Remote Operation of an Electrical Resistance Tomography System via an IP Connection

Jason Salkinder

A dissertation submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the requirements
for the degree Bachelor of Science in Engineering.

Cape Town, October 2005

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the fulfillment of the degree of Bachelor of Science in Engineering at the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author . . . .

Cape Town

18 October 2005

Abstract

This report describes the implementation of a MATLAB user interface for the remote operation of an electrical resistance tomography instrument via an IP connection. MATLAB code together with open source EIDORS reconstruction functions and existing C++ functions have been combined to form an easily modifiable, graphical user interface for data capture and image reconstruction. A description of the present state of the UCT tomography system is presented along mathematical theory used for image reconstruction. An overview of TCP/IP communication between MATLAB and C++ is presented before the design functionality, coding and implementation of the remote client is discussed .The application of the remote MATLAB client in feedback control for a control volume within the measurement vessel is investigated. Performance testing of both remote imaging and control feedback are discussed. Conclusions are drawn based on these results and improvements recommended.

To my family for all

their support

Acknowledgements

I would like to thank my supervisor, Dr A. Wilkinson for his advice and guidance throughout this thesis.

I extend my gratitude to the following people for their assistance:

- Bill Randall for his untiring help, advice and input into this thesis.
- Tim Long for his advice and coding of the C++ server.

Contents

List of Figures

List of Tables

Glossary and Acronyms

Opcode—The code instructing the server to perform the required operation specified by the client

ERT—Electrical Resistance Tomography

TCP/IP— Transmission Control Protocol/Internet Protocol

EIDORS—Electrical Impedance Tomography and Diffuse Optical Tomography Reconstruction Software

EPROM—Electrically Programmable Read only Memory

MCU—Microprocessor Unit

ADC—Analogue to Digital Converter

DAC—Digital to Analogue Converter

MSB—Most Significant Bit

DC—Direct Current

Drive Pair—The pair of electrodes used for current injection

Sense Pair—The pair of electrodes used for sensing

Sense Layer—The layer in which the voltage sensing is implemented

Chapter 1

Introduction

1.1 Subject of this Thesis

This thesis describes the development of a remote operation MATLAB front end to an Electrical Resistance Tomography (ERT) System via an IP connection.

1.2 Background of this Thesis

The ERT project was initiated in 2000 as a result of an approach by UMIST Chemical Engineering to produce a low cost system for minerals processing applications. Commercially available systems were based on AC excitation with synchronous detection or anagous digital signal processing (DSP) techniques. These designs developed by UMIST Electrical Engineering in the early 1990's and reported in several papers, e.g. Dickin and Wang [1], typically achieved capture rates of 50 frames/sec. Instruments based on this work became commercially available in the late 1990's. Operating speeds of up to 100-200 frames/sec were acheived. Updated versions of these systems are currently available at costs exceeding GBP 35000.

A design was proposed by UCT that involved the injection of a DC current pulse. This simplified the measurement electronics and had potential for high speed operation and would lower manufacturing costs. An initial implementation of this technique was reported by Cilliers et al 2001 [2] which verified the measurement concept.

The present UCT current pulse ERT system (as described in [3]) uses a graphical C front end developed by Tim Long (MSc student, UCT EE) and reconstruction algorithms implemented in MATLAB by Wilkinson for data capture and image reconstruction. C code is not easily modified by users for specific experimental applications. A MATLAB 'front end' was required to provide an easily programmable, interactive interface for acquiring data and implementing real-time Remote control.

1.3 Basic Tomography Principles

Tomography can be implemented using X-rays, gamma rays, lasers, electrical excitation etc [4]. A general block diagram of a tomography system is given in figure *1.1*. If a detection method was implemented so that any distrubances between the medium and excitation source are detected, tomography image reconstruction is possible. In the case of medical X-rays the X-ray beam passes through the medium in a straight line which is distrurbed/attenuated by human tissue and detected by the photographic film. The higher density regions show up as a 'shadow'. If the excitation source and detector are rotated around a specified axis and an appropriate reconstruction algorithm is used, a 3-dimensional image can be formed.

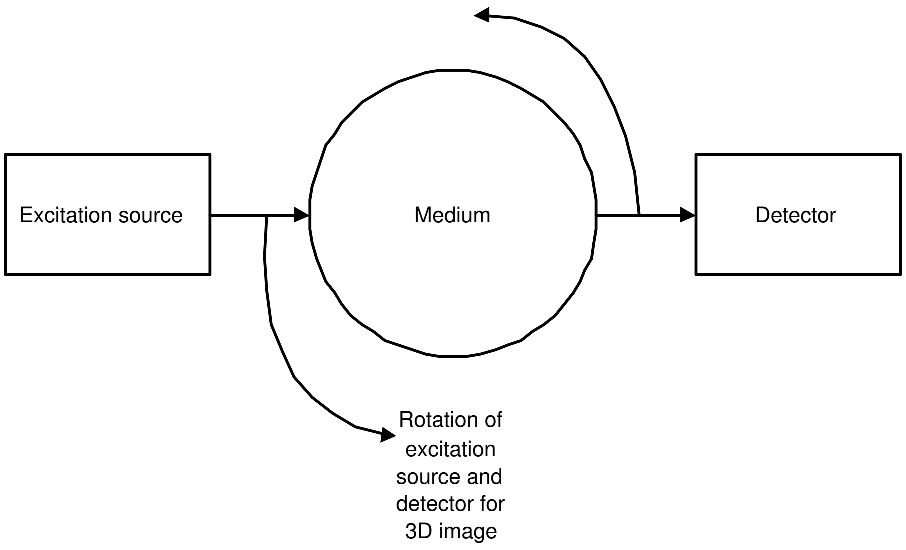


Figure 1.1: Basic Tomography Principle

There are three types of electrical tomography namely:

- Electrical Impedence Tomography (EIT)
- Electrical Capacitance Tomography (ECT)
- Electrical Resistance Tomography (ERT)

ERT is the simplest to implement and is the technique of choice were the experimental medium is primarily resistive and effective inductance/capacitance can be ignored. This is indeed the case in minerals processing where the medium of investigation is often a slightly saline aqueous solution.

An ERT measurement system typically consists of measurement vessel with a ring of 16 perphiral electrodes as shown in figure 1.2. Injecting a known current between two electrodes (*drive pair*) and sensing the voltage between two other electrodes (*sense pair*), the conductivity between the two electrodes can be calculated. Electrical changes within the region effect the measured voltages at the boundary. The measured data set is compared with pheriperal voltages predicted by a forward finite element model (FEM). An iterative procedure is used to match the conductivity across the measurement plane and predicted data set. The result of this operation is generally presented as the reconstructed image.

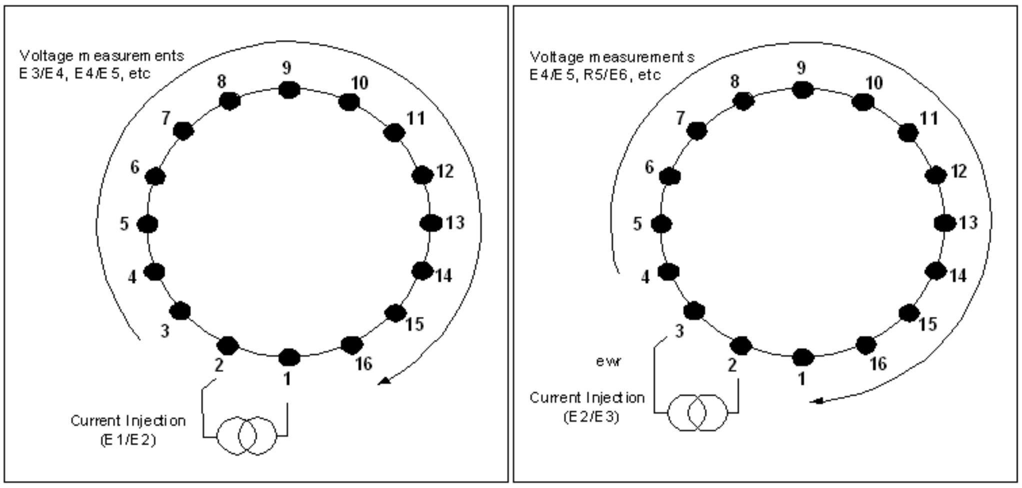


Figure 1.2: Current injection and subsequent voltage measurements *Courtesy of EW Randall*

1.4 Objectives of This Thesis

The objectives of this thesis are to:

- Understand the principles and concepts involved data acquisition and imaging of ERT.
- Give an overview of the functions required for MATLAB to C++ functions via IP and test data rates over the IP network.
- Discuss the required functionality required for ERT remote operation and design an IP communications protocol to implement the functionality between client and server.
- Code the functionality of the client using the designed protocol into a user friendly MATLAB GUI for data capture and image reconstruction.
- Aquire data sets for processing using open source EIDORS code.
- Test and analyze the performance of remote operation.
- Use the remote system in a feedback control loop.
- Draw conclusions based on these findings in this thesis.
- Make appropriate recommendations for system improvements.

