to close the box
 */
XtSetArg(args[0], XtNlabel, " OK ");
XtSetArg(args[1], XtNforeground,fore);
XtSetArg(args[2], XtNbackground,back);
button = XtCreateManagedWidget("command",commandWidgetClass,
      box, args,3);
/*
 *  add callbacks to pop down the box
 */
XtAddCallback(button, XtNcallback,cb_func,buffer);
XtAddCallback(button, XtNcallback,pop_down,popup);
/*
 *  Merge the defined translations with the existing
 *  translations.
 */
XtOverrideTranslations(w,trans_table);
XtSetKeyboardFocus(box,w);
XtRealizeWidget(popup);
XSetWindowColormap(XtDisplay(popup),XtWindow(popup),
   cmap);
return(popup);
}
/*****************************************************************************/
/*  callback for the ScaleWidget. When executed it sets the sensitivity
 *  of all the widgets within the ScaleWidget to false. The sensitivity
 *  of the widgets is reset after the rubber band process has been completed.
 */
void ClickOff(w, w_struct, call_data)
Widget w;
widget_struct *w_struct;
caddr_t call_data;
{
Arg arg[1];
XtSetArg(arg[0],XtNsensitive,False);
XtSetValues(w_struct->widget1, arg,1);
XtSetValues(w_struct->widget2, arg,1);
XtSetValues(w_struct->widget3, arg,1);
XtSetValues(w_struct->widget4, arg,1);
if(w == w_struct->widget1) horizontal_flag = True;
else horizontal_flag = False;
}
/*****************************************************************************/
/*  creates a "composite" widget for entering scale factors, etc.
 *
 */
Widget  CreateScaleWidget(parent,label,label_str1,label_str2,cb_func,
buffer1,buffer2,w_struct,x_pos,y_pos)
Widget parent;           /* parent of composite widget */
char *label;             /* label for whole box */
char *label_str1,
     *label_str2;        /* prompt strings to display */
void (*cb_func)();       /* pointer to callback function */
char *buffer1,
     *buffer2;           /* pointers to buffers to hold strings */
```

```
widget_struct *w_struct; /* pointer to structure to hold widgets */
int x_pos, y_pos;        /* x,y position of popup */
{
Widget popup, box, label1, label2, w1, w2, close_button;
Arg  args[10];
int x_size, y_size;
char geom_str[20];
XtCallbackRec callback[2];
XtTranslations trans_table;
/*
 *   register new actions and compile the new translation table
 */
XtAddActions(actionsTable,XtNumber(actionsTable));
trans_table =
XtParseTranslationTable(defaultTranslations);
/*
 *  create a popup shell for the scale widget
 */
/* first set up geometry string */
x_size = 500;
y_size = 125;
sprintf(geom_str,"%1ix%1i",x_size,y_size);
XtSetArg(args[0], XtNx, x_pos);
XtSetArg(args[1], XtNy, y_pos);
XtSetArg(args[2], XtNgeometry,geom_str);
XtSetArg(args[3], XtNallowShellResize,False);
popup = XtCreatePopupShell("popup",transientShellWidgetClass,
   parent,args,4);
XtSetArg(args[0], XtNheight, y_size);
XtSetArg(args[1], XtNwidth, x_size);
XtSetArg(args[2], XtNforeground,fore);
XtSetArg(args[3], XtNbackground,back);
box = XtCreateManagedWidget("box",boxWidgetClass,popup,args,4);
XtSetArg(args[0], XtNlabel, label);
XtSetArg(args[1], XtNforeground,fore);
XtSetArg(args[2], XtNbackground,back);
XtCreateManagedWidget("label",labelWidgetClass,box,args,3);
XtSetArg(args[0], XtNlabel, label_str1);
XtSetArg(args[1], XtNforeground,fore);
XtSetArg(args[2], XtNbackground,back);
label1 = XtCreateManagedWidget("label",commandWidgetClass,box,args,3);
/*
 *  create an ascii string widget.
 */
XtSetArg(args[0], XtNeditType, (XtArgVal)XttextEdit);
XtSetArg(args[1], XtNstring, buffer1);
XtSetArg(args[2], XtNwidth, 110);
XtSetArg(args[3], XtNlength, 10);
XtSetArg(args[4], XtNforeground,fore);
XtSetArg(args[5], XtNbackground,back);
w1 = XtCreateManagedWidget("string",asciiStringWidgetClass,box,
  args,6);
/*
 *  create another label and ascii string widget.
