1996

Automation of real-time X-ray scans

Madhusudhan R. Midhe
Iowa State University

Automation of real-time X-ray scans

by

Madhusudhan Rao Midhe

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Major Professors: Doug W Jacobson and Joe Gray

Iowa State University

Ames, Iowa

1996

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Madhusudhan Rao Midhe

has met the thesis requirements of Iowa State University

Committee Member

Co-major Professor

Co-major Professor

For the Major Program

For the Graduate College

DEDICATION

To my parents and to my brother without whose support I wouldn't be at this position and to my friend Bhanu for his constant encouragement and support.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

My primary thanks go to Dr. Terrence Jensen, without whose guidance and support, this research work wouldn't be a possibility. I wish to express my deepest appreciation for his constant guidance and also for pouring over every inch of this report with painstaking attention to detail and making a semi-infinite number of helpful suggestions. I also would like to thank Dr. Joe Gray for his continual support and encouragement. I would also like to thank Dr. Doug Jacobson, my major advisor for his valuable time and cooperation. I am also grateful to Dr. Charles Wright for his valuable time. This work was supported by the FAA Center for Aviation Systems Reliability program under Federal Aviation Administration Grant No. 95G-032.

Finally, I would like to thank all those at the Center for NDE for helping me make this project a success. I sincerely appreciate their effort.

1 INTRODUCTION

The present day needs of the commercial market have forced the industry to manufacture materials which are stronger, weigh less and cost less. In other words, it is of prime concern to characterize the material properties. The characterization of a material can be done either at the time of fabrication or while the material is being used.

A material can be tested in two ways, destructively or non-destructively. In destructive means of testing materials, a sample is selected randomly from a batch and is inspected thoroughly and the results are extrapolated in a manner so that they represent the whole batch. Since there is a need of high performance parts to be inspected while they are in use, this method of testing cannot be used. Hence, there is a need for alternate means of testing these type of materials which is answered by nondestructive testing.

Nondestructive evaluation (NDE) is a way by which one can inspect a material or a structure without disrupting its serviceability. Aircraft industry is a prime example which shows the prime importance of NDE. With newer light weight materials being used for the manufacture of the aircraft, there is a need for a more strenuous testing procedure so that the parts are not compromised while they are in use. Also, the inspection systems must be capable of handling a variety of materials like aluminum and nickel used in the manufacturing of aircraft. Also, the inspection systems must be fast enough while keeping the cost associated with them to the bare minimum.

Primarily, three inspection methods of testing a material are available at the Center for Nondestructive Evaluation: X-ray radiography, ultrasonic testing and eddy current

testing. The idea behind all these techniques is the same. Energy is being introduced into the sample from a source to interact with the sample. This interaction is characterized by a detector to detect any flaws in the sample. Eddy current techniques are used for determining the surface cracks and their length in materials. Ultrasonic technology is used for in-service inspection of composite panels for delamination and detection of crack problems, process control monitoring, and to detect voids in materials. Ultrasonic and eddy current methods of non-destructive testing work fine for objects with few facets and uniform composition, but the present day objects with complex shapes with multiple facets present a challenge to these techniques and they often fail in detecting voids or defects in such materials. Hence, X-ray based NDE techniques have remained a prime means of testing complex castings. My research primarily focuses on this aspect, X-ray radiography and more precisely, real-time radiography.

In X-ray radiography, a source would generate a beam of high energy electrons which are made to collide with a target thus producing a beam of photons whose energy will be in the X-ray energy range. In a typical instance of characterizing a material with X-rays, a sample will be kept in between an X-ray generator and an X-ray sensitive film. Depending upon the density and various other properties of that material, some of the X-rays will be attenuated, some will be scattered and some will be passed through the object under inspection. These X-rays which are passed through the object after some attenuation form a latent image which is captured onto a sensitive film.

Since X-ray inspectability is extremely dependent upon crack opening and orientation, there exists a need to quantitatively assess the detectability of cracks for various crack geometries. But, several disadvantages are associated with film radiography; the amount of time it takes to develop the film and the cost of the film. Real-time radiography, which uses an X-ray to light converter device (image intensifier) proves to be a better method of testing these cracks compared to film-radiography [13]. This is due to the fact that one can see a scanned image as soon as X-rays are passed through the

sample unlike in film-radiography. Also, crack orientation plays a crucial deciding factor of these two methods of radiography. One can obtain good results with film radiography if one has prior knowledge about the crack opening and orientation. Since, in real life this is not available, there is every chance that the crack can go undetected if the crack is not aligned properly with the X-ray beam. For such alignment, one has to move the sample in various directions and real-time radiography makes it much faster thereby increasing the detectability of the crack compared to film-radiography. But the advantage of film radiography over real-time radiography is its ability to store in archival records for a long time.

In order to expedite the X-ray imaging process, a good approach would be to absorb the transmitted X-rays and convert them into light using an X-ray to light converter rather than using a film to absorb the radiations. This is called real-time radiography as the radiations are absorbed, converted to light and the resultant image is displayed onto a monitor using a camera instantly unlike in film radiography. This method of inspection along with film radiography is carried on at the Center for NDE. Figure 1.1 gives an outline of the real-time radiography inspection method [4]. The sample to be inspected is placed on the sample positioner between the X-ray source and image intensifier as in film radiography but instead of a film, an image intensifier is used. An image intensifier absorbs the incident X-rays, converts them to visible light and amplifies that visible light which can either be seen by a human eye or can be recorded by a CCD camera. This camera in association with a frame grabber and a host computer can be used to enhance this light and do some image processing of the resultant image from the camera output to obtain a better picture. The host computer will control the motion of the sample positioner and also the image acquired.

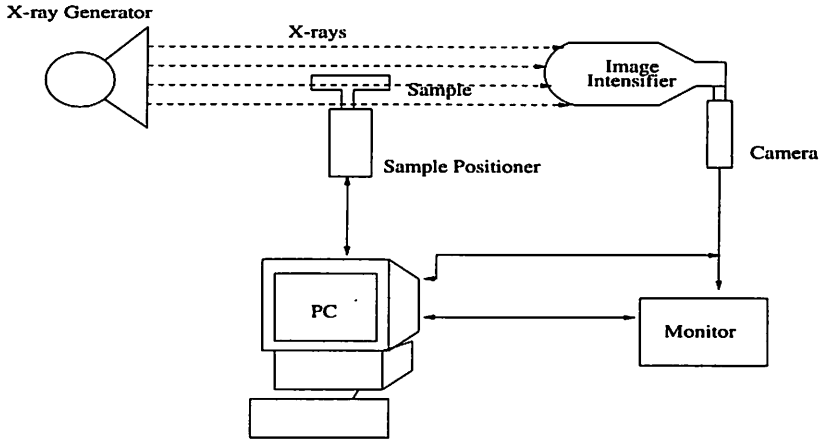

Figure 1.1 Figure showing the basic setup needed for real-time radiography

1.1 Motivation and Goal of this Research

The width and the orientation of a crack play a crucial role in its detection. Many times while scanning a material to detect cracks in it, it happens that the X-ray beam is not aligned properly with the crack opening. This might result in the flaw going undetected even though sufficient equipment are available to detect the crack. An aircraft presents a prime example for these kinds of cracks. Whenever an aircraft leaves ground and flies high into the air, it will be subject to repeated cycles of stress on its body due to the pressure difference in the cabin and outside atmosphere. Again while landing, the external pressure will be the same as inside and this process of take-off and landing of the aircraft takes place many times in its life-time, subjecting its body to heavy stress every day and the result is the formation of the cracks on its body. This is fatigue in the areas joined by rivets. These rivet held metal plates tend to separate due to this stress and they start forming cracks as an indication of this. These cracks will tend to grow linearly until a point, and very rapidly after that. This forces the industry to detect such sort of cracks at an earlier stage before they tend to grow rapidly. More pronouncedly,

these cracks are shown in aging aircraft which are being used beyond their design life. The capability of detecting such sort of cracks before they reach the point they tend to grow rapidly enables one to replace the defective part before the aircraft is damaged any more. Eddy current techniques produce good results for plain surfaces, but if the inspection area contains rivets, they produce results which aren't so obvious. Eddy current signals for a scan taken in the vicinity of a rivet will have the same strength both for the rivet opening and the crack under the rivet. X-rays provide reasonable results for such sort of cracks though they are limited by the size of the crack under the rivet head. The width and the angle of the crack opening become the key factors in detecting the crack.

In order to detect such narrow cracks one might need to either translate (move in x,y and z directions) or rotate the sample or source in different directions for aligning the crack opening in the direction of the X-ray beam, so that the crack can be seen on the monitor. In this process of aligning, there is every chance that one might repeatedly be doing the same thing again and again, like, moving the sample in a particular direction with all other parameters like the kV and mA for generating X-rays remaining the same. This obviously is a heavily time consuming process. Also, since it is not possible to move a large sample like an aircraft on a typical sample positioner, it is well worth the effort to have the capability of moving the source too, in which case there is no need of moving the sample. In order to expedite the process and also to decrease the amount of monotonous work involved in this procedure of detecting the flaws, it would be a good idea to automate this entire process of the movement of the sample and also the source in order to align the crack opening in the proper direction to be able to detect the crack of reasonable size. The basic idea behind this project is this automation of the entire procedure. This thesis explains how this has been implemented.

1.2 Organization of the Thesis

This project deals with the basic automation of the otherwise manually performed functions used in real-time X-ray radiography. Chapter 2 gives the general background on stepper-motor control and the existing laboratory set up. Also, it explains the two software packages XR Vision and RMover. Chapter 3 deals with the drawbacks of these two packages and what could be done to overcome their principal drawbacks. It also explains how the existing two packages have been incorporated into one with lots of additional features included in it. Chapter 4 provides a sample scan file for automatic scan control and explains how the sample scan file is being read and handled by the program. Finally, Chapter 5 provides the experimental results obtained with this program and compares them with the results obtained with the existing programs. It also includes the conclusion of this thesis with future developments in this field.

2 LABORATORY SETUP AND LIBRARY FUNCTIONS

This chapter will cover the aspects and details of the two principal packages in use at the Center for Non Destructive Evaluation for real-time radiography. The first one is the motion control program (RMover) responsible for the motion of the sample positioner and the second one is the XR Vision program, the image acquisition software package. A basic picture showing the setup of the machines in the laboratory and a brief explanation of the packages will present a clear picture of the functioning of these two packages.

2.1 Laboratory Setup

Figure 2.1 depicts the current laboratory setup. The primary part of the setup is an X-ray generator or source which generates the X-ray beam. The source contains a heated element and a target. Whenever the electrons generated by a heated filament are accelerated through a very high potential region on the order of a few hundred kilovolts and made to hit a target like tungsten, they produce bremsstrahlung X-ray photons. An X-ray tube [6] is designed in such a way that it accelerates electrons in vacuum and makes them collide with a target of high density and high melting point like tungsten. An X-ray generator is the combination of such an X-ray tube and an electrical apparatus designed to generate and control electrical energy for applying to the X-ray tube. The existing laboratory contains two X-ray generators, a broad focus IRT320 and a micro focus FXE-200.50. The two generators differ from each other in their maximum level of the accelerating voltages, their beam spot size and also in their power levels. The

accelerating voltage is in the range of 20-200kV for FXE-200.50 and in the range of 20-320kV for the IRT320. The beam spot size and power level of FXE-200.50 are 10μm and 200W respectively. Whereas, for IRT320 the beam spot size and the power level are 1mm and 3200W, respectively.