1.5 Scope and Limitations of this Thesis

The primary objective of this thesis is to implement the remote operation of the ERT hardware via an IP connection and demonstration that this configurartion can be used for remote data capture and real time feedback control. In order to carry out this project a preliminary study of the present ERT hardware and associated data acquisition software was required. In addition it was necessary to to become familiar with the open source EIDORS reconstruction functions (MATLAB) which were used for data processing.

The computers used in this thesis were connected via a 100Mb/s ethernet link via switch and had the following specifications:

SPECIFICATION	SERVER	CLIENT
PROCESSOR	32-bit Intel Processor	64-bit Intel Processor
SPEED	2.8GHz	2.8GHz
RAM	512MB	4GB
OPERATING SYSTEM	Windows XP, Service Pack 2	Windows XP, Service Pack 2

Table 1.1: Computer Specfications

MATLAB v7.04.365 (R14) Service Pack 2 was installed on the client PC.

1.6 Development Plan

This thesis begins with a description of the present hardware and software of ERT system at UCT.

Chapter 3 will examine the mathematics behind image reconstruction and the use of EIT and Diffuse Optical Tomography Reconstruction Software (EIDORS) package.

In Chapter 4 an overview of IP interactions between two computers on a network will be presented in the form of a demonstration of transferring data over a TCP/IP communication link between a MATLAB client and C++ server. Data transfer rates between the two computers will be tested and the results discussed.

Chapter 5 will look at the designed functionality of the MATLAB client in terms of operations and protocols required for remote operation. The remote client code sequencing for continuous reconstruction of updated data sets will then described.

Chapter 6 will show the implementation of the remote client GUI with a discussion of the reconstructed images.

Chapter 7 will investigate the application of the MATLAB remote client for feedback control.

Chapter 8 and 9 are where the conclusions will be drawn and design improvements recommended respectively.

Chapter 2

Description of the ERT System

This chapter briefly describes the present state of the UCT ERT system as used in this thesis project.

A note must be made on the nomenclature:

- A *layer* refers to all points that lie in the same plane as a single electrode ring.
- A *frame* refers to a complete set of data samples for one iteration of the measurement sequence[1] as specified in the measurement sequence table which is downloaded to the embedded proccessor. This data set can be used to reconstruct the image.

2.1 System Configuration

The present phyiscal UCT ERT instrument is shown in figure 2.1

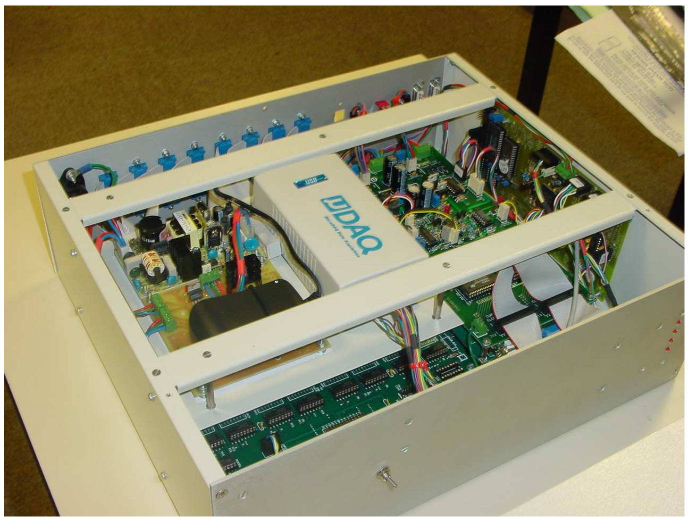


Figure 2.1: The present UCT ERT instrument

[1]See Section 2.1.2

The configuration of the hardware is an extention of that described in [5, 6] and includes an electrode multiplexing scheme as described [7] and is capable of measurements using 8 planes of 16 electrodes at a rate of 100 frames/sec on all 8 layers as shown in figure 2.2. It also allows cross plane current injection and measurement. The multiplexors are controlled by a HC12 microprocessor and and selects the drive and sense pairs as specified in a table download from the controlling PC. The layer multiplexors used having high impedence allowing modules to be connected in parallel without loading the circuit. Electrodes within each ring are numbered *E1* to *E16* and layers *L1* to *L8* with layer one being the bottom most layer. The design implementation has 16 separate differential amplifier and sample and hold channels as opposed to one multiplexed differential amplifer with a single sample and hold channel. This enables high data capture rates as there is no delay in the avaibility of measurements for analog to digital conversion.

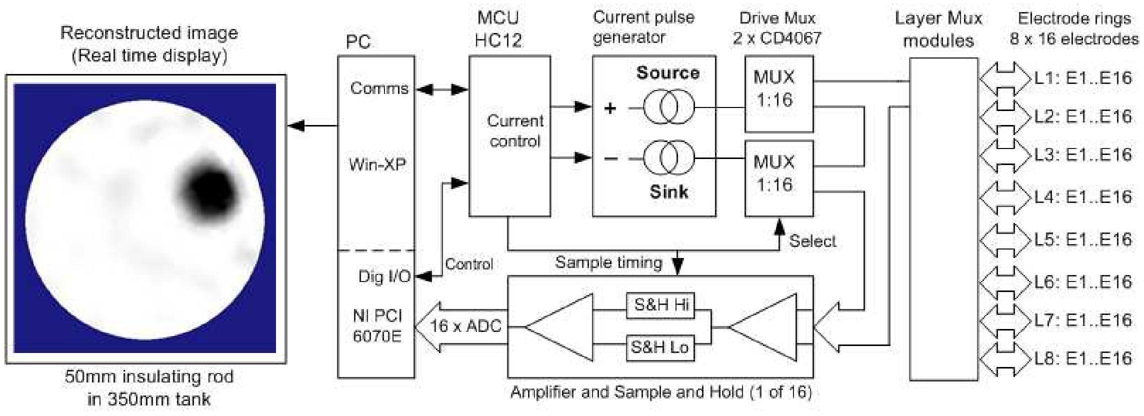


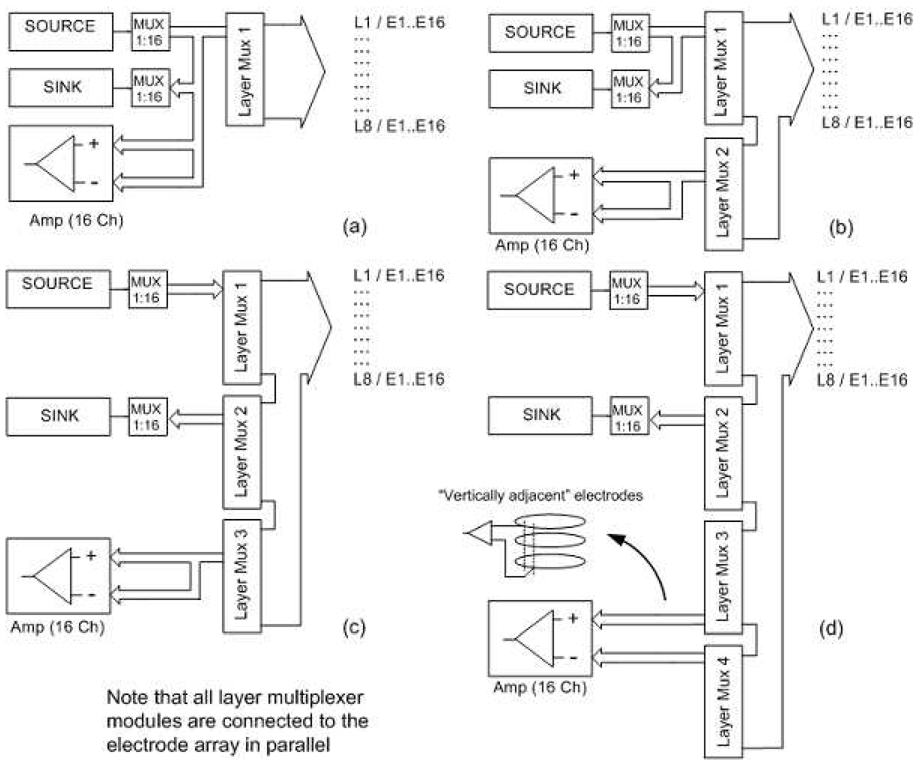
Figure 2.2: Hardware block diagram of the UCT ERT system

The addition of the “layer multiplexors” allows current injection and sensing across electrode planes. Up to 4 of these modules may be connected in parallel to the electrode system in order to achieve the flexilibility of current injection and meausrement sensing as shown in figure 2.3 .

The unit used for this thesis had three multiplexor modules installed *(figure 2.3c)*, enabling current injection between planes and adjacent measurements on any specified plane. All data set acquired in this study used the adjacent pair strategy (as shown in appendix A) on the four electrode ring independently and is referred to the “four layer independent measuring sequence”. The physical multiplexor modules are shown in figure 2.4 .