 */
XtSetArg(args[0], XtNlabel, label_str2);
XtSetArg(args[1], XtNforeground,fore);
XtSetArg(args[2], XtNbackground,back);
label2 = XtCreateManagedWidget("label",commandWidgetClass,box,args,3);
XtSetArg(args[0], XtNeditType, (XtArgVal)XttextEdit);
XtSetArg(args[1], XtNstring, buffer2);
XtSetArg(args[2], XtNwidth, 110);
XtSetArg(args[3], XtNlength, 10);
XtSetArg(args[4], XtNforeground,fore);
XtSetArg(args[5], XtNbackground,back);
w2 = XtCreateManagedWidget("string",asciiStringWidgetClass,box,
  args,6);
/*
 *  create a button to close the box
 */
XtSetArg(args[0], XtNlabel, " OK ");
XtSetArg(args[1], XtNforeground,fore);
XtSetArg(args[2], XtNbackground,back);
XtSetArg(args[3], XtNsensitive,False);
close_button = XtCreateManagedWidget("command",commandWidgetClass,
      box, args,4);
/*
 *  add callbacks to pop down the box
 */
XtAddCallback(close_button, XtNcallback,cb_func,NULL);
XtAddCallback(close_button, XtNcallback,pop_down,popup);
/*
 * fill in the structure for client_data and add callbacks
 * for the command labels.
 */
w_struct->widget1 = label1;
w_struct->widget2 = label2;
w_struct->widget3 = w1;
w_struct->widget4 = w2;
w_struct->widget5 = close_button;
XtAddCallback(label1,XtNcallback,ClickOff,w_struct);
XtAddCallback(label2,XtNcallback,ClickOff,w_struct);
/*
 *  Merge the defined translations with the existing
 *  translations.
 */
XtOverrideTranslations(w1,trans_table);
XtOverrideTranslations(w2,trans_table);
XtRealizeWidget(popup);
XSetWindowColormap(XtDisplay(popup),XtWindow(popup),
   cmap);
return(popup);
```

```
}
/********************************************************************************/
/* a function to create a widget for error messages which must be acknowledged
 * by the user. ie) click on the widget to close the error message window
 */
Widget CreateErrorWidget(parent)
Widget parent;
{
Widget  popup, box;
Arg  args[10];
XtSetArg(args[0], XtNx, 400);
XtSetArg(args[1], XtNy, 900);
XtSetArg(args[2], XtNallowShellResize, True);
popup = XtCreatePopupShell("popup",transientShellWidgetClass,
   parent,args,3);
/*
 * create a box to go in the shell
 */
XtSetArg(args[0], XtNforeground, fore);
XtSetArg(args[1], XtNbackground, back);
XtSetArg(args[2], XtNtransient, True);
XtSetArg(args[3], XtNheight, 100);
XtSetArg(args[4], XtNwidth, 200);
box = XtCreateManagedWidget("box",boxWidgetClass,popup,args,5);
/*
 *  create a button to go in the box;
 *  only thing the button does is display the error message
 *  and pop down the shell when selected.
 *  Error message set in function Error_window().
 */
XtSetArg(args[0], XtNforeground, fore);
XtSetArg(args[1], XtNbackground, back);
error_button = XtCreateManagedWidget("error",commandWidgetClass,
box,args,2);
/*
   * add callback to popdown the shell
 */
XtAddCallback(error_button, XtNcallback, pop_down,popup);
XtRealizeWidget(popup);
XSetWindowColormap(XtDisplay(popup), XtWindow(popup),
   cmap);
return(popup);
}
/********************************************************************************/
/*  ExposeEvent event handler for the graphics widgets */
void RePaint(w,client_data,event)
Widget w;
caddr_t client_data;
XEvent *event;
{
int dest_x,dest_y,
    width,height,
    count;
dest_x = event->xexpose.x;
dest_y = event->xexpose.y;
width = event->xexpose.width;
height = event->xexpose.height;
count = event->xexpose.count;
if(!count)
{
if(w == winwidget1)
{
XPutImage(XtDisplay(w),XtWindow(w),
  gc1,im1,dest_x,dest_y,
dest_x,dest_y,width,height);
AddRadarImage(im1);
}
if(w == winwidget2)
{
XPutImage(XtDisplay(w),XtWindow(w),
gc1,im2,dest_x,dest_y,
dest_x,dest_y,width,height);
AddRadarImage(im2);
}
if(w == winwidget4)
{
if(scrn1_flag)
{
XPutImage(XtDisplay(w),XtWindow(w),
gc1,im3,dest_x, dest_y,
dest_x,dest_y,width,height);
AddRadarImage(im3);
}
else if(scrn2_flag)
{
XPutImage(XtDisplay(w),XtWindow(w),
gc1,im4,dest_x, dest_y,
dest_x,dest_y,width,height);
AddRadarImage(im4);
}
}
}
XFlush(XtDisplay(w));
}
/*******************************************************************************/
/*
 * A function to write a message on the screen immediately ( putting up
 * a message widget requires too much time due to buffering )
 *
 */
WriteMessage(message,window,x,y,color)
char *message;
```

```
Window window;
int x,y;
unsigned int color;
{
XSetForeground( display, gc1, color);
XSetFont(display,gc1,font2->fid);
XDrawImageString(display,window,gc1,x,y,message,strlen(message));
XSync(display,0);
}
```

Appendix C

X-RADAR Errors

The recoverable errors which can be generated while using X-RADAR are listed below by the error message displayed for each. While unrecoverable errors will usually cause the program to crash, every attempt has been made to keep this from happening.