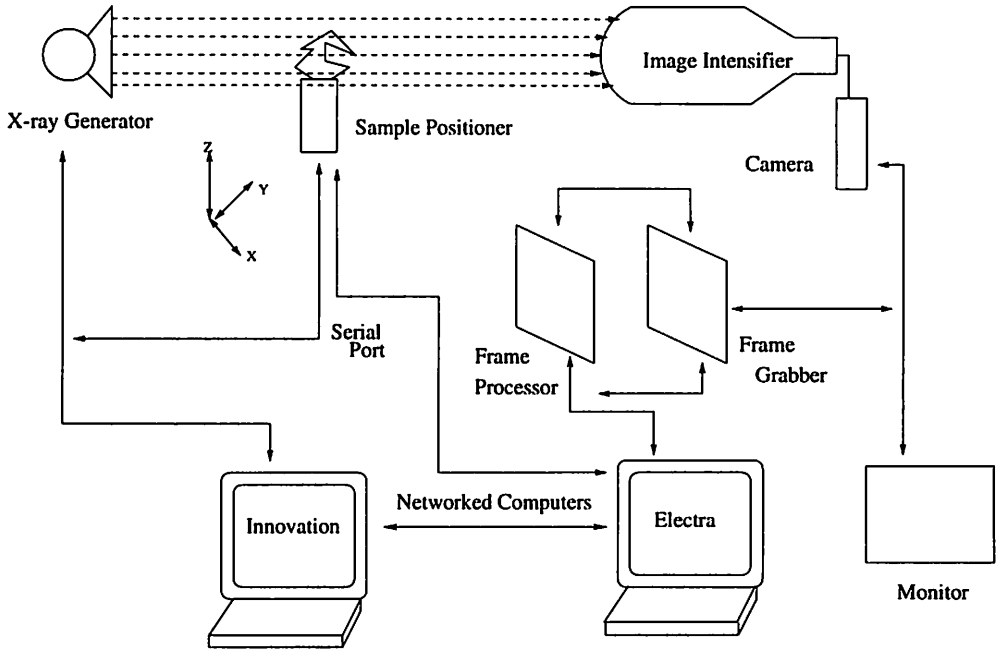

Figure 2.1 Laboratory setup

The sample positioner (from DAEDAL Inc.) serves as the pedestal for any object under inspection and this can be translated either in the x,y, and z axes or can be rotated in ϕ and θ. The positioner is being controlled by two PCs with the PC23 Indexer cards [1] serving as the interface between the computer and the motors. A Compumotor PC23 is a microprocessor-based indexer that is designed to be inserted into a single open slot in the computer expansion bus or I/O channel. Manufactured as a single board, the PC23 can control up to three axes of any Compumotor system. The PC23 uses a 16-bit processor with custom circuits to simplify generation of motion profiles and shorten the

processing time. The PC23 receives acceleration, velocity and position information from the computer in ASCII format. Two PC23 Indexer cards are used to handle all the 5 different axes.

The two PCs are connected by a network through Windows for Work Groups 3.11, and two way communication between these two computers, one serving as a server and the other as a client, is possible. The PCs run MS-Windows version 3.11 operating system. The two PCs are named Electra and Innovation. The need for connecting these two PCs arises due to the limitation of the IRQ and DMA controls. For the motion control program Innovation serves as server and Electra as client. The PC23 indexer card on Electra serves for the movement of the sample positioner motors in θ and ϕ directions, whereas the one on the Innovation controls the motion of the motors in x, y, and z directions.

An image intensifier is used as an alternative to a radiograph for acquiring an X-ray image in real time. An X-ray to light converter in the image intensifier forms the core part of the real-time radiography. The absorbed image from the image intensifier can either be seen by a naked eye (image would be very small) or can be visualized using a CCD camera connected to the image intensifier.

A monochrome CCD camera converts the optical image into a video camera signal (RS-170) which is further digitized by the frame grabber. A primary disadvantage of a CCD camera is the small size of its sensitive surface. A lens is required to focus an image onto the CCD element. A typical CCD will have 512 X 512 pixels with each having a size of about $20X20\mu m$. Each element of the CCD imager is sensitive to light and each pixel stores the charge proportional to the input light and there is also a maximum limit on the charge collecting capability of each element.

An 8-bit frame grabber on a single IBM PC AT plug-in board (Data Translation model DT2867) with built in processing circuitry for real-time frame averaging and two frame arithmetic and logic operations is used to digitize the RS-170 signal. In addition to

a 16-bit image buffer for video-rate true frame averaging, the board has two 8-bit buffers for secondary image storage and display. The main advantages of the DT2867 board for being used as frame grabber are real-time image capturing capability, selectable input gain and offset and the ability to accommodate low light signals with full resolution. The frame digitizer is a versatile piece of hardware which can be accessed remotely by the existing XR Vision program (which will be discussed later in the chapter).

Due to the high energy radiations from the X-ray generator and scattered radiations, the X-ray generator, sample positioner, image intensifier and the camera are all enclosed in a radiation shielded vault. The interlock facility provides another level of safety which ensures the X-ray generator can be turned on only when the door has been properly shut preventing the people outside of the room from receiving any unwanted radiations. The X-ray generators and the sample positioner can be controlled remotely by the host PC via a serial link and from any other PC by Winsockets. The host PC has been connected to the network by an Ethernet cable with a unique IP address for network access.

2.1.1 Network Access and Control

For the smooth movement of the sample positioner in different directions the two PC23 indexer cards should be accessed. But owing to the limitations of the IRQ and DMA controls, the two PC23 indexer cards cannot be plugged in to Electra which currently holds 6 cards in its back plane. Since a PC can access cards only with three IRQs and since two IRQs have already been taken away by other boards leaving only one IRQ control for controlling the two PC23 indexer cards. Hence there is a need that one PC23 indexer card be plugged into a different PC and must be networked to access it. So, the two computers Electra and Innovation hold one PC23 card each, and are connected to the network through Ethernet cards enabling the remote control of the hardware connected to these two PCs. The network software can even transfer the data collected on one machine to be saved on a remote machine [11], [12].

The network access can be obtained through Microsoft's Winsockets 1.1 [9] running under Windows for Work Groups. Windows supports four mechanisms of interprocess communication :

- dynamic data exchange (DDE)
- shared memory in dynamic link libraries (DLLs)
- clipboard
- Winsockets

Of all the four mechanisms only Winsockets can be used for communication between two different computers which are connected to the network by Ethernet. A Windows program is an executable file that generally creates one or more windows and uses a message loop to receive the user input. A server is a program which has the data and is willing to share the data with its clients. DDE [10] is a message based system in which data needs to be transmitted with messages. Since, Windows allows only two parameter values, wParam (WORD) and lParam (DWORD) for data transfer one can use DDE for sending messages which are not very big in size like sending PC23 Indexer card commands. But, using DDE, there is not enough room to transmit information on the order of a 512 X 512 image. One way to send the data would be to break the image into pieces to transmit it, but this would take a long time to send an image across the network owing to the fact that data can be sent at a maximum rate of 8 bytes at a time. So, DDE fails to serve the purpose for this sort of application. DLLs can't be used as the data can be transferred only between stacks on the same machine. Clipboards allow data (which is in a particular format such as a text, a bitmap or metafile) to be transferred from one program to the other. When a program uses clipboard for data transfer, it copies the message or text or file that needs to be transferred into a memory area. It allocates a global handle to this area. The program then opens the clipboard

and passes the global handle to it thus transferring the data. The disadvantage in using the clipboard is that the memory handle, generated while transferring the data, cannot be used by the calling program once it is passed on to the clipboard. Hence, to use the memory handle again, the program has to allocate memory and a handle to it again, thus consuming unnecessary memory twice. This leaves us with the option of using only Winsockets for the network data transfer. Also, Winsockets are initialized on the Electra and Innovation machines [11]. Hence, it will be easy to do the network transfer using Winsockets, rather than implementing something new from scratch which has a little advantage over Winsockets.

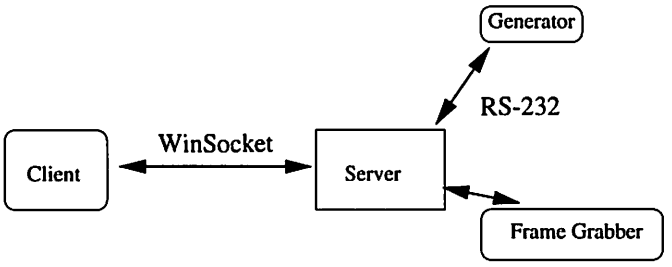

Figure 2.2 Client/server model

2.1.1.1 Design issues

The standard model for network applications is a client/server model and is shown in Figure 2.2. The client and server require a set of standard protocols to enable them to establish a connection between each other. The server will be running continuously and will listen at the well known address and "wakes up" at the request of the client for a connection. To establish this connection, each process establishes a socket system call which returns an integer value socket identifier, SID, the same as a file descriptor,

identifying the connection. The socket call would even specify the type of the communication protocol supported by it. A structure has been defined for windows sockets to be compatible over different protocol families, each with different address formats and address lengths. The five parameters of the structure; namely, the protocol, local address, local process, foreign address and foreign process, must contain valid data before any data transfer can take place between the client and the server. Any process running on a PC can be identified with the Internet address of that PC and the process ID obtained from the operating system. This allows two processes to communicate via Winsockets with each containing a valid five tuple structure.

Figure 2.3 shows the flow chart for the server software [7]. Whenever the client wishes to communicate with the server, it will execute some system calls. The first step toward this will be to create a socket, and for this the client executes the socket system call. The socket call would assign the protocol field of the above discussed five-tuple structure. Once the type of the protocol has been specified, the server "binds" a socket descriptor returned by the "socket" system call to a protocol specific address and also lets the system know that this address is owned by this process and any messages received for this address are to be given to it. The server would "listen" on this socket for an active connection from a client. The server then will execute an "accept" system call to let the client know that the server has processed and accepted its request. Using the "connect" system call, the client process connects with the server and will assign a socket descriptor to identify the connection.

2.2 RMover: Current Motion Control Program

This program gives the user a high level control of positioners using a PC23 Indexer card and MC5300 controller. The existing motion control program RMover, connects two different PCs on the same guidelines as described above. Here, Electra functions

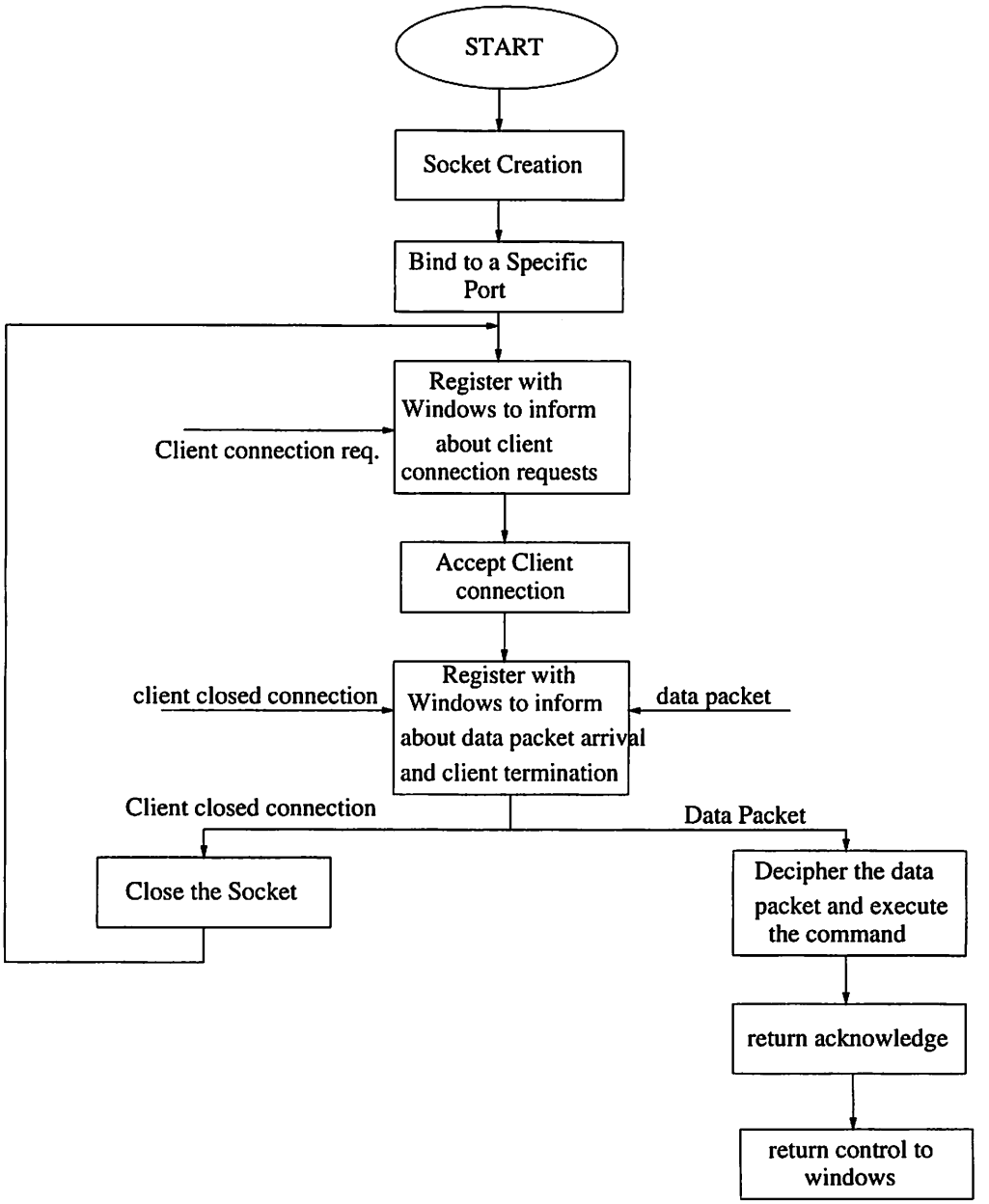

Figure 2.3 Flow chart of a Windows based non blocking server

as client and Innovation as server. If the client has to access the PC23 indexer card on the server, it should have a source module [7] that will implement the network access of the PC23 card in the main program. The RMover program has a C++ [8], [16] class called **Motor** declared in its header file Motion.h, for accessing the PC23 card on the local machine and another class **Rmotor** declared in Rmotion.h (shown in Appendix D) for accessing the PC23 Indexer card on the remote machine or on the server. A class contains three different sections called private, protected and public. Each of these sections contains different members referred to as member functions. The private and protected member functions of a class are not visible to any program outside of that class and they can be accessed only by the public member functions of that class. Also, each class will have a constructor which will build an object with all the attributes of that class by taking in the necessary parameters. The member functions of the class can be accessed by calling "object.memberfunction". One can add as many member functions as they want by simply declaring them in the class and then writing the code for that declaration.