2.1.1 Fundementals of Current Pulse Technique

The single plane, 4 electrode system of figure 2.5 a illustrates the operating principle of the current pulse technique. A bi-directional constant current is injected into the driving eletrodes and the resulting potential difference (Vpp) is measured on the sense pair of *(figure 2.5b)*. The current waveform is a bi-directional pulse with constant amplitude throughout the positive and negative half cycles . The potential difference across the



SOURCE
SINK
MUX 1:16
Layer Mux 1
Layer Mux 2
Layer Mux 3
Layer Mux 4
L1 / E1..E16
L8 / E1..E16
Amp (16 Ch)
(a)
(b)
(c)
(d)
"Vertically adjacent" electrodes
Note that all layer multiplexer modules are connected to the electrode array in parallel

Figure 2.3: Effect on flexibilty current injection and voltage sensing due to the addtion of layer multiplexor modules [7]

Figure 2.4: The physical multiplexor modules [7]

drive pair exhibits a first order transient response due to the capacitance developed at the electrode/solution interface *(figure 2.5c)*. The design of a bi-directional current source eliminates polarisation of the solution which effects the charaterstic impedence of the solution. In the case of the measurement electrode pair, there is zero current flow between electrode and solution and therefore no capacitive effects. A squarewave is therefore observed at the output of the differential amplifier *(figure 2.5d)*. The circuitary allows the amplitude of the squarewave between each measurement pair to be recorded for every current injection cycle [6].

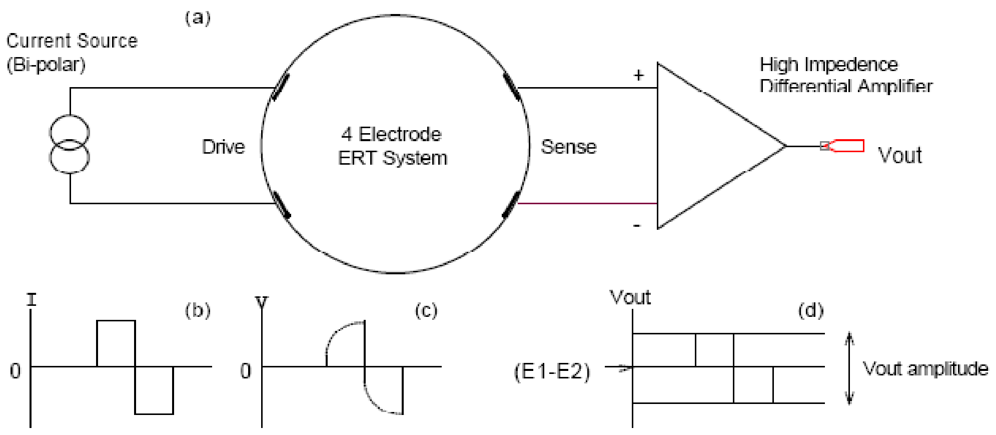


Figure 2.5: The bi-directional current system [6]

2.1.2 HC12 Embedded Microprocessor

The HC12 microprocessor has two main functions:

- PC Interfacing
- Analogue to Digital Timing Control

The embedded HC12 and the PC communicate via a serial connection. The reprogramming the HC12 after every measurement sequence has been avoided by the inclusion of an EPROM resident in the MCU. The EPROM is downloaded with a binary excecutable which is given control once the download is complete[2]. The executable includes information about the timing sequences, current injection adjustment charateristics and voltage sampling functions. In addition to this executable, a binary measurement sequence table is downloaded to the EPROM. 'The table consists of 3 byte sequences which specify the current source and sink layers, current source and sink electrodes and layers for non-inverting and inverting amplifier inputs. The executable then decodes this information and sequentially selects the electrodes (via the multiplexers) for each step required in the current injection and measurement sequence [2].' The flowchart of the measurement sequence logic is shown in figure 2.6.

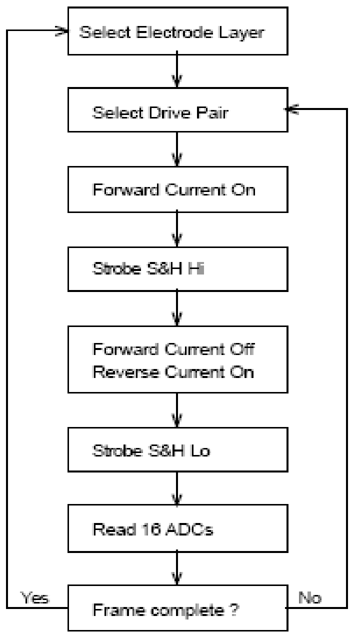


Figure 2.6: Measurement sequence logic [7]

The HC12 controls the timing of the current injection pulse and the sample and hold strobe lines. On completion of the data frame, the the HC12 triggers the *strobe ADC* line and samples the 16 differential amplifiers *(figure 2.7)* . Analogue to digital conversion of the voltage measurements are aquired by a National Instruments Data Acquisition Card (PCI-6070E) installed in the PC. The card has an internal 16 bit, 16 channel ADC. The executable is repeated until terminated by the PC. [6]

2.2 System Extension

This thesis extends the block diagram shown in figure 2.2 by moving the reconstruction of an image to a remote client. The new system of figure 2.8 shows that the PC server interacts exclusively with the ERT hardware. Opcodes are sent via a TCP/IP link from client to server, where they are decoded and the necessary operations completed. The server program was written in C++ by Tim Long using data acquistion techniques and functions existing in *UCT_Tomography* but without the online reconstruction and GUI where as the remote client was programmed using a MATLAB GUI.

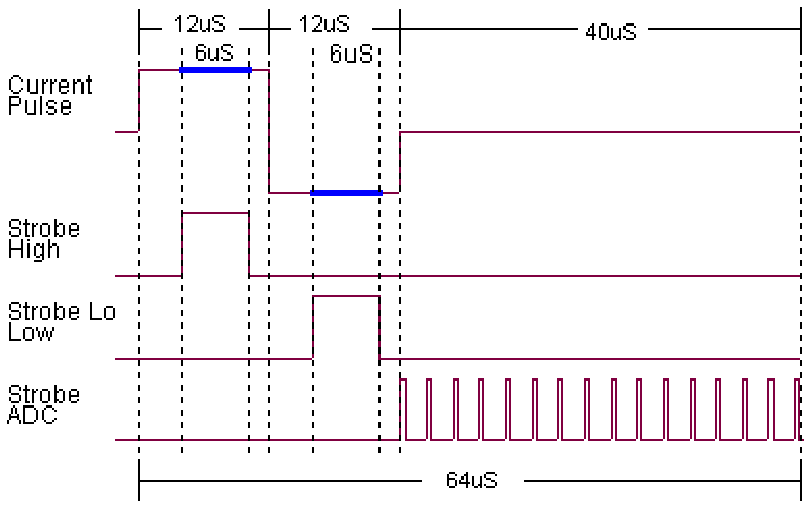


Figure 2.7: HC12 timing diagram[6]

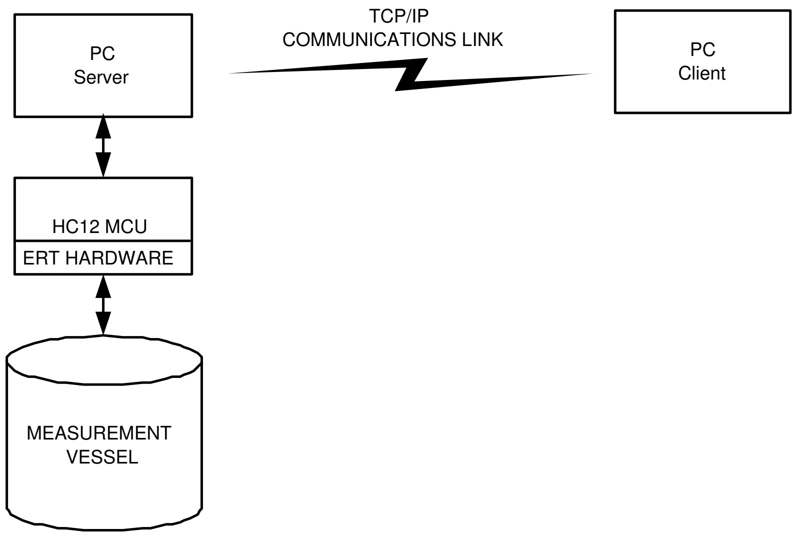


Figure 2.8: New System

Chapter 3

Image Reconstruction Principles

This chapter examines the mathematics behind image reconstruction and the use of EIT and Diffuse Optical Tomography Reconstruction Software (EIDORS) package. Mathematical theory presented in this chapter is not fully described. An in depth analysis of mathematical theory and EIDORS reconstruction is documented in [9, 10] and [11] respectively.

3.1 The Reconstruction Algorithm

The objective of the reconstruction is to calculate the conductivity distribution[1] solution in a region under investigation. Various methods exist to aquire the conductivity mapping but only a one step iteration of the Newton Raphson Method was used in this thesis project.