C.1 X-RADAR Error Messages

1. "X-RADAR I/O Error : File *filename* Not Found"
   While this error is fairly self-explanatory, it can also occur if no file name is entered at the prompt. Suggested remedies are making sure the file exists and is readable and checking the complete path and file extension spellings.

2. "X-RADAR Protocol Error : Must be in single screen mode to set scale factors."
   Again, this error is self-explanatory. The reason this was implemented as an error was to lessen the possible visual errors in setting the scale factors.

3. "X-RADAR Protocol Error : Function not available for ISAR images."
   This error covers a number of functions which were not implemented for ISAR images due to the format of ISAR images. While it is technically possible to apply these functions to ISAR images, it was decided to leave this work for the future.

4. "X-RADAR Protocol Error : Must be in single screen mode to rotate image." As described previously for setting the scale factors, a full-screen image is easier to see than a small image.

5. "X-RADAR Protocol Error : Must be in single screen mode to shift image." Same as above.

Appendix D

X-RADAR Data File Formats

D.1 Radar Image File Format

Radar image files are formatted ASCII text files consisting of the number of scattering centers, the angle of incidence of the radar signal, followed by the x, y, z coordinates (in inches from the phase center of the target) and the magnitude of each scattering center (normalized to 1.0) in the following format (␣ = <space>) :

number of scattering centers␣incident angle <ret>

x␣y␣z␣magnitude <ret>

x␣y␣z␣magnitude <ret>

⋮

<eof>

An example is given below :

```
<tof>
4 45.0 <ret>
12.24 10.38 4.57 .87 <ret>
4.08 5.67 7.99 .45 <ret>
34.45 40.00 23.00 .33 <ret>
27.99 30.12 12.87 .55 <ret>
<eof>
```

D.2 Video Image File Format

Video image files consist of row-column binary raster scan dumps of the 512x480 image array. The data type of each pixel in the raster scan is 'unsigned short' or 'byte', with a value of zero representing the lowest brightness and a value of 255 representing the highest brightness. The pixel data is written to the file as an unformatted stream, as shown below :

<tof>pixel1,row1;pixel2,row1;...;pixel511,row1;pixel1,row2; pixel2,row2; ...;pixel511,row480 <eof>

D.3 ISAR Image File Format

ISAR image files are of the same format as video image files but the data is scaled differently. Due to the look-up tables used for displaying the ISAR data, a pixel value of 0 represents no data, a value of 128 represents the lowest magnitude to be displayed and a value of 255 represent the highest magnitude to be displayed.

Bibliography

[1] A. Dominek, I. Gupta, W.D. Burnside, "A Novel Approach for Two- and Three-Dimensional Imaging", The Ohio State University ElectroScience Laboratory, Department of Electrical Engineering.

[2] Walton, Eric K., "Comparison of Fourier and Maximum Entropy Techniques for High Resolution Scattering Studies", *Radio Science*, Vol. 22, No. 1, January–Febuary 1987, pp. 350-356.

[3] Harrington, Roger F., *Time-Harmonic Electromagnetic Fields*, McGraw-Hill, 1987, pp. 292-298.

[4] T.H. Lee and W.D. Burnside, "A Focussed Image Processing Procedure Using Near Zone Scattered Fields Obtained in the Compact Range," Technical Report 720150-1, 1988, The Ohio State University ElectroScience Laboratory, Department of Electrical Engineering.

[5] Mensa, D.L., *High Resolution Radar Imaging*, Artech House, 1981.

[6] Lin, W.,"A User-Oriented, Menu-Driven, Software Interface to Control a Radar System and Manipulate Measured Data", Ohio State University, Department of Electrical Engineering, M.Sc. thesis, August, 1988.

[7] R.R. Swick and T. Weissman, "X Toolkit Widgets - C Language X Interface", Massachusetts Institute of Technology and Digital Equipment Corporation, 1988.

[8] J. McCormack, P. Asente, R.R. Swick, "X Toolkit Intrinsics - C Language Interface", Massachusetts Institute of Technology and Digital Equipment Corporation, 1988.

[9] Jones, Oliver, *Introduction to the X Window System*, Prentice Hall, Inc., 1989.

[10] Young, Douglas A., *X Window Systems : Programming and Applications with Xt*, Prentice Hall, Inc., 1989.

[11] "FG-100-V User's Manual", Part Number 47-H10018-01, Imaging Technology Inc., 1987.

[12] "UTeK V User's Reference", Part Number 070-7576-00, Tektronix Inc., 1989.

[13] M.T. Bell and W. Scheckla, "OEM/VAR System Integration Support for Third-Party I/O Bus Devices", Tektronix Inc., 1989.

[14] Lin, W., Personal Communication, in preparation for publication, The Ohio State University ElectroScience Laboratory, Department of Electrical Engineering.