As soon as the program is executed, the client would try to connect with the server whose Ethernet address will be provided in advance. Once the client is connected to the server, the data transfer between the server and the client will take place in standard packet size. The PC23 Indexer card [1] accepts commands in ASCII format from the computer and will send them to an adaptor box through a cable. This adaptor box will send appropriate commands to the drives of the corresponding axes (each axis will have a separate drive).

The program will read a configuration file called motors.cfg (a sample file is shown in Appendix A), which contains all of the details pertaining to the motors, such as the number of motors, their PC23 Indexer card addresses and the name of the axis they correspond to. Upon reading from the configuration file, the program will create a standard interface structure, one for each motor. The interface structure contains the

name of the axis, edit boxes for acceleration, distance to move and velocity settings. Also, there is an option to choose whether one wishes to move the positioner in a particular axis by a fixed step size or to move it continuously until the user kills the motion explicitly. The motion of all the motors can be stopped at once by pressing the END key at any instant. The existing system has the PC23 Indexer card controlling the x, y and z axes on the Innovation and the PC23 Indexer card controlling the ϕ and θ axes on the Electra PC. The software which governs both of these machines and the network access is resident on the Electra PC.

An example from one of these classes will provide a better understanding of the above discussed concepts. Let's consider a member function move() from the class **motor**. An "object" has to be constructed before any of the member functions in any class can be used. Since the motors responsible for the θ and ϕ direction of motion of the sample positioner are located on the local PC, one has to use the **motor** class for constructing objects to control the motors in these directions. Let's say we want to move in the negative ϕ direction. So, we have to pass the name of the axis and the address of the PC23 Indexer card to be used for the constructor to build an object governing the motion related to ϕ. Once an object has been constructed for motion in the ϕ direction, all the functions one would like to perform in the ϕ direction can be addressed to this object. The velocity for moving the sample positioner in this direction has to be set before it can actually be moved. It is set by calling object.setvelocity(float(velocity)), where setvelocity is the member function of the **motor** class. Acceleration of the motor is set in the same way as the velocity with the only difference being that the member function to be used in this case is setacceleration(float). Now the direction of the motor has to be set and since we wanted to move the positioner motor controlling the phi direction in the negative ϕ direction, the direction of the motor is set by calling setdirection(char *) where the char * can be ccw standing for clock-wise direction or cccw for counter clock-wise direction of the motor, and for our purpose this should be set to cccw. Then

one can use either "move" or "move(float)" member functions for moving the motor continuously or for a fixed step size, respectively. A list of the member functions of the **rmotor** class has been provided in Appendix D. These commands can be used in controlling the motion of the remote motors. The local **motor** class commands are the same as the remote motor class commands with the only exception that each of the classes will have a different constructor and the **motor** class doesn't have any network related commands unlike the **rmotor** class.

The **motor** class has been programmed in such a way that it can handle only objects whose PC23 cards are on the local machine. Since the PC23 Indexer cards which control the motion of the x, y and z motors are on a remote machine, another class has to be used and hence **rmotor** has been implemented. This class will handle objects on remote machines, provided one supplies the proper Ethernet address while constructing an object for that machine. So, if one wants to move the sample positioner in positive y direction, firstly an object or an instance has to be created on which one would like to perform some operations. So, an object, let's say remote, is created by passing the appropriate parameters for the constructor of the **rmotor** class. Before an object which has to remotely accept the data from another PC can be created, proper network connection and also reliable data transfer must be ensured. Once the object has been created, all the operations on this object will send appropriate commands to the corresponding hardware specified while constructing it. Once the instances or objects for moving in one of the x, y or z axes have been specified, the functioning of the **motor** and **rmotor** classes is one and the same with the only difference being that an object can use only the member functions of that class which has been used while creating it. The member functions of the **rmotor** class can now be accessed by "remote" and since we wanted to move in the positive y-direction, we need to set the velocity and acceleration in the same way as we did for θ. The direction of the motors is being set by calling setdirection(char *), where (char *) can be set as ccw for clockwise direction of motion

or cccw for counter-clockwise direction of motion of the specified motor. Here it will be set to ccw as we wish to move the positioner in the positive y direction. By using one of the member functions, "move" or "move(float step)" we can move the motor either continuously or by a fixed step size respectively.

2.3 XR Vision: Image Processing Hardware/Software

XR Vision [14], [15] is an image processing hardware/software system which can be hosted by any PC running Windows 3.11. XR Vision is capable of real time image acquisition and processing. To perform this, the system contains a special purpose computer in addition to the host PC. This computer is Data Translation 2867 frame grabber [5]. Data is passed between the host and the DT2867 via the AT bus which provides dual functionality, the first being control of the frame grabber and the second one is file I/O for the frame grabber. MS Windows provides a logical and flexible user interface [10] to all components of the system. XR Vision can be configured simultaneously for real time, near real time and PC based image processing by simply implementing the appropriate control structure in MS Windows and hooking that structure to a MS Windows menu item [2].

By choosing a menu item, the user initiates a control structure which starts and stops operations in the hardware. If for example, a user has a custom filter written in C or assembly language he could implement a control structure in Windows to automatically acquire an image and return the image to the frame grabber for additional processing or for display or for file I/O.

There are currently two stationary cameras used in this program. Two cameras are connected to the input of the frame grabber for getting the image. With the help of this software package, one can grab a real time image and can do some processing to get a better picture out of it.

Average and accumulate are the two methods primarily used in this program for acquiring an image from the camera. In the average procedure of acquiring an image, the output of the frame grabber will be captured for a set of frames and then the sum of the individual pixel values for all the frames is divided by the number of frames to get an image which is an average of all these frames. The procedure for accumulating an image is the almost the same with the only exception being that after adding up all the pixel values, there wouldn't be any division by the number of frames. The grey scale value of each pixel of this image is the result of summing up of the individual pixel values for each frame. XR Vision program handles both 8-bit and 16-bit images. The average part of the program can handle only 8-bit images whereas the accumulate part of the program can process both 8-bit and 16-bit images.

The accumulate part of the program helps in reducing the noise signals generated out of the CCD pixel elements and also to increase the contrast of any cracks in the material. An 8-bit image out of the frame grabber is accumulated over a number of frames in a 16-bit accumulation buffer which can be used for image processing. Even though the accumulation buffer can hold 16-bits of data per pixel, the display of such an image is limited by the fact that there aren't any commercial VGA display monitors which are able to display 16-bit images. Hence, the images have to be normalized and converted into 8-bit images before they can be displayed.

3 XRSCAN: THE NEW SOFTWARE DEVELOPED

The current versions of XR Vision and RMover programs are stable versions currently hosted by a PC running Windows 3.11. If one wants to use these programs for acquiring multiple scans, then the amount of time involved in such sort of scans will be prohibitive. If the amount of manual intervention in such sort of scans is negligible, automating the entire scanning procedure would be good idea. The following section explains the principal drawbacks of the two existing software packages and the latter part of the chapter will describe how they have been overcome.

3.1 Drawbacks of the Current Programs

Although RMover is a stable program, the principal feature lacking in that system is the display of the sample positioner's relative distance from a reference point. If one doesn't know how far the positioner has been moved in a particular direction, then one can never be sure whether the positioner has moved for the distance one wished to move it by. This feature of displaying the relative distance by which the sample positioner has moved in each direction is a primary feature an end user wants to be incorporated in the program. Each of the sample positioner controlling motors has a home switch which when enabled will preset a reference position for each motor and whenever a motor is sent to go home, the motor will move depending upon its current position until it encounters the reference position, termed as home.

Using the existing XR Vision program, one can specify the minimum and the max-

imum limit of the 16-bit data to be selected and converted properly to display on the VGA monitor. These two variables can be changed with the help of two scroll bars which are independent of each other. Since the two scroll bars are independent of each other, the selection of the proper values for these two variables is done by the trial and error method which not only is a time consuming process but also depends upon how lucky one is.

One more thing of prime concern is that these two software packages are separate making it difficult to use them simultaneously. Hence a good program will be one which will incorporate these two programs into one with all the functions in the two software packages included, thereby enabling the user to perform all the functions under a single interface. The current research provides such an interface where the user will have the control over the sample positioner using the motion control part of the interface, over image processing under the image acquisition part and over the X-ray generator settings in the X-ray generator section of the interface.

The current project also takes into consideration another aspect of the real-time X-ray scans, automation of real-time X-ray scans. If an object has to be scanned for a number of times with little change in many of the parameters associated with the motion controller and X-ray generator settings, the amount of time it takes to finish such a job manually will be too prohibitive. Reducing the amount of human intervention in such kind of jobs to the bare minimum will present the end user a high degree of flexibility and uniform repeatability. If there is a file where in an end user can type in all the commands which he otherwise had to perform manually, a program can accept the commands from this file and execute one command at any point of time. The time involved in this method of testing will be decreased and also the testing is done automatically without much human intervention. This aspect of automation software has been developed to handle this and has been incorporated into this package. This will be explained in greater depth in the next chapter.

3.2 Software Development

The rest of the chapter will cover the aspects of the new software program in terms of its user interface and background actions (what happens if one clicks a button on a particular window) related to the sample positioner, image processing and X-ray generator controls. Figure 3.1 shows the parent window which will be displayed upon the execution of this program. Clicking on the menu item, Real_Time SetUp, will invoke a dialog box [10] partitioned into three different sections: namely, Motion Control, Image Acquisition and X-ray Generator Options as shown in Figure 3.3. The other menu item, Real_Time Scan, when selected will initiate a window which will have controls for automated scans (the prime objective of this research) and is shown in Figure 4.1. This other aspect of the program is explained in detail in the next chapter.

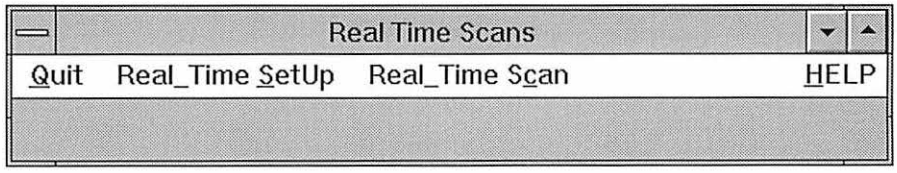

Figure 3.1 Program main window

3.2.1 Motion Control

The motion control part of the control program builds two types of objects for controlling all the 5 motors. The program will read a configuration file, motors.cfg (shown in Appendix A), which contains the details pertaining to the motors with a line for each of the motors. For each motor, the details will be whether the PC23 Indexer card controlling the motor is on the local machine or on the remote machine, address of the PC23 card and the number of the motor. Once the file has been successfully read, two

types of objects will be constructed, one for remote Indexer cards using the **rmotor** class and the other one for local PC23 Indexer cards with the help of the **motor** class. Thereafter, one can use all the functions of the above two classes on these objects (note: objects built using a particular class can use only the public member functions of that class) for the proper movement of the sample positioner. Figure 3.2 shows the axes the sample positioner can move with directions indicated by arrows.