As metioned in section 2.1, current is injected on the boundary of the region. Poisson's equation governs the resulting electric field in the region and is given by[2]

$$\rho^{-1}\nabla \cdot \nabla V = f \tag{3.1}$$

with boundary conditions

$$V = V_0 \; on \; \partial A \tag{3.2}$$

$$\rho^{-1}\frac{\partial V}{\partial n} = J_0 \; on \; \partial A \tag{3.3}$$

where $\rho, \; V \; and \; f$ are the resistivity, voltage and current density distributions respectively, in the region under investigation. $V_0 \; and \; J_0$ are the voltage and current density at the boundary of the region. Since there is no current source within the region equation 3.1 can be reduced to

[1]Conductivity is the inverse of resistivity
[2]Equations in section 3.1 are taken from [9]

$$\nabla \cdot \sigma \nabla V = 0 \tag{3.4}$$

Equation 3.4 can further be reduced to $\nabla \cdot \nabla V = 0$ if the medium is homogeneous.

The aim of reconstruction is to find σ, given the boundary voltages as measured by the ERT instrument. This presents two interrelated problems, the 'forward' and 'inverse' problems. Their relationship is shown in figure 3.1.

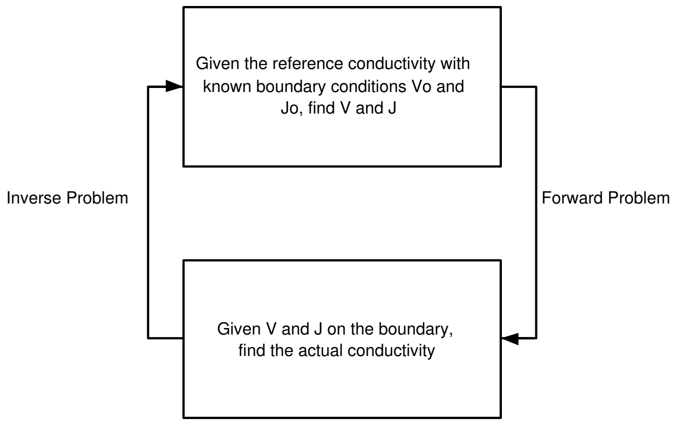

Given the reference conductivity with known boundary conditions Vo and Jo, find V and J
Inverse Problem
Forward Problem
Given V and J on the boundary, find the actual conductivity

Figure 3.1: The relationship between inverse and forward problems[12]

3.1.1 The Forward Problem

The Finite Element Method (FEM) allows the interchange of the calculus problem of equation 3.4 with a algebraic problem solvable by numeric methods. FEM results in the discretisation of the region to allow for a continuous problem to be transformed into a discrete elementwise sum by dividing the region into a finite sum of individual elements[3]. The combined discrete elements are known as a mesh as shown in figure 3.2. Interpolation functions calculate the unknown field variables within each element and are defined in terms of a nodal approximation *(figure 3.2)*. According to [9] the approximation error to a true solution is decreased as more elements are added.

Consider a 2d element as shown in figure 3.2 and a linear interpolation function defined in terms of cartesian position of the element

$$V(x,y) = \alpha_1 + \alpha_2 x + \alpha_3 y \tag{3.5}$$

where $\alpha_1$and $\alpha_2, \alpha_3$are the standard ambient voltages and according to *(x,y)* coordinates respectively. Each node must be indiviually numbered in order for computation [9]. Equation 3.5 represents a $3x3$ system of matrix equations given by the voltage at each node. The constant terms can be found by matrix algrebra of the system

[3]In 2d elements are usually triangular or quadrilateral and in 3d tetrahedral or hexahedral

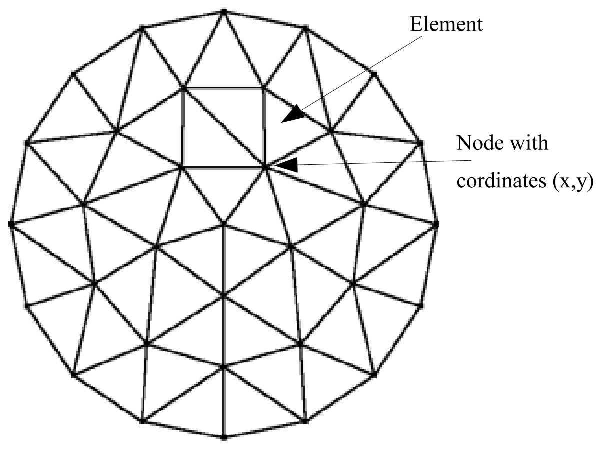


Figure 3.2: An example of a meshed plane

$$\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}^{-1} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \tag{3.6}$$

The system can be written in the form

$$V_i f(x_i, y_i) = \alpha_i \tag{3.7}$$

From the variational method an intergral can be formed relating the resistivity of an element to equation 3.7 [9]. The integral of area has the form

$$y_{ij} = \frac{1}{\rho} \int \int_A \left(\frac{\partial f_i}{\partial x} \frac{\partial f_j}{\partial x} + \frac{\partial f_i}{\partial y} \frac{\partial f_j}{\partial y} \right) dA \tag{3.8}$$

and can be thought of as a matrix system since each resistivity value has a row and column *(i,j)* component. Equation 3.8 allows the formation of a 'global master addmittance matrix' to be computed. The *Y* matrix is the admittance over the entire region and is a NxN matrix where N is the number of nodes in the mesh. A linear matrix is assembled by setting $Y_0 = 0$ and interconnecting nodes to have the same resistivity [9]. A reference node must be chosen and set to zero for a singular solution [12].

The modified matrix has the form

$$Y^{\star} v = c^{\star} \tag{3.9}$$

where v and c are the node voltage vector and node current vector respectively[4]. Through the modification, Y is a symmetric matrix and thus using Cholesky factorisation can be written as a lower triangular matrix L multiplied by its transpose $U = L^T$ [5]. Equation 3.9 becomes $L^T L v = c^{\star}$ .If a subsitution of $L^T v = q$ is made, the vector *q* can be solved

[4]The*denotes a modification to the original matrix.

[5]The *Direct LU* method see [1]

using *forward substitution* and *v* by *backward substitution*[6].

3.1.2 The Inverse problem

The 'inverse problem' is such that the actual conductivity is not known. An inital guess of the 'reference conductivity' is thus made and the 'forward problem' computed. Boundary voltages are measured and compared to those predicted by the forward solution [12]. The resulting error is minimised with Newton Raphson iteration methods until reaching the specified error limit. An 'inverting matrix' is found that transforms the error into an associated conductivity distribution, referenced to the reference conductivity. Regularisation factors *(smoothing operators)* are added to the final 'inverting matrix' to limit sudden changes (the derivative function) within a certain radius of each element. It is noted that conductivity distributions in this thesis are found from a one step algorithm without iterative calculations of the error.

3.2 EIDORS Software

The EIDORS 3D package was written by Nick Polydorides at UMIST and was released in July 2002. EIDORS is an open source package containing various MATLAB functions that combined, solve the forward problem for electrical tomography [11]. Included in the package are imaging functions used to view various aspects of the region under investigation. A listing and description of the functions used in this thesis are shown in table 3.1. A description of the EIDORS function input and output arguments are given in [13].

[6]See [9] pg 106

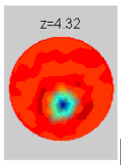
FUNCTION	DESCRIPTION
fem_master_full	This function builds the global system admittance matrix Y. Input arguments include the mesh and electrode structure of the experimental region, ground/reference node index, initial conductivity of the region and measurement scheme opposite/adjacent.
forward_solver	Solves the forward problem by using the mathematical principles in 3.1. The solver chooses the optimim method of calculation of either using the Cholesky, LL^T method or conjugate gradients based on the characteristics of Y [9]. It includes functions that minimise the error of the forward solution. An input arguement of the solver is the forward tolerance error factor.
get_3d_meas	Calculates and returns the predicted voltage distribution on all planes of the region. The electrodes carrying current are omitted during prediction. The returned voltages are sorted according to the reference node and the type of measurement scheme.
m_3d_fields	'This function calculates the measurement fields using preconditioned conjugate gradients.[14]'
jacobian_3d	The final 3-dimensional Jacobian is calculated for the *backward subsitution* to find *v*, the node voltage vector.
iso_f_smooth	*iso_f_smooth* calculates the required first order Guassian regularisation factor of the matrix Y according to the input arguement of the regularisation factor.
slicer_plot	Extracts conductivity values for a specified planar height (z axis) and produces a reconstruct image slice of the region under investigation . Deviations from the reference conductivity are shown in different colours. The image below is an example of non-conducting rod placed into the region under investigation and the resulting deviations are shown. The rod can clearly be seen to be intersecting the plane.  [13]

Table 3.1: EIDORS Functions

Chapter 4

TCP/IP Networking

This chapter provides an overview of IP interactions between two computers on a network in the form of a demonstration of transferring data over a TCP/IP communication link between a MATLAB client (a computer that has access to a service over a computer network) and C++ server (a computer providing the service). The principles used of TCP/IP communication is central to the understanding of the remote client operation.

4.1 Ethernet Network Concepts

A TCP/IP computer network consists of two or more computers connected via an Ethernet link. Communication is achieved via a packet structured protocol for resource sharing of the network and error detection and correction. The packets are addressed to the IP numbers of the respective machines. A sample of an ethernet packet frame is shown in figure 4.1 which shows the destination and source IP address in the packet header. The maximum data size per frame is 1500 bytes. The physical link may be a single UTP cable connection or via multiple hubs, switches, routers etc linking remote computers on the internet. The detailed mechanisms of TCP/IP are described in [15].

Destination IP Address (6 bytes)	Source IP Address (6 bytes)	Packet Type i.e. IP (2 bytes)	Data (Maximum 1500 bytes)

Figure 4.1: An Ethernet frame

4.2 Client/Server Demonstration

A simple MATLAB graphical user interface client was programmed to test the functions needed for TCP/IP communications. The server was programmed in Microsoft Visual

C++ echo server[1]as MATLAB is not capable of easily switching roles between server and client.

4.2.1 MATLAB Client

The MATLAB GUI was implemented using the MATLAB GUI editor. The editor is entered by typing *guide* on the MATLAB command line. Layout of the GUI is shown in figure 4.2.

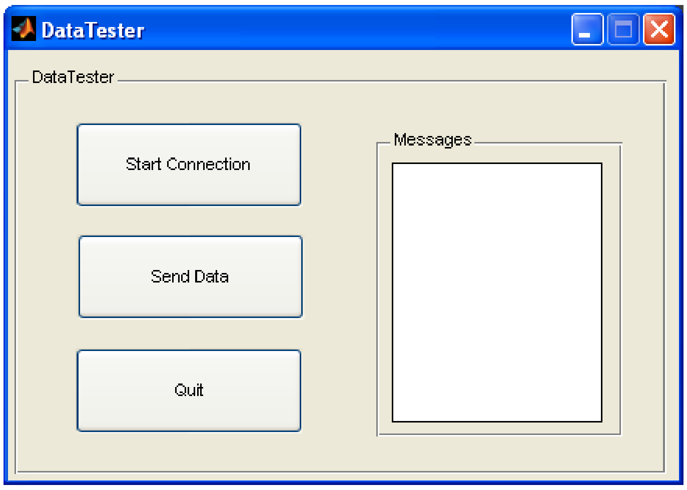


Figure 4.2: Client GUI

Objects placed on the editor frame can have their characteristics altered by entering the *Property Inspector.* The *Property Inspector* window is entered by double clicking on the specified object and as is shown in figure 4.3.

MATLAB functions used for TCP/IP communications can be found in the *Instrument Control Toolbox*. A listing and description of standard IP functions used in the transceiving of data are shown in table 4.1.

A note non the nomenclature:

- **' '** in table 4.1 refers to the input argument is entered as a *string*.

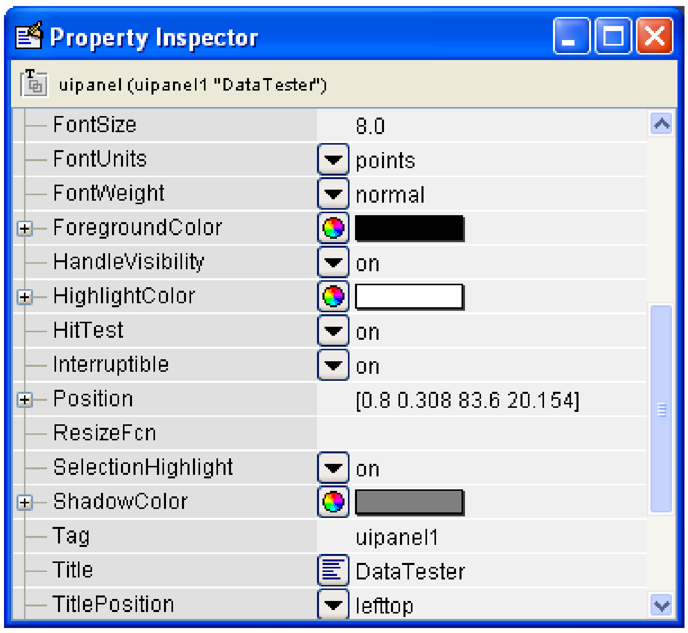
The excecution of the GUI automatically creates a M-file template with a function for each object placed on the frame window. Code entered into the *callback* functions are executed when the user event occurs e.g. a button being pushed. Global variables are defined in the function *DataTester_OpeningFcn(..)* and every subsequent function that uses the variable[2].

TCP/IP functions from table 4.1 were applied to the test client and placed in the template function *Start_Callback(hObject, eventdata, handles)* as shown in figure 4.4 . It

[1]Echos reveived data back to the sender
[2]Global variables must have the permission of *global*

FUNCTION	DESCRIPTION
Obj = tcpip('Rhost', Rport)	A TCP/IP object is created using this function. The object can be given any desired name. The *Rhost* argument of the TCP/IP function specifies the IP address of the server which can be found by running *ipconfig* in the Ms-Dos command promt. *Rport* specifies the communication port. Both server/client must communicate on this port and is thus usually chosen in the system design stage.
set(Obj,'PropertyName',PropertyValue)	Different properties of the TCP/IP object can be altered with the *set* function [16]. Among the *'PropertyNames'* altered for the use in this thesis were: • *'TimeOut'* • *'InputBufferSize'* • *'OutputBuffer'* • *'SizeByteOrder'*
fopen(Obj/Filename,'Permission')	*fopen(..)* opens the communications port specified in the evaluation of the *tcpip* function. The *'Permission'* input argument is not required as it is only used for the opening of an external file for writing/reading [16]. The function used for TCP/IP is *fopen(Obj).*
fwrite(Obj,Variable x,'Precision')	The opening of the port allows communication to commence. *fwrite(..)* writes data to the object specified by *Obj*. *Variable x* contains the data to be written to the object. *'Precision'* describes the form and control of the resulting data that will be transmitted. Examples of *'Precision'* include: • *'ushort'* writes data as an unsigned 16-bit integer. • *'uint32'* writes data as an unsigned 32-bit integer. • *'double'* writes data as a double
Variable y = fread(Obj,Size,'Precision')	*fread(..)* reads data from the object described by *Obj*. The function expects the number of bytes of data to be read by the *Size* argument. *'Precision'* is the same as in the *fwrite(..)* command.
fclose(Obj)	This function closes the port specified in the TCP/IP object *Obj* and prohibits any further communications between the client/server. In order to re-open the port the *fopen(..)* function must be implemented.

Table 4.1: Standard MATLAB IP functions



Property Inspector

uipanel (uipanel1 "DataTester")

FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	■
HandleVisibility	on
HighlightColor	□
HitTest	on
Interruptible	on
Position	[0.8 0.308 83.6 20.154]
ResizeFcn	
SelectionHighlight	on
ShadowColor	▨
Tag	uipanel1
Title	DataTester
TitlePosition	lefttop

Figure 4.3: The property inspector window

can be seen that the communications port number is 3001 and the server IP address is 137.158.228.173. In order for other functions to use the server object, the handles structure of the GUI has to be updated. A new handle is implemented by the statement *handles.Obj=NewHandle*. The functon *guidata(hObject,handles)* updates the internal GUI handles structure.

Code for data transmission is programmed in the function *Data_Callback(..)* shown in figure 4.5. On the pressing the the *Send Data* button, the user is asked to specify the size in kB of data to be sent to the server. This is accomplished by the *inputdlg(..)* statement[3] . The new handle defined in the *Start_Callback* is passed to this function by the execution of the statement *guidata(hObject,handles).*

Quit_CallBack routine only has the *fclose(server)* statement to close the communications port and therefore is not shown here.