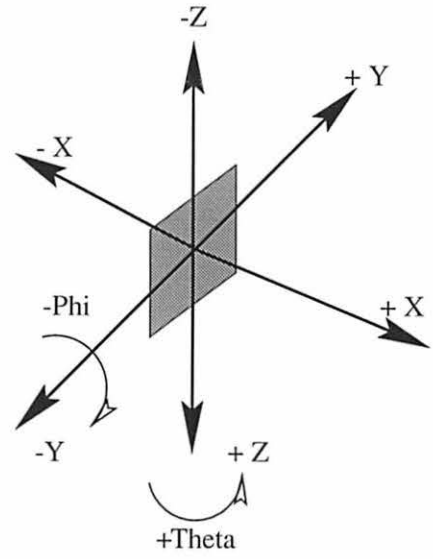

Figure 3.2 Direction of the axes

The existing motion control program as discussed in section 3.1 doesn't have any option displaying how far the sample positioner has moved in a particular direction after a command has been issued for it to move by some distance. This capability has been incorporated in the present program as shown in the *Motion Control* part of Figure 3.3. The edit boxes [2] under the *Relative Position* title show the distance the sample positioner has moved in a particular direction for each of the five directions. This program has the capability of updating the distance while the sample positioner is in motion. The current position of the sample positioner when the program is started

will be treated as the reference position for all the motors for all the directions. If the sample positioner is moved by some distance in a direction, the relative position fields will display the distances from the reference position to the current position of the positioner with continuous updates. If the NEW REF button is selected, the program will mark the current position of the sample positioner as the new reference position for all the directions. From there on, the distances are measured from that new reference position for the sample positioner. The program also features a *H* button which upon clicking will make the corresponding motor go to the home position (a fixed reference position). There is an individual button for each of the motors to send it home.

Now comes the main part of the motion control section of the program. Under the *Distance* title, there are five edit boxes wherein one can enter a number specifying the distance by which one would like to move the sample positioner in a particular direction. This mode of motion is called fixed mode. Clicking on the *C/F* buttons makes the corresponding motors toggle between the continuous and fixed modes. In the continuous mode of motion, the sample positioner will move in a particular direction until the user stops the motion of the motors explicitly. This can be done either by pressing the *STOP* button or by pressing the right mouse button. The program also features a *P/R* button for each axis which a user can use to pause the motion of any motor in any particular direction. This button toggles between the pause and resume modes, where in the former case, the motion of a motor in a particular direction will be paused till the user explicitly clicks on the corresponding resume button. This feature has been incorporated to give a high degree of freedom for the user.

The + and - buttons next to each edit box of distance control will provide the motion in the positive and negative direction of an axis respectively. By clicking on a + one can move a motor in the positive direction of the corresponding axis, and in the negative direction by clicking on the - button. The *Motor Options* button, when clicked, will invoke a window (shown in Figure 3.4) with fields for typing in the required speed and

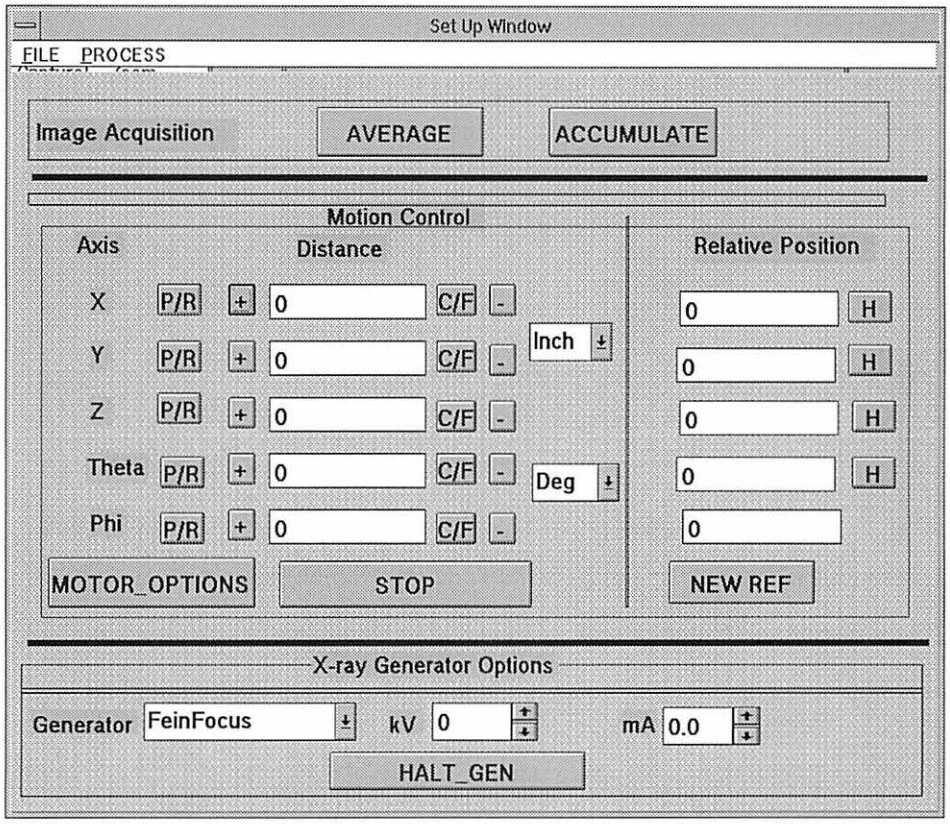

Figure 3.3 Setup window incorporating the three different packages

acceleration for each axis. Using this window, one can even select between the metric and English system of measurements. This will enable the user to have the capability of moving the x, y and z motors either in inches or cms and the ϕ and θ either in milli-radians or degrees.

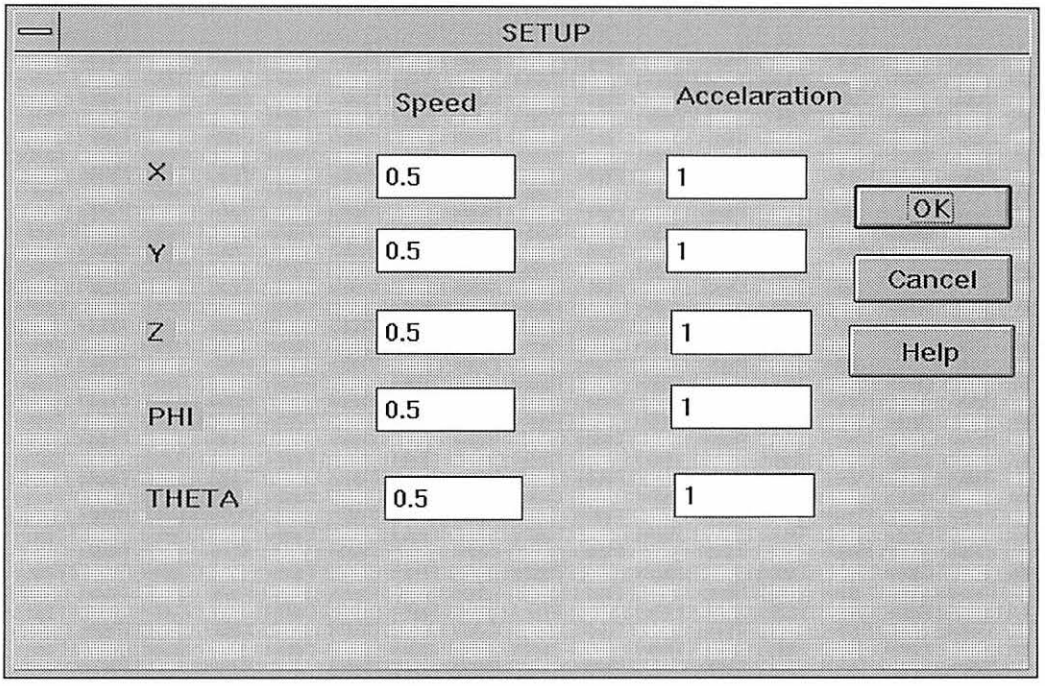

Figure 3.4 Options window for speed and acceleration settings

This option of selecting between the English and the Metric systems is for the convenience of the user to specify the units of motion. Since the motors can accept only a number specifying the number of steps it has to move, it is up to the programmer to convert either the metric or English system units into appropriate number of steps and send it to the corresponding motor.

3.2.2 Image Acquisition

This part of the program will control the image processing part of the scan. Much of this part has been derived from the existing XR Vision program for image processing [14], [15]. The Image Acquisition part of the program features two individual buttons titled Average and Accumulate (see Figure 3.3). The Average part of the program can handle only an 8-bit image whereas the Accumulate part can process both the 8 and 16-bit images. Two cameras serve the imaging needs at the center; namely, scan camera and surveillance camera. The image acquisition part of the program reads the camera configuration file (provided in Appendix A) to list the available cameras in the appropriate combo boxes of figures 3.5 and 3.6. This facilitates adding other cameras to the system with appropriate port numbers for each of them. With the help of the surveillance camera one can see inside of the X-ray room to make sure everything is intact. The scan camera is used to collect the radiations for real-time image processing. The port numbers of each camera specified in the camera configuration file are stored in a structure. The program reads the structure to get the correct port number (or channel number) for a selection in the combo boxes.

3.2.2.1 Average

The window which will pop up when the *Average* button of the setup window is is pressed, is shown in Figure 3.5. It features a combo box [10] which enables the user to select between the two available cameras (*scan camera* and *surveillance camera* in the present situation). The *No.of frames* field in the window will be used to set the number of frames one would like to grab and average. The default value of this has been set to 20. This value can be set to any number between 1 and 256.

Upon selecting the *acquire* button of the average window, the program will read the number of specified frames and will divide the sum of the individual pixel values (for the

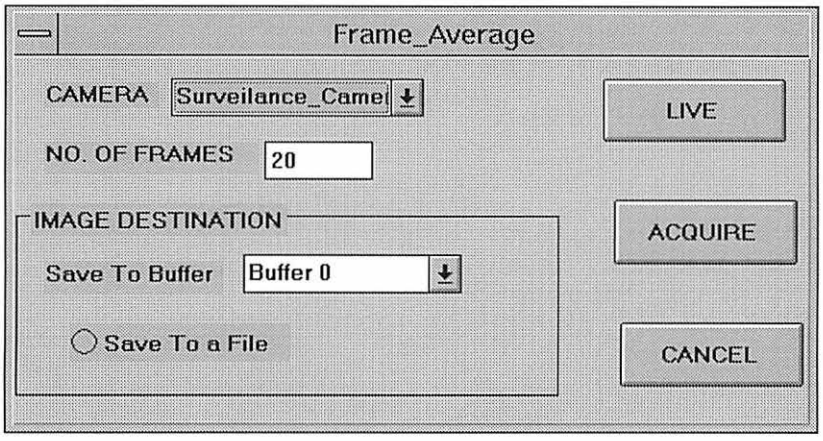

Figure 3.5 Interface of the frame averaging window

number of frames specified in the window) by the frame count specified, to obtain an averaged image. The averaged image can be saved either into a buffer or into a file at the same time. There are two 8-bit buffers on the frame grabber board which are referred to as buffer0 and buffer1. Selecting the *save to a file* option would open a dialog box prompting one to type in the name of the file one would like to save the image into. The *LIVE* button when selected will connect the display monitor to the camera selected in the combo box and will give constant video signals from the camera to the VGA display monitor at the standard rate of 30 frames per second.