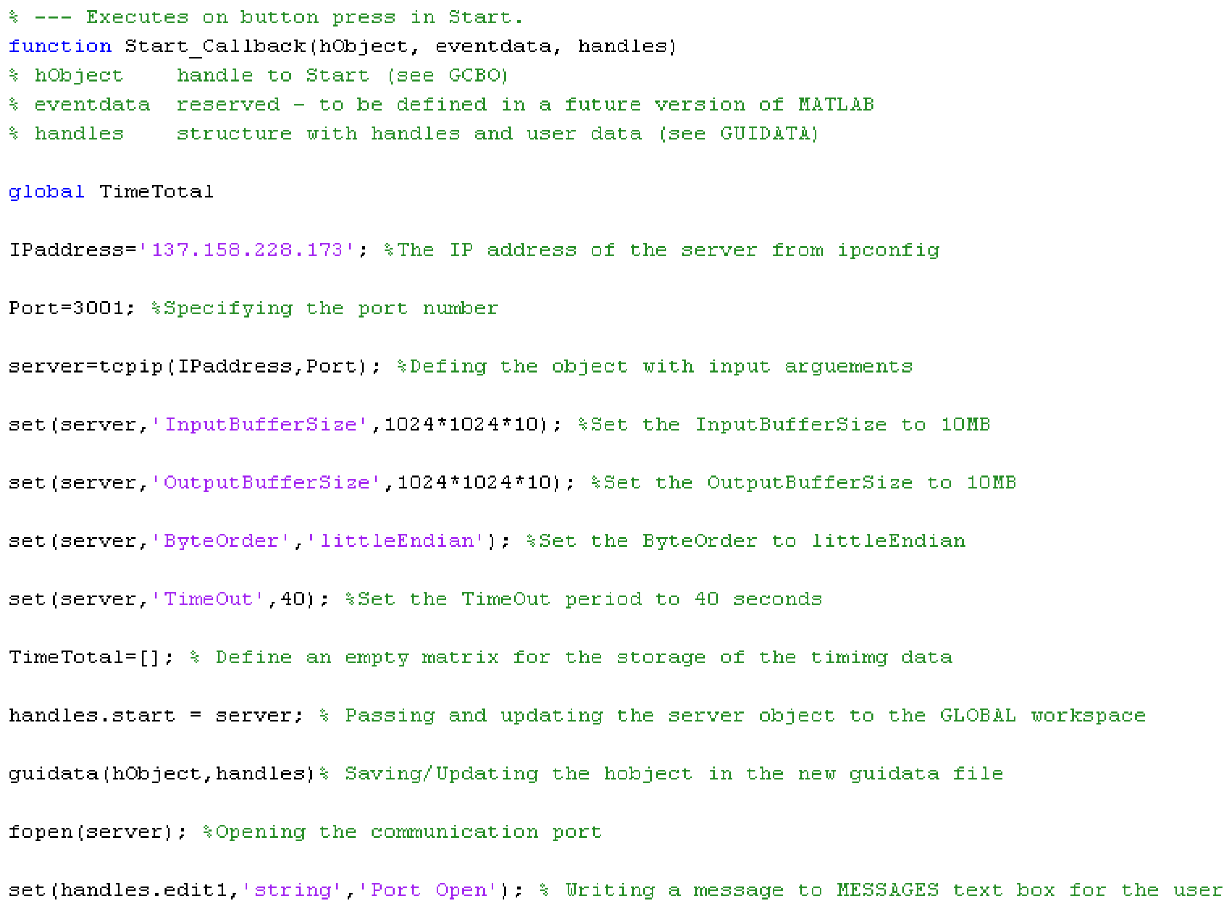
The coding shown in figure 4.5 sends the specified data to the server and reads it back into a matrix. The message textbox displays a message when the operation has been completed by using the set function to update the *'String'* property. The elaspsed time[4] taken to complete this procedure are stored in a matrix and saved to memory for offline manipulation. Transmission times are discussed in section 4.3.

4.2.2 C++ Echo Server

The C++ Echo server was written with *wX windows*, a free set of libraries allowing GUI development complied for use on any system architecture. *wX Windows* includes special

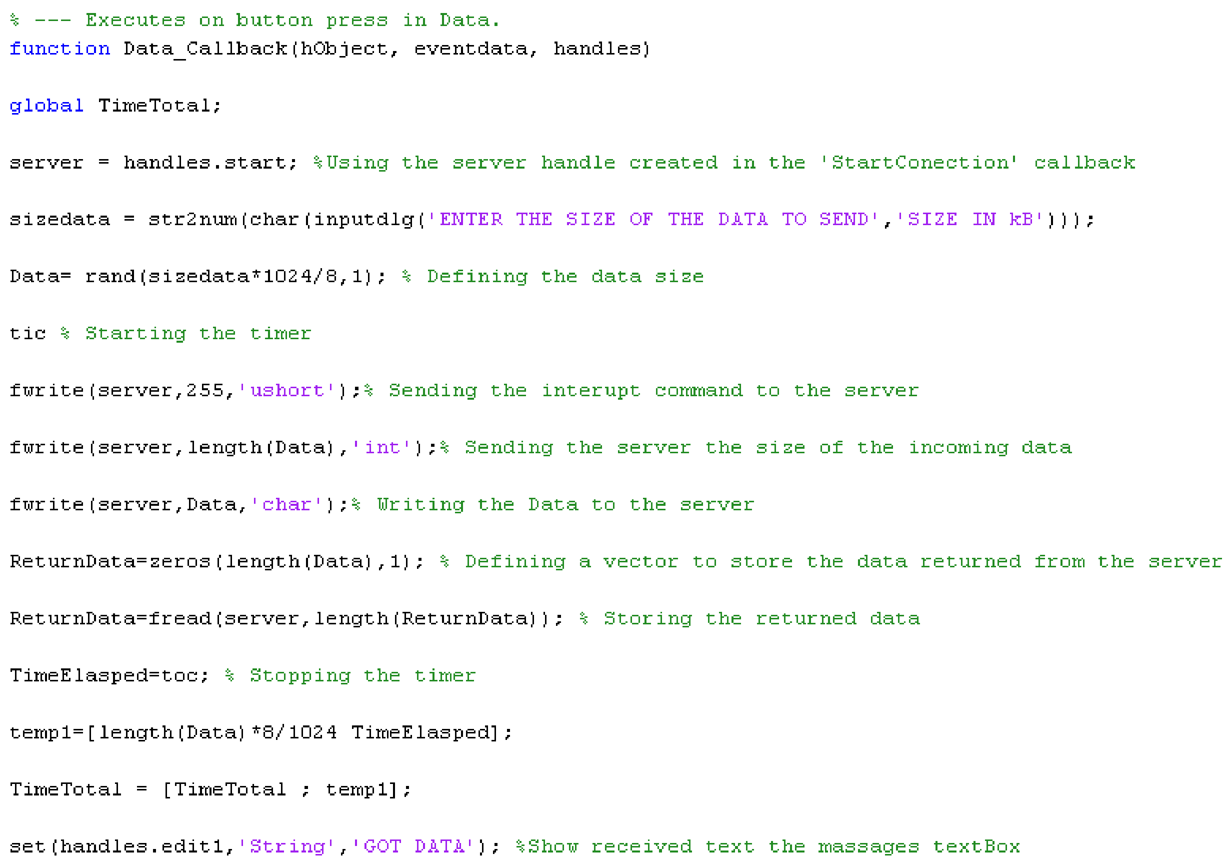
[3]Refer to [16]

[4]The time is calculated using the MATLAB tic/toc statement



```matlab
% --- Executes on button press in Start.
function Start_Callback(hObject, eventdata, handles)
% hObject    handle to Start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global TimeTotal

IPaddress='137.158.228.173'; %The IP address of the server from ipconfig

Port=3001; %Specifying the port number

server=tcpip(IPaddress,Port); %Defing the object with input arguements

set(server,'InputBufferSize',1024*1024*10); %Set the InputBufferSize to 10MB

set(server,'OutputBufferSize',1024*1024*10); %Set the OutputBufferSize to 10MB

set(server,'ByteOrder','littleEndian'); %Set the ByteOrder to littleEndian

set(server,'TimeOut',40); %Set the TimeOut period to 40 seconds

TimeTotal=[]; % Define an empty matrix for the storage of the timimg data

handles.start = server; % Passing and updating the server object to the GLOBAL workspace

guidata(hObject,handles)% Saving/Updating the hobject in the new guidata file

fopen(server); %Opening the communication port

set(handles.edit1,'string','Port Open'); % Writing a message to MESSAGES text box for the user
```

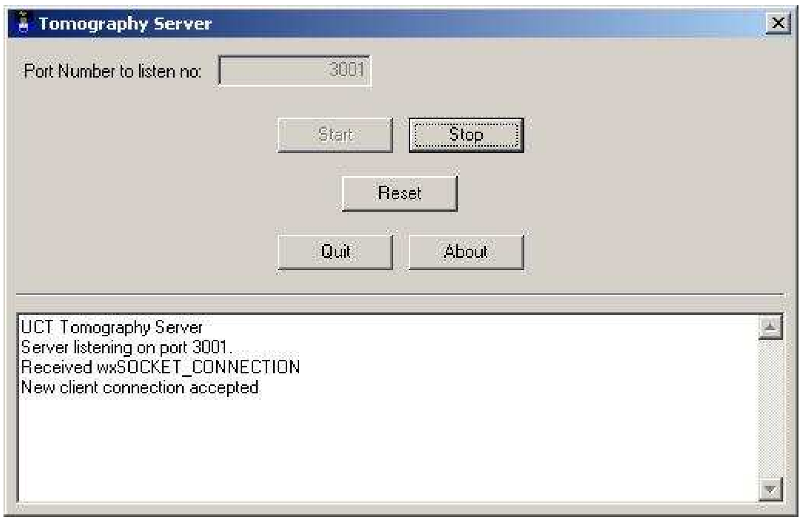
Figure 4.4: *function Start_Callback(hObject, eventdata, handles)*



```
% --- Executes on button press in Data.
function Data_Callback(hObject, eventdata, handles)

global TimeTotal;

server = handles.start; %Using the server handle created in the 'StartConection' callback

sizedata = str2num(char(inputdlg('ENTER THE SIZE OF THE DATA TO SEND','SIZE IN kB')));

Data= rand(sizedata*1024/8,1); % Defining the data size

tic % Starting the timer

fwrite(server,255,'ushort');% Sending the interupt command to the server

fwrite(server,length(Data),'int');% Sending the server the size of the incoming data

fwrite(server,Data,'char');% Writing the Data to the server

ReturnData=zeros(length(Data),1); % Defining a vector to store the data returned from the server

ReturnData=fread(server,length(ReturnData)); % Storing the returned data

TimeElasped=toc; % Stopping the timer

temp1=[length(Data)*8/1024 TimeElasped];

TimeTotal = [TimeTotal ; temp1];

set(handles.edit1,'String','GOT DATA'); %Show received text the massages textBox
```

Figure 4.5: *function Data_Callback(hObject, eventdata, handles)*

classes for implementation of TCP/IP communication[5] which were used throughout this thesis project. The graphical display of the echo server is shown in figure 4.6. *wX Windows* functions permit the server to remain in a listening mode until a client input event occurs (an opcode is received) and the required reactions are processed. However, MATLAB functions do not permit this server functionality as the *fread(..)* command listening for an input event would time out after the specified period [16].



```
UCT Tomography Server
Server listening on port 3001.
Received wxSOCKET_CONNECTION
New client connection accepted
```

Figure 4.6: The C++ Echo Server

Echo server operations only required read and write functions. The necessary *wX Windows* functions listed in table 4.2.