3.2.2.2 Accumulate

The accumulate portion of image acquisition controls the processing of images of 16-bit length and 8-bit length as well. Figure 3.6 portrays the interface of the accumulate window which will be popped upon clicking on the *Accumulate* button in the *Setup Window*. The camera selection, number of frames and the LIVE buttons when selected will perform exactly as the corresponding ones in the *Average* window. Another primary

feature of this program is the ability of displaying the histogram of the grey scale values of the pixels in an image. The histogram window is activated upon selecting either the *GRAB* or the *SUBTRACT* buttons. When a user acquires an image by accumulation, it will be saved into one of the 16-bit buffers and it is this data which one uses to display

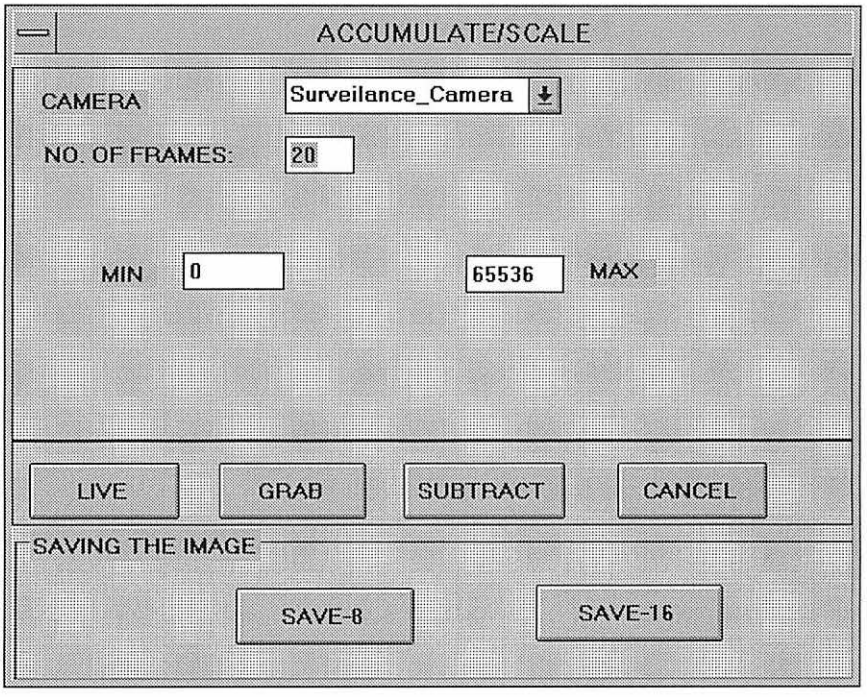

Figure 3.6 Accumulate window

the histogram. A histogram (shown in Figure 3.7) is a graph with the pixel count on the Y-axis and the pixel grey scale value on the X-axis. The saved data in one of the buffers can range anywhere from 0 to 2^{16}. Since a PC monitor can be able to display up to a maximum of 640 pixels on a horizontal scale, there must be a way of translating this data presenting it conveniently on the computer monitor in the form of a histogram. This is done by using the histogram library functions supplied with the DT2867 board. The sensitivity of real-time X-ray inspection system is based on several factors [15]; spatial

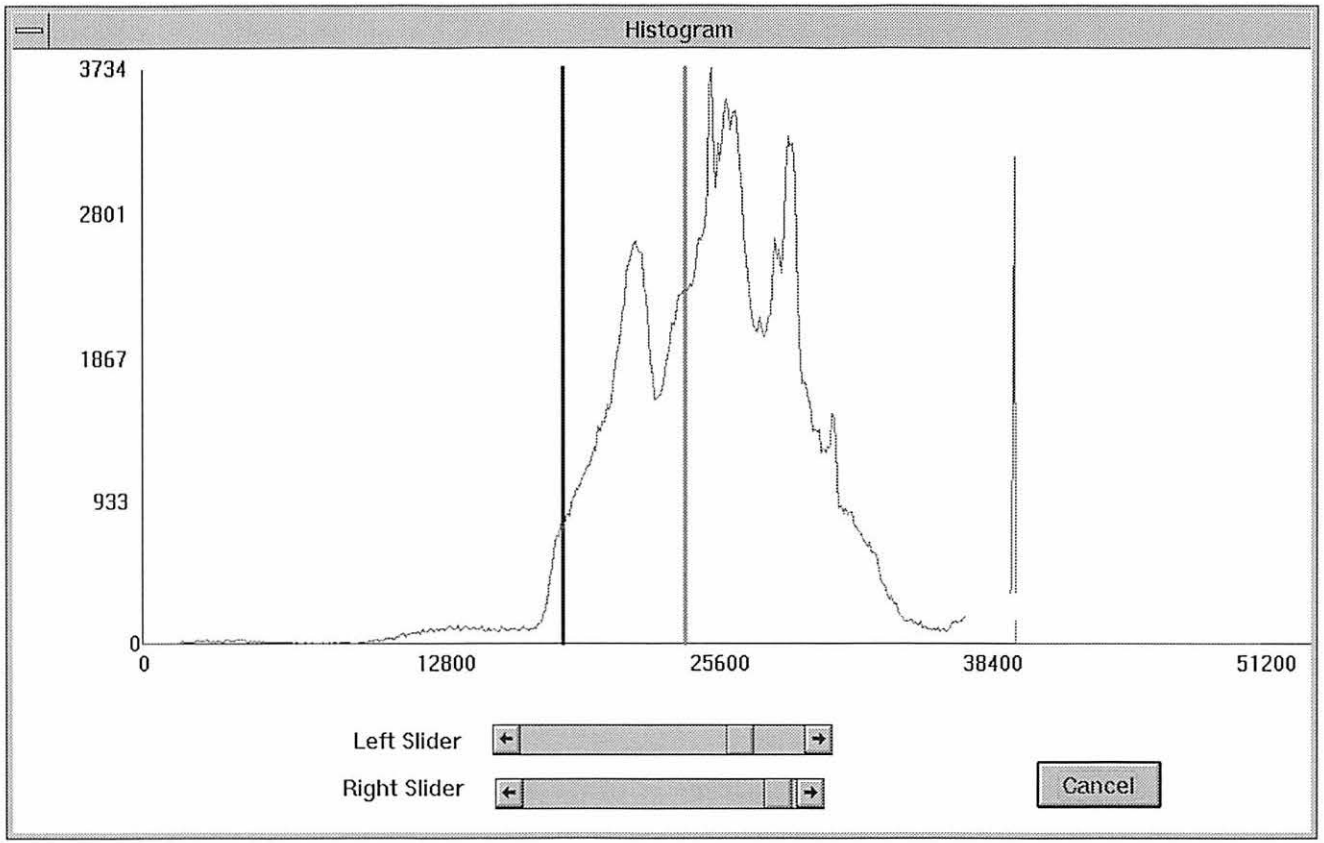

Figure 3.7 Histogram of an image accumulated over 256 frames

variation in the response of the image intensifier and CCD camera are the prime ones. There are two types of variations limiting the sensitivity of real-time X-ray inspection systems; systematic and random variations. Due to the random variations, a single accumulated image is not a true representation of a sample and an image acquired over a specified number of frames will eliminate them. But, the systematic variations are inherent in the hardware. In order to counter them, images of a sample are accumulated over a specified number of frames and then subtracted with the incoming measurements. If the material is uniform all over, then subtracting the image from the accumulated image should result in a perfect zero scale value for each of the pixels. So, in order to make the crack stand out, the material is moved a little (upwards or downwards) and subtracted from the incoming images of the sample from its new position. Owing to the difference in the density of the material at the crack and rest of the sample, the crack will markedly be distinct. The variation in radiometric sensitivity of the individual CCD pixels can easily mask a low contrast flaw indication. The responses can be corrected using a calibration image and subtracting the calibration image in real-time from the incoming measurements. This is done by using a single 16-bit accumulation buffer. Initially, the program integrates the number of frames of measurement into a 16-bit buffer (This is being done by the *GRAB* button of the accumulate window). Then, a calibration field is established, which is used to decrement the accumulation buffer (performed by the *SUBTRACT* button of the window) by the same number of frames and the resultant image can be viewed using hardware divide and offset circuitry. Experiments have proven that this type of calibration with 16-bit precision can dramatically increase the sensitivity limitations of the real-time X-ray inspection systems [15].

As soon as the *GRAB* button is pressed, the program will get the current selection of the camera from the *accumulate* window for image acquisition. Then, it will acquire an image for the specified number of frames from the camera selected. The *GRAB* and *SUBTRACT* buttons, when clicked, will invoke a window displaying the histogram of

the data saved in the 16-bit accumulation buffer. The histogram window also features two sliders which can be used to select a region of grey scale values for image display on the monitor. The principal reasons for the sliders over the histogram are to provide a visual interface of the range of the grey scale values currently being displayed on the monitor and to have the flexibility for the user to select his range of grey scale values.

The images which are acquired after using either of the functions, *GRAB* or *SUBTRACT*, can be saved to a file. The images can be saved in 8-bit or 16-bit formats. Clicking on either of the buttons, *save-8* or *save-16*, will pop open a window wherein a user can type in the file name which is to be saved. An 8-bit image can be saved in PGM format and a 16-bit image in XRV format. A PGM file allocates one byte for each pixel whereas an XRV file saves data related to each pixel in two bytes. In simple terms, the size of an XRV file is twice the size of a PGM file. Appendix B shows the source code for creating and saving an XRV file.

3.2.3 X-ray Generator Control

Increasing inspection demands owing to the more efficient usage of materials are increasing the complexity of information required from an inspection. This implies more complex data acquisition often from a large number of expensive custom acquisition boards. With fewer resources available and multiple users wishing to access them, the need arises to make resources multi-user accessible. Extensive research in NDE came up with a variety of techniques which use different X-ray equipment like generators, cameras, detectors, sample positioner, data acquisition board, etc. Since most X-ray equipment are enclosed in lead shielded vault for safety, they must have an option of being remotely controlled for the enhanced safety of the user and also to change the equipment settings from where one can see a real-time image. The phenomenal growth in the network area client/server design methods makes it possible to make a single resource available to multiple users.

X-ray data acquisition hardware are single user and manually controlled devices. The steep increase in price and usage of these equipment motivates one to make them accessible to multiple users remotely, and the applications of networks enables this. By using remote control capability the exposure to radiation can be reduced. Using the computer control of X-ray equipment one can impose maximum threshold values for the equipment settings, so that whenever a user tries to increase them beyond a limit, the program warns the user and makes the source non-functional for these values. This X-ray generator control part of the program takes care of the remote control of the X-ray generators over the network. The user can choose from a set of generators which are listed in the combo box [2] of the X-ray generator control part (see Figure 3.3). Before the setup window comes up, the program will look for a configuration file containing the details of the generators. A sample generator configuration file is shown in Appendix A. The details include the name of the generator, maximum kV, maximum mA range and the port id, through which data transfer to a remote machine takes place. The program will read the configuration file one line at a time and stores the values of these parameters into a structure and will access them when needed.

4 AUTOMATION OF REAL-TIME SCANS

The success of the X-ray method in scanning materials for crack detection is heavily dependent upon the orientation of the opening and on the width of the opening. If the X-ray beam is not aligned properly in the direction of the crack opening, then there is every chance that the crack can go undiscovered. In order to eliminate that uncertainty, a sample has to be scanned in different orientations with reference to an axis. Film radiography will be cost prohibitive and also time consuming for such sorts of scans. Real-time scans provide an answer for these sorts of scans, where one can repeatedly scan a sample with variation in many of the parameters associated with the scan. If the amount of human intervention in such sort of monotonous scans can be reduced to a bare minimum if not eliminated entirely, automation of the entire scan procedure will be a good idea. This is the idea behind this project. This chapter explains in detail the implementation of this automation of the real-time scans. Figure 4.1 shows the window which handles the automation of the real-time scans.

4.1 Basic Setup

The automated part of the program should be capable of controlling the sample positioner, X-ray generator and the image processing hardware and software. So, the PC23 Indexer cards responsible for motion and the DT2867 board responsible for image processing must be initialized before the program can actually be used, and since the program features the manual part along with the automated part, initializing them once

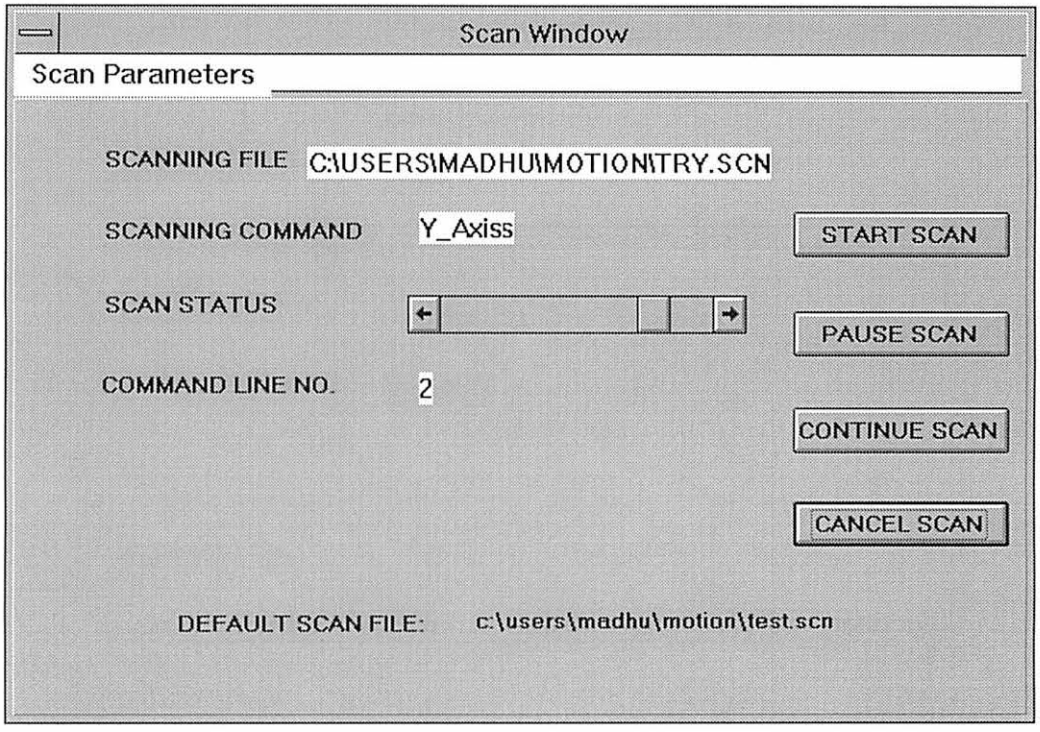

Figure 4.1 Automatic scan window

will be enough. The program upon execution initializes the respective boards and they are accessible to the program till the program is quit. The program creates 3 different objects based upon the constructors of the **motor**, **rmotor** and **dt** classes. The created objects can be used extensively both by the automated and manual sections of the program without duplicating one for each part of the program.

The program upon execution looks for a default file in the current working directory. This file contains various operations which the program can interpret and also serves as a reference of the available functions of the program. This file should be in the same working directory as the executable of the program is. A sample default file is provided in Appendix C. Care must be taken while editing the default scan file. All the automatic

scan files must have an extension of ".scn" to distinguish them among other files. As a safety measure, the X-ray generator has to be turned on manually before any automatic scan can be started.

4.2 Functionality of the Program

The scan window provides a menu (scan parameters) with four different choices of selection: namely, *Load a default file, load a file, edit a file* and *quit.* The first two menu choices will enable the user to load a file, with the first one to load a default file and the second to load a file of the user's choice. The automatic scan window displays the name of the default scan file which the program is using. Since the needs of all the automatic scans will not be alike, there should be a way by which the user should be able to load a file of his choice in addition to the default file. This capability of loading any file with a ".scn" extension has been incorporated into this program. The program also provides the facility of editing an existing file. A file can be edited as many times as one wishes to during the course of the execution of the program. If the edited operations of any file are to be performed, the changed file must be saved before the program can read that file. An option to do this has been provided in the edit window which will be opened when the *edit a file* option of the menu of the scan window has been selected. Finally, as the name says, the last menu option *Quit* will close the scan window (shown in Figure 4.1) and control will be returned to the parent window.

The scan window as can be seen from Figure 4.1, has five fields on its window displayed when opened. They are, *Scanning File*, *Scanning Command*, *Scan Status*, *Command Line No.* and *Default Scan File.* As the names suggest, *Scanning File* and *Default Scan File* commands display the current scanning file and the default scan file, respectively. The *Scan Status* uses a scroll bar [10] to display the current status of the processing of the file. *Command Line No.* displays the line number which is currently

being processed by the program.

The program must be supplied with a valid file name either by selecting the default file or a file of the user's choice. The program will check for the validity of the file upon clicking on the *start scan* button and further execution of the program is contingent upon the validity of the file contents. If everything associated with the file and its contents is appropriate, the program will read the contents of the file one line at a time and will split the line into five equal parts. The first one being the command itself and the remaining four will be the arguments needed for the functioning of the command. Currently, the number of parameters to be passed along with the command has been set to a maximum of 4, but could be varied if necessity arises. Though space has been allocated to pass four arguments along with a command in a single line, it is not necessary to pass all four arguments with each command. If there are any parameters whose values have not been specified in the command line, they will be interpreted as zero. Also, as soon as this button is selected, the program will parse through the scan file to get the total number of lines in the file. This number will be used to set the range of the scroll bar which depicts the status of the processing. A counter which is initialized to "0" is used to display the line number currently being processed. This counter value is incremented in steps of 1 and will be checked to be sure that the counter value doesn't exceed the total of number lines in the file. In order to make sure that the program executes only one line at a time, a subroutine "moto_check()" has been defined. After reading every line from the file this subroutine will be called to check for the motion of all the motors and the next line is read only after all the motors have been halted. This will take care of the motion part. The Data Translation board for image processing is supplied with a status bit to see if the board is busy with any operations requested previously. The same subroutine checks this status bit and will wait until the board is not busy. Also, the subroutine features a flag. The program will read the next line only if this flag is set to 1. So, this flag will be set to 0 as soon as the program control is switched to the

subroutine. This flag will be set to 1 just before the control of the program is shifted back to the main program. Since the subroutine is called once after reading every line, this will make sure that the program executes the command issued in the current line and is ready to read the next line.

The first parameter of the line will be interpreted as the command to do some action and the program will search among all the possible list of commands. If the command doesn't match any of those in the list, no action will be taken in that case and the program will give out a warning message saying that the command typed in a file is not in the acceptable format. But if it does match, the program will perform the necessary action by taking in the necessary arguments supplied along with the command.

The *pause scan* button when clicked will stop processing the file specified until either the *continue scan* button or the *cancel scan* has been selected. This feature of the program is crucial because during the processing of the file if the sample positioner happens to bump into something (which is very much a reality in some cases) or if there is any cause of alarm due to any precarious situation inside the room enclosing the X-ray generators and the sample positioner, then one will be forced to halt whatever operation he/she is doing before any action can be taken. A flag will be set for performing this operation which when set to 0 will halt any further processing of the file and the processing of the file will be done only when this flag is set to a 1. Before halting the processing, the program will finish executing the current instruction and the processing will be halted from the next command onwards. The *pause scan* command stops execution after finishing the current instruction and the *cancel scan* command terminates the execution of the program immediately.

So, after pausing the processing of a file, there should be a way by which one can resume the processing of the file again. This is handled by the *continue scan* button. When clicked, this button will set the flag to 1 and will resume the processing of the file. This functions in almost the same way as the *start button* with the exception of the

position of the file pointer. The start button when clicked on will start processing the file one line at a time always from the beginning of the file, whereas the *continue scan* will 'remember' the position of the file pointer when the *pause scan* button was clicked and will continue processing from that line. Finally, one can cancel a particular scan at any point of time by just clicking on the *cancel scan* button.

A sample default file which is currently under use is provided in Appendix C. This default file is for demonstration purposes only. It provides a reference to the user about the contents of a typical scan file. In order to ensure the safe operation of the program, the distances by which the motors are to be moved are set to zero. Let's look into the default file to see how the file is being read and executed by the program. The program reads one line at a time and splits it into 5 arguments. The first one will be interpreted as the command and the remaining as the arguments to aid the functioning of the command. Upon clicking the *start scan* button, the program will read the first line of the file and since the first line has only two arguments, frames and 20, the first argument will be interpreted as command and the second one as data needed for the command. This line will set the number of frames for any image acquired after that, until the frame value is changed again by passing another command like this. Then the program will read the second line of the file and will interpret X_Axis as the command and the next argument as the amount of distance the sample positioner has to be moved in this direction. Since the objects for remote and local motors have been created already (i.e., during the execution of the program), those objects can be used here for moving the sample positioner. Similarly, the other commands for moving the sample positioner in other axes will be handled. For safety reasons zero is passed as the argument for all the motors in this demonstration file. The sample positioner distances, by which it has to be moved can be specified either in inches or in cm units (for moving the sample positioner in linear direction) and in degrees or mill-radians (for rotational motion) using the *InchtoMM* and *DegtoRad* commands, respectively. If 1 is passed as the argument

to these commands, inches and degrees are selected for the translation and rotational motions. If 0 is passed, cm and milli-radians are selected as the units of motion. Since the default file has the second argument as 1 for both the commands, the units of motion are set to inches and degrees. The acceleration and velocity of the motor to move in the direction of X-axis are specified using the X_accl and X_Vel commands, respectively. The default acceleration and velocity are set to 1 and 0.1. The rest of the motors are handled in a similar manner. In this file, the velocity and acceleration of all the motors have been set to the default values.

One can acquire an image in two types of modes, accumulate and average. The average and accumulate commands are the same as the ones used in the XR Vision program. The average mode will average the specified number of frames for the specified camera and will display the averaged image onto the monitor. The accumulate mode of image acquisition will accumulate the grey scale value of the individual pixels for the specified number of frames up to 16 bits. After acquiring an image through one of these methods, it can be saved in PGM or XRV format (note:Averaged image can be saved only as a PGM file while the Accumulated image can be saved in either of the two formats). Using the *LoadPGMFile* command one can open a specific PGM file and can induce some delay into the program before loading another PGM file onto the monitor, by using *hang* command. The *hang* command makes the program wait for the specifed number of seconds (which will be passed as an argument in the file) before executing any other command.

Now comes the X-ray generator part of the automation program. The current and voltage settings of the individual generators can be set using the A-GenSetting and K-GenSetting commands. The commands have to be supplied with the name of the generator and a number as the first and the second arguments of the command. The program will check to see if the specified generator is available or not. If it is a valid one, the second argument on the line will be read and the voltage or current value of

the specified generator is set to the defined value. Before setting the current/voltage for any generator, the program compares the maximum current/voltage/power ratings with the specified value and checks if it is within the limits. This is to ensure the safe usage of the X-ray generators by not exceeding their maximum ratings.

5 RESULTS, CONCLUSIONS AND FUTURE WORK

As a test of this program, various samples have been scanned, using the automated part of the program and also using the RMover and XR Vision programs. A scan of a typical sample using this program is described in the next section. Also, the time gained by using this program over the RMover and XR Vision programs used together has been measured. The conclusions section of this chapter gives a broad picture of what has been done in this research, and the future work section will describe several ideas as to how this program can be extended for an efficient usage of its features.