The server uses embedded *case* statements[6] to switch to the desired response requested by the opcode send by the client. In effect, case statements decode the opcode and the function under the corresponding opcode is switched and executed. The client initially writes the opcode of 0x00FF instructing the server to switch to the *Echo* routine *(figure 4.7)*. The server proceeds to read the size of the data and stores it in the variable *sizeofdata* with precision of a 16-bit integer. Every element of the received data is read separately and stored into *buffer* array with size that of *sizeofdata*. The *buffer* is then written back to the client without the preceding *sizeofdata* as the client already has this value stored in memory.

4.3 Data Transfer Testing

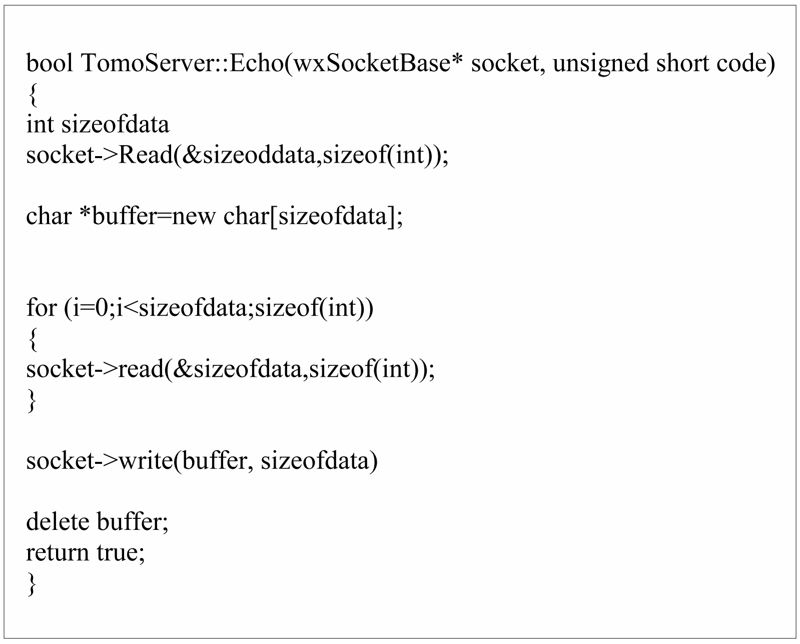
The preformance of the client/server system was tested by finding the time taken for data to be echoed by the server. Timing data sizes of 1kB, 2kB, 5kB, 10kB, 100kB, 1MB and 5MB were conducted and the results shown in table 4.3.

[5]Refer to [8] for information on *wX windows* TCP/IP setup

[6]*case* statements are described in [16, 8]

FUNCTION	IMPLEMENTATION	DESCRIPTION
Read	*sock->Read(&Variable x,sizeof(Precision))*	The socket or port specified in the C++ socket setup is read and the information is stored in predefined *Variable x.* Examples of *Precisions* used include: • *char* • *wxUint16* • *wxFloat32* • *wxShort*
Write	*sock->Write(&Variable y,sizeof(Precision))*	The explanation is the same as for the *Read* command except that *Variable y* is written to the port to be read by the client.

Table 4.2: *wX Windows* read and write functions



```
bool TomoServer::Echo(wxSocketBase* socket, unsigned short code)
{
int sizeofdata
socket->Read(&sizeoddata,sizeof(int));

char *buffer=new char[sizeofdata];


for (i=0;i<sizeofdata;sizeof(int))
{
socket->read(&sizeofdata,sizeof(int));
}

socket->write(buffer, sizeofdata)

delete buffer;
return true;
}
```

Figure 4.7: *Echo* subroutine code

DATA SIZE (kB)	TIME (s)
1	0.2848
2	0.2850
5	0.4956
10	0.4597
100	0.8351
1024	8.3092
5120	41.5025

Table 4.3: Results of data transfer testing

The UCT network is rated at 100Mb/s but experimental results showed the average transfer rate to be 0.97Mb/s. The data rate is approximately 100 times slower than the rated value. This test is not conclusive as echo server coding has not been refined for optimal transferring rates.

In terms of the data required for image reconstruction, the four independent layers measurement scheme used in this thesis performs 256 samples/layer with each sample equal to one byte of data. There will thus be 1kB of data per frame. The results of testing showed that with the present coding system, an approximate 123 frames/s can be received for processing and therefore the system is capable of ’real time’ remote operation.

Chapter 5

Remote Client Code Operation

This chapter will look at the designed functionality of the MATLAB client in terms of the operations and protocols required for remote operation. The remote client code sequencing for the continuous reconstruction of updated data sets then is described with reference to the actual code.

5.1 Client Functionality

The design functionality of the remote client required nine independent operations interacting via IP. More complex functions are implemented using these operations inside *M-file template* callback functions [1] .

5.1.1 The Opcode Design

Design of a 16 bit opcode structure allowed independent operations to be sorted according to their mode of operation. The bitwise representation of the designed structure is given in table 5.1. The design allows 128 modes of 256 operations per mode. Opcodes are sent to the server were they are decoded and the specified operations on the ERT hardware performed. The correct acknowledgement of the received opcode requires the server to resend the opcode with b_{15} set before any further data exchange occurs. Successful completion of the requested operation is signaled to the client by the returning of a frame with all bits set *(0xFFFF)*.

ACK	MODE							OPERATION							
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Table 5.1: The opcode structure

Four operational modes have been predefined, namely the request, transmit, hardware and control modes. The use of only four modes permits the subsequent development of the

[1] See section 4.2.1

network protocol and implemented IP operations. Operational modes with their respective codes are illustrated in table 5.2.

MODE	RESPECTIVE MODE OPCODE $(b_8 - b_{14})$
REQUEST	0000000 *(binary)* or 0 *(decimal)*
TRANSMIT	0000001 *(binary)* or 1 *(decimal)*
HARDWARE	0000002 *(binary)* or 2 *(decimal)*
CONTROL	0000003 *(binary)* or 3 *(decimal)*

Table 5.2: Modes and their respective opcodes

5.1.2 Independent Operations

Independent operations were grouped according to their mode of IP interaction with the server are shown in table 5.3.

5.1.3 Error Detection and Handling

TCP/IP is a state full protocol which ensures the order of transceived data by means of embedded error detection and retransmit routines that are hidden from the user [15]. The onus is left to the client software to perform error checking. Implementation of the acknowledge bit and successful frame in the designed opcode structure, permit elementary error detection and handling to occur, if the server fails to complete the operation. The summary of the error handling process is shown in the flowchart of figure 5.1. A user input event (on the remote client) triggers a timer to be started and the opcode sent to the server. The client waits for the correct acknowledgement frame before process data is sent. Non-arrival, signals a user-defined response. The server executes the opcode command and returns the required data to the client if no errors are encountered. Returned data is only verified if the server returns the successful frame to the client. Reception of the successful frame stops the timer and updates a log file with information on the execution of the function. All MATLAB client functions have this embedded error detection and handling technique. An extract from the log file is given in appendix B.