5.1 Results

Figure 5.1 shows a sample constructed by riveting together two sheets of aluminium 0.04 inch thick to represent a structure commonly found on the skin of airplanes. The sample has two cracks on either side of the central rivet with each measuring up to a length of 102 and 97 mils (one thousandth part of an inch). The sample positioner was 37 inches away from the detector and was 7 inches away from the X-ray generator. This gave a magnification of about 5. The sample has been scanned for 200 frames using the FXE Generator set up at 45kV and $650\mu A$. In order to reduce the noise signals coming out of the CCD pixel element and also to have a better contrast of the crack with the material of the body, the acquired image is subtracted by moving the sample a little upwards and the resultant image is saved in a file. Then the sample mounted on the sample positioner is moved along the ϕ axis in a range of $+15$ and -15 degrees to obtain

the best alignment of the crack with the X-ray beam. The RMover program on average took a minute to align the sample for each step size of ϕ. In total, the RMover program took 30 minutes to scan this object for an entire 30 degrees of Phi rotation (at a step size of 1 degree in the direction of ϕ. If the crack is much narrower, then one might need to

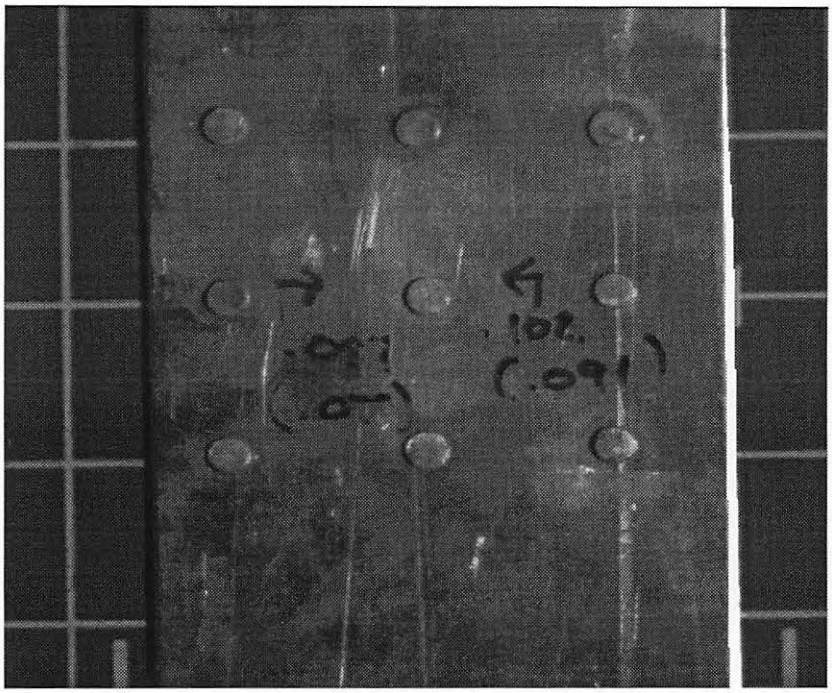

Figure 5.1 Visual image of a sample aluminium panel used in the studies

go at a smaller step size). Apart from moving the sample, it took about an hour for acquiring, subtracting and for processing the image after it had been acquired. So, in total it took 90 minutes to scan a sample. The same sample has been scanned for the same step size using the automated program. The portion of the scan file used for the scan is shown below.

```
Frames          200
```

```
SetCamera       Scan_Camera
Accumulate
Z_Axis         -0.1
Subtract
SavePGMFile      c:/users/madhu/motion/results/samp3p1.pgm
Phi_Axis        1
Accumulate
Z_Axis          0.1
Subtract
SavePGMFile     c:/users/madhu/motion/results/samp3p2.pgm
|
|
|
|
|
SavePGMFile     c:/users/madhu/motion/results/samp3p16.pgm
Phi_Axis        -15
Accumulate
Z_Axis          -0.1
Subtract
SavePGMFile     c:/users/madhu/motion/results/samp3n1.pgm
Phi_Axis       -1
Accumulate
Z_Axis          0.1
Subtract
SavePGMFile     c:/users/madhu/motion/results/samp3n2.pgm
Phi_Axis       -1
```

```
|
|
|
|
|
|
SavePGMFile     c:/users/madhu/motion/results/samp3n14.pgm
Phi_Axis        -1
Accumulate
Z_Axis          -0.1
Subtract
SavePGMFile     c:/users/madhu/motion/results/samp3n15.pgm
```

The frame count has been set for 200. The sample is initially positioned at a zero reference position with respect to ϕ-axis. The sample then is accumulated for 200 frames and then it is moved by one-tenth of an inch in the direction of z-axis (positive or negative depending upon the present position of the sample positioner) where a new image is subtracted from the previously accumulated image. The subtracted image is then saved into a PGM file for future reference. The sample positioner has been rotated for a total of 30 degrees with fifteen degrees on either side of the zero-reference position. Using the automated program, the same scan took fifteen minutes. Adding the overhead as another five minutes for typing the commands in a file, the automated program takes a total of 20 minutes. This shows an overall improvement of 450% using this program over the two programs, XR Vision and RMover. Figure 5.2 shows the accumulated image of the sample over 200 frames. As can be seen from the figure, the cracks on either side of the central rivet are not visible in this method. This can be attributed to the systematic and random variations of the CCD pixel elements (refer, section 3.2.2.2).

In order to have a better contrast of the crack with the rest of the sample material, some image processing techniques have to be applied. Hence, the image of the same rivet after it is accumulated over 200 frames has been moved laterally upwards by one-tenth of an inch and then subtracted from the incoming measurements [15]. The resultant image is shown in figure 5.3. The crack is now visible on either sides of the rivet.

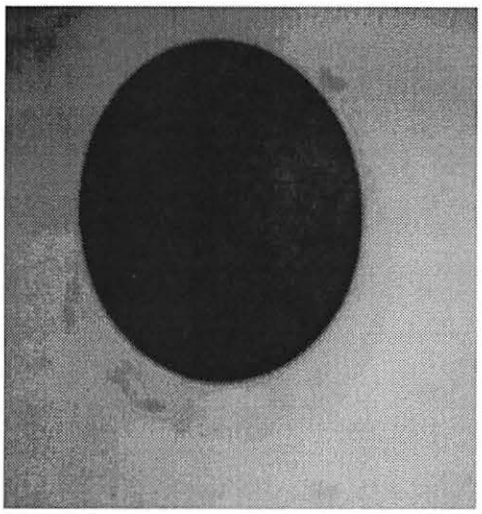

Figure 5.2 An accumulated image before subtraction

5.2 Conclusions

The research work described in this thesis was motivated by some of the requirements in the field of X-ray based nondestructive testing and the limitations of existing systems in meeting these requirements. The principal advantage of this research is for those scans where one has to repeatedly do the same operations on a sample. As a part of this research, the two software packages XR Vision and RMover are used extensively to develop a module which can handle image processing and motion control, both manually and

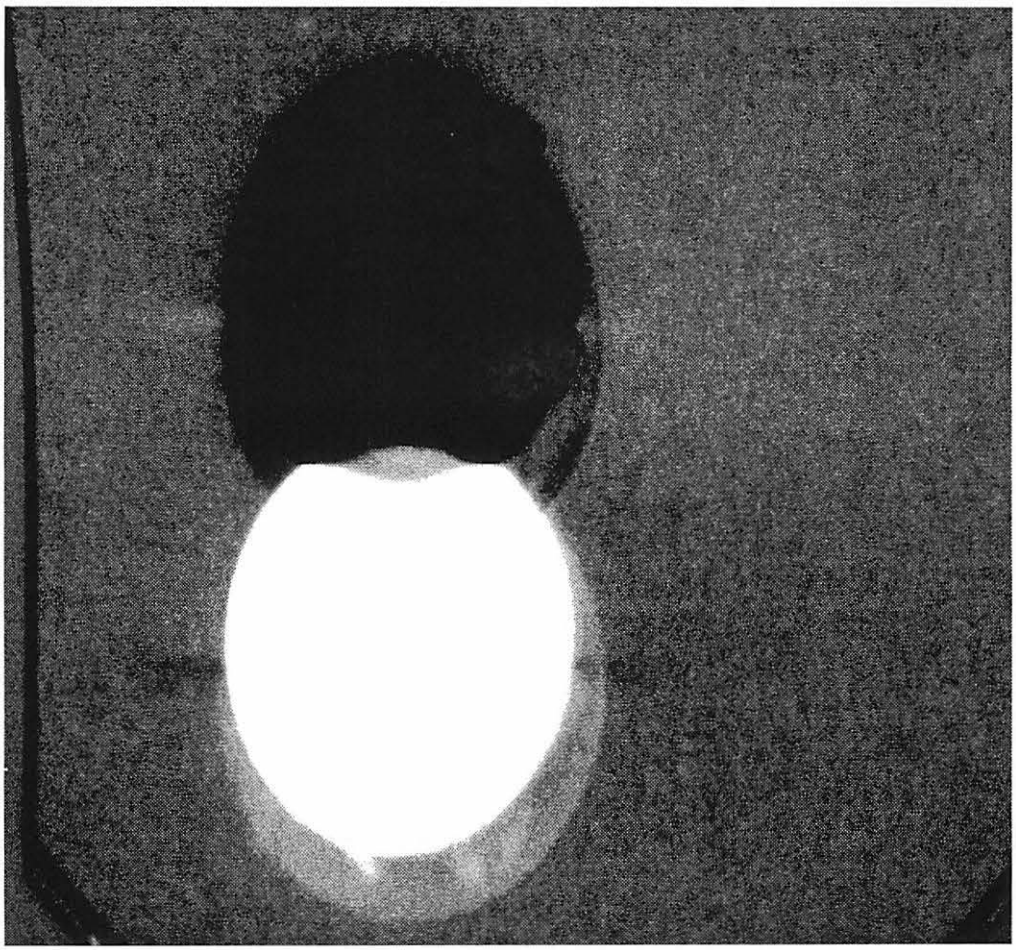

Figure 5.3 Accumulated image after subtraction

automatically. The new XRSCAN program enables the user to select between manual and automatic scanning of parts at will. This project has demonstrated the capability of using the available member functions in the three classes; namely, **rmotor**, **motor** and **dt**. Also, other classes can be added if needed. Histogram display of a 16-bit image has been developed and the sliders on the histogram enable one to select the grey scale range to display over the monitor. The motion control part of the program provides the facility of setting the mode of motion (continuous and fixed), sending the motors home and also the selection of reference position. This program also features the continuous display of the sample positioner distance from a reference position. The program also provides the user with an option to select between the English and Metric system of units for specifying the velocity, acceleration and distances. The X-ray generator section of the manual part can be used to set the Generator kV and mA settings.

As the primary focus of the research work, an automated version of all the operations available in the manual setup mode have been developed. Using the current automated program one can acquire multiple scans at various positions of the sample positioner in very short time and can save the acquired images in either 8-bit or 16-bit format for future reference. Most commands which are available in the manual mode are also available in the automatic mode.

5.3 Future Developments

This research demonstrated the capability of automating the real-time scans which need little or no human intervention. Though this is a small step towards automation, it is a much deserved one. The present program can handle smaller objects which can be moved into the laboratory and can be placed on the sample positioner. But, considering a bigger sample like an airplane, there is a need to move the X-ray generator with respect to the sample, contrary to the existing setup where the sample will be moved against a

stationary source. Hence, the motion control part of the program has to be extended to X-ray generators and to real-time detectors, as well. The program should be enhanced to handle even other type of detectors. Also, the future plans are to incorporate additional image processing processes such as FFT.

This program doesn't have any knowledge about the appropriate scan step size and also about the range over which the sample positioner has to be moved in a direction. This can be obtained by using the XRSIM program [3], [17]. XRSIM is a UNIX based X-windows application for simulation of X-ray scans. XRSIM program features several CAD models of various objects which can be simulated for specific generator settings of kV and mA. Also, one can vary the angle of orientation and distance of the sample relative to the source and detector using the XRSIM program. These features of XRSIM program can be used to obtain simulated radiographs enabling the user to get a good idea about the step size for the scans in real-time. Hence, the automation program can be linked with the XRSIM program to predict the optimized scans.

APPENDIX A CONFIGURATION FILES REQUIRED FOR PROGRAM OPERATION

Motor Configuration file

```
/*
This is the configuration file for the real time motion control
program. This file must be present at the proper location and any
comments to be included in future must be included here only as the
code has been written to handle a single comment section at the top.
Don't write anything after the "*/" sign, without knowing what you are
doing. Presently it has 5 motors.
*************    */

5
LOCAL  ROTATION  512  1
LOCAL  ROTATION  512  2
REMOTE  TRANSLATION  890  1
REMOTE  TRANSLATION  891  2
REMOTE  TRANSLATION  892  3
```

Camera Configuration file

```
/* This is the camera configuration file describing the available
cameras and their port numbers */


2
Scan_Camera   0
Surveillance_Camera  1
```

X-ray Generator Configuration file

```
/* Generator Configuration file with all the details of the generators.
The format of the file looks like:

Gen. Name MaxkV  MaxmA  Port */

2
IRT   320      10  100
FeinFocus  200      0.1  101
```

APPENDIX B SOURCE CODE FOR SAVING AN XRV FILE

```
/*   Function for writing a 16-bit image file in Binary Format. This is
the source code in C for saving an XRV file into a file */

BOOL BigFileWrite(HWND hwnd, unsigned short huge * lpImage, LONG nRows,
LONG nColumns, LPSTR lpszFilename)

        {

        HANDLE          hFile;
        long            i, nPixels, nBlocks, nRemain, lBytesWritten;
        char            szP5[3], szRC[9], szMaxGray[4], szSpace[1];
        char            szRows[6],szColumns[6],szBlocks[6],szRemain[6];
        char            szReturn[1];

        extern unsigned short huge *  lpBigImage;


        // Create output file
        hFile = _lcreat(lpszFilename, 0);

        // Compute number of 64k blocks and remainder
        nPixels = nRows * nColumns *2;
        nBlocks = (long) nPixels/0xFFFE;
        nRemain = nPixels - (nBlocks*0xFFFE);

        // Write the 64k blocks
        for (i=0; i<nBlocks; i++)
                {
                _lwrite(hFile, lpBigImage+(i*0x7FFF), 0xFFFE);
}

        // Write the remaining bytes
        if (nRemain != 0)
```

```
        {
        _lwrite(hFile, lpImage+(0x7FFF*nBlocks), nRemain);
        }

// Close the file
_lclose(hFile);

return TRUE;
}
```

APPENDIX C DEFAULT FILE FOR AUTOMATED SCANS

```
Frames    20
X_Axis 0
Y_Axis 0
Theta_Axis 0
Z_Axis 0
X_Vel 0.1
Y_Vel 0.1
Z_Vel 0.1
Theta_Vel  0.1
Phi_Vel 0.1
X_accl 1
Y_accl 1
Z_accl 1
Theta_accl 1
Phi_accl 1
RadtoDeg   1
InchtoMM   1
SetCamera   ScanCam
Average
Accumulate
SavePGMFile C:\users\madhu\motion\trial.pgm
SaveXRVFile C:\users\madhu\motion\trial.xrv
LoadPGMFile C:\users\madhu\motion\1.pgm
hang 10
LoadPGMFile C:\users\madhu\motion\1.pgm
hang 15
K-Gensetting IRT   0
A-GenSetting FXE 0
```

APPENDIX D RMOTION.H : HEADER FILE LISTING THE FUNCTIONS AVAILABLE FOR MOTION CONTROL

```
/*************************************************************************
RMOTION.H
By Vivekanand Kini
Update : 08/22/93
 *************************************************************************/
#include <stdio.h>
#include <time.h>
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include <stdarg.h>
#include "network.h"

#ifdef WINDOWS
 #include <windows.h>
 #include <windowsx.h>
#endif



#ifndef DBooLean
enum BooLean {MY_FALSE = 0, MY_TRUE = 1};
#define DBooLean
#endif


// The following code defines a structure called 'Motor' which has data
// variables as well as certain "member functions". These functions have access
// to all the data variables within the structure. When we define variables of
```

```
// type 'Motor', that variable gets it's own set of member functions.
// To call a particular function, the syntax is: variable.member function
// We can also define one of the memeber functions as a 'constructor' which
// is called as soon as the variable is declared. The constructor can be
// used to initialize the structure's data variables. The name of consrtuctor
// should be the same as the structure variable name.

class  RMotor {

 // Data variables. None of these variables can be directly accessed by user
 protected:
  char name[80]; // name of motor

  BooLean pc23_initialize_flag; // enables resetting of PC23 board when ON
  BooLean message_flag;     // enables debugging messages when ON

  int device_id;
  int (*sendfn)(char far *);
  int (*recvfn)(char far *);
  packet spkt, rpkt;

  // Member functions
  // The private member functions cannot be used by the user
  protected:
    // These are a generic set of output routines which can be used instead
    // of printf() for easy plugging of any graphical user interface. These
    // routines would have to modified/rewrittern for any user interface.
    // Currently standard DOS & MS-WINDOWS 3.1 is supported.
    void out_message(char *format, ...);
    void out_warning(char *format, ...);
    void out_error(char *format, ...);

    void send_command(int command, char *format = "", ...);
    void recv_response(int command);

    BooLean pc23_initialize();

    #ifdef WINDOWS
    HWND handle;
    char szAppName[20];
    #endif

  // These routines form the high level function to be used by programmer
  // for more information about these functions, refer compumotor's pc23
```

```
  // indexer guide. the specific command to look up corrosponds to that of
  // comment provided at the end of each function declarations below
  // the general structure of most of the following functions is very similar
  // except for the command string - look at motion.cpp
  // NOTE:
  // ANY FURTHER HIGH LEVEL ROUTINES CAN BE DECLARED HERE AND ADDED TO
  // motion.cpp

  public:

    RMotor(char *c_name, int id);  // constructor
          /* 1: pointer to a string containing the name of the motor
     2. device_id on network.h
     4: if MY_TRUE - the board is initialized
           MY_FALSE - the board is not initialized. This is default
NOTE: when using multiple motors, initialize the board only
      once because each time the motor is initialized, all the
      earlier information is lost.
      This routine initializes the motor variables as follows:
         Mode = normal
 Positioning = absolute
         Velocity = 1.0
                 Acceleration = 1.0

  */

    void register_sendfn(int (*fn)(char far *)) { sendfn = fn; };

    void register_recvfn(int (*fn)(char far *)) { recvfn = fn; };

    virtual void initialize(BooLean pc23_initialize_flag = MY_TRUE);

    virtual void setmode_continuous();       // MC
          /* the motor is set to move in continuous mode
     this mode is not used frequently
  */

    virtual void setmode_normal();       // MN
          /* the motor is set to operate in normal mode
     this is most commonly used for our applications
          */

 //   virtual void setmode_alternate();       // MA
          /* the motor is set to operate in alternate mode
```

```
   this mode is not used frequently
*/

//   virtual void setmotor_resolution(int resolution);  // MR
        /* sets the motor resoultion i.e. number of steps/inch or degree
   not used frequently
   by default set to 100000 steps/inch and 10000steps/degree
*/

  virtual void setposition_absolute();    // MPA
        /* sets the motor in absolute positioning mode
   used very commonly
*/

  virtual void setposition_incremental();  // MPI
        /* sets the motor in incremental or relative positiong mode
*/

  virtual void setdirection_cw();      // H+
        /* sets the motor motion clockwise
*/

  virtual void setdirection_ccw();      // H-
        /* sets the motor motion anticlockwise
*/

  virtual void change_direction();       // H
        /* changes the motor motion direction
*/

  virtual void setposition_zero();       // PZ
        /* sets the position counter to zero
*/

  virtual void setvelocity(float velocity);      // V
        /* 1: velocity of the motor
   sets the motor velocity
*/

  virtual void setacceleration(float acceleration);    // A
        /* 1: acceleration of the motor
   sets the motor acceleration
*/
```

```
  virtual void setlimit_acceleration(float acceleration);     //LA
        /* 1: decceleration when limit switch is encountered
*/

  virtual void setgohome_acceleration(float acceleration);   // GA
        /* 1: acceleration when homing
*/

  virtual void setbackupto_home(BooLean backupto_home);     //OSB
        /* 1: when MY_TRUE - backup to home when home is encountered-default
           MY_FALSE - do not backup to home
*/

  virtual void setactivehome_state(BooLean active_state);   //OSC
        /* 1: when MY_TRUE - active home state is high
           MY_FALSE - active home state is low - default
*/

  virtual void setfinalgohome_direction(BooLean finalgohome_direction); //OSG
        /* 1: when MY_TRUE - final homing direction is counterclockwise
                             - default
           MY_FALSE - final homing direction is clockwise
*/

  virtual void sethomedge_reference(BooLean homeedge_reference); //OSH
        /* 1: when MY_TRUE - final home edge is clockwise side of signal
                             - default
           MY_FALSE - final home edge is conterclockwise side
*/

  virtual void move(long int absolute_steps);      // D, G
        /* 1: long integer specifying the number of steps to be moved
   moves the motor by the specified number of steps
*/

  virtual void move_inch(float absolute_inches); // D, G
        /* 1: number of inches to move the motor
   moves the motor by the specified number of inches
*/

  virtual void move_degree(float absolute_degrees); // D, G
        /* 1: number of degrees to move the motor
   moves the motor by the specified number of degrees
*/
```

```
  virtual void move(void);       // G
        /* moves the motor - used in continuous mode
*/

  virtual void limit_enable(); //LD0
        /* enables the limit swicthes
*/

  virtual void limit_disable(); //LD3
        /* disables the limit switches
*/

  virtual BooLean report_limit(); // RA
        /* returns MY_TRUE if limit has been encountered
           MY_FALSE if limit has not been encountered
*/

  virtual void pause_motion();       // P
        /* pauses the current motor motion
*/

  virtual void continue_motion();        // C
        /* continous the current motor motion - used after pause
*/

  virtual void stop_motion();        // S
        /* stops the motion - the motor deccelerates to a halt
        */

  virtual void kill_motion();       // K
        /* kills the motion - motor sudddenly stops
   use not recommended except during emergency
*/

  virtual void shutdown();     //ST
        /* removes the holding torque from the motors
   highly recommended when leaving motors energized for a long
   period of time, reduces heating.
   disabled as soon as any other command is issued
*/

  virtual void gohome(float velocity);  // GH
        /* 1: velocity of motion
```

```
    sets the motor velocity
  */

    virtual BooLean checkmotion();   // Check motion status
          /* returns MY_TRUE if the motor is moving
             MY_FALSE if the motor is not moving
  */

    virtual long getposition_ab();   // returns the ab pos in steps
          /* returns a long integer with current absolute position of the motor
  */

    virtual long getposition_rel();   // returns the rel pos in steps
          /* returns a long interger with current relative position with
     respect to that last move
  */

    virtual int getaddress();    // returns the board address
          /* returns the adress of the pc23 board
  */

    virtual int getnumber();     // returns motor number
          /* returns the motor number
  */

    virtual void setmessage_flag(BooLean c_message_flag);
          /* 1: if MY_TRUE - enables the debugging messages
           MY_FALSE - disables the debugging messages - default
  */

#ifdef WINDOWS
 void setwindow_handle(HWND c_handle, char * c_szAppName);
#endif
};
```

BIBLIOGRAPHY

[1] Compumotor. *PC23 Indexer User Guide.* Rohnert Park, CA, May, 1990.

[2] James L. Conger. *Windows API Bible: The definitive programmer's reference.* Waite Group Press, Corte Madera, CA, 1992.

[3] J. Xu et al. Recent developments in the x-ray radiography simulation code: XRSIM. *Review of Progress in Quantitative Nondestructive Evaluation*, 13:557–563, 1994.

[4] American Society for Nondestructive testing. *Nondestructive testing handbook.* Marlboro, MA, second edition, 1985.

[5] Data Translation Inc. *Data Translation: DT2867 and DT2867-LC Software User Manual.* Marlboro, MA, second edition, October, 1992.

[6] Emmett Frank Kaelble. *Handbook of X-rays, for diffraction, emission, absorption, and microscopy.* McGrawhill Book Company, NewYork, 1993.

[7] Vivekananda Kini. Tomographic inspection systems using x-rays. Master's thesis, Iowa State University, Ames, IA 50011, 1994.

[8] Stanley B. Lippman. *C++ Primer.* Addison-Wesley Publishing Company, Reading, MA, second edition, 1991.

[9] Microsoft. *WINSOCK Reference Manual.* http://www.microsoft.com, 9th July, 1996.

[10] Charles Petzold. *Programming Windows 3.1.* Microsoft Press, Redmond, WA, third edition, 1992.

[11] Sudha Puvvadi. Remote control of X-ray hardware. Master's thesis, Iowa State University, Ames, IA 50011, 1995.

[12] Nutakki Gangadhar Rao. Stereography: A low cost alternative to computerized tomography. Master's thesis, Iowa State University, Ames, IA 50011, 1995.

[13] R.M.Wallingford and J.N.Gray. Use of an X-ray process model to determine crack detectability in a multi-layer geometry. *Review of Progress in Quantitative Nondestructive Evaluation*, 12:319–326, 1993.

[14] R.M.Wallingford and J.N.Gray. Application of Real-Time Image Processing and Calibration techniques to Real-Time X-ray NDE. *Review of Progress in Quantitative Nondestructive Evaluation*, 13:755–762, 1994.

[15] R.M.Wallingford and J.N.Gray. Real-time X-ray image processing; techniques for sensitivity improvement using low-cost equipment. *Review of Progress in Quantitative Nondestructive Evaluation*, 14:871–878, 1995.

[16] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley Publishing Company, Reading, MA, second edition, 1994.

[17] T.Jensen and J.N.Gray. RTSIM: A computer model of real-time radiography. *Review of Progress in Quantitative NDE*, 14:353–359, 1995.