5.2 Remote Client Code Sequence

The remote client code consists of over 2000 lines, excluding EIDORS or external functions used to implement the remote system and thus a detailed description of each individual function is not presented in this thesis. Instead, an analysis of the sequencing procedures *(figure 5.2)* and important code segments (extracted from the remote client MATLAB code) required for continuous data set capture and image reconstruction are

OPERATION	OPCODE	DESCRIPTION
Get N frames with current off	0x0002 *(hex)* 2 *(decimal)*	The client requests *N* frames *(each the average of M frames)* with current off. The server then aquires and returns the raw measurement data.
Get N frames with current on	0x0003 *(hex)* 3 *(decimal)*	The client requests *N* frames *(each the average of M frames)* with current on.
Start continuous capture with current on	0x0006 *(hex)* 6 *(decimal)*	Client instructs the server to continuously capture, buffer and then discard frames from the ERT unit with current on.
Stop continuous capture with current on	0x0007 *(hex)* 7 *(decimal)*	The client instructs the server stop continuous capture with current on.
Get N frames continuous	0x0008 *(hex)* 8 *(decimal)*	This operation interupts the server while in its continuous capture mode and instead of discarding the captured frames, transmits the *N* frames to the client.
Start continuous capture with current off	0x000A*(hex)* 10 *(decimal)*	Instructs the server to continuously capture, buffer and then discard frames from the ERT unit with current off.
Stop continuous capture with current off	0x000B*(hex)* 11 *(decimal)*	The server stops continuous capture with current off.
Set current level	0x0101*(hex)* 257 *(decimal)*	Current injection level is sent to the server.
Upload sequence table	0x0102*(hex)* 258 *(decimal)*	The sequence table and HC12 executable[2] be uploaded to the server for data capture to begin.
Set samples per frame	0x0200 *(hex)* 512 *(decimal)*	This parameter is set by the client and enables the server to know when an iteration of the measurement sequence has been completed.
Reset ERT hardware	0x0201 *(hex)* 513 *(decimal)*	The client instructs the server to reset the HC12 MCU.
Set number (M) of frames to average	0x0202 *(hex)* 514 *(decimal)*	*M* represents the number of frames the server must capture and store before the averaged frame is returned.
Set DAC output voltage	0x0303 *(hex)* 771 *(decimal)*	The DAC output of the Eagle Card is updated to provide real-time feedback control. This operation will be implemented in chapter 7.

Table 5.3: Independent IP operations for remote client operation

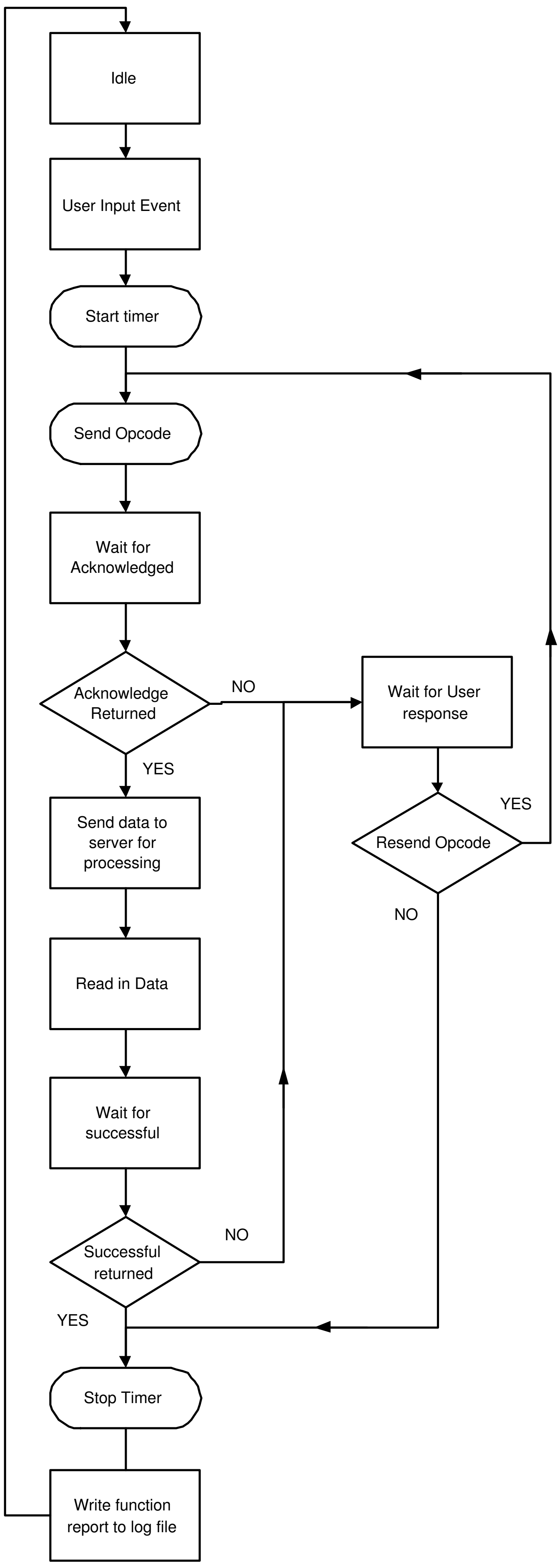
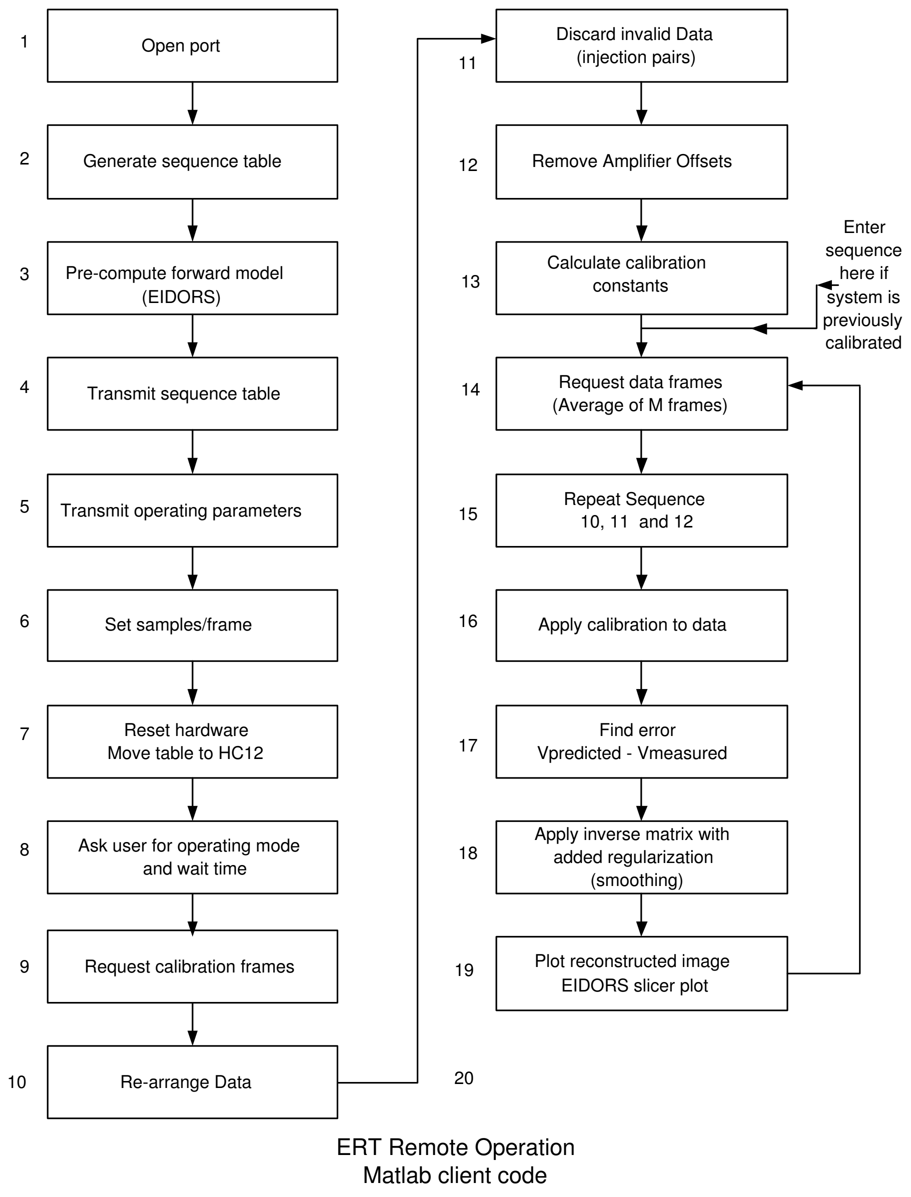


Figure 5.1: The error flow diagram

discussed and presented respectively. Each operational procedure *(figure 5.2)* has been given a reference number corresponding to its discussion. Server functions and ERT reactions to client operations are not shown as it is assumed they perform the required task without error. Full documentation of the remote client coding is provided on a CD attached to this thesis.



1 Open port
2 Generate sequence table
3 Pre-compute forward model (EIDORS)
4 Transmit sequence table
5 Transmit operating parameters
6 Set samples/frame
7 Reset hardware Move table to HC12
8 Ask user for operating mode and wait time
9 Request calibration frames
10 Re-arrange Data
11 Discard invalid Data (injection pairs)
12 Remove Amplifier Offsets
13 Calculate calibration constants
Enter sequence here if system is previously calibrated
14 Request data frames (Average of M frames)
15 Repeat Sequence 10, 11 and 12
16 Apply calibration to data
17 Find error Vpredicted - Vmeasured
18 Apply inverse matrix with added regularization (smoothing)
19 Plot reconstructed image EIDORS slicer plot
20
ERT Remote Operation
Matlab client code

Figure 5.2: The flow of the code

1. The IP connection is opened.

2. The 4 layer independent measurement sequence required for the multiplexors implementation is generated with the function *makeseq(..)*. This function returns the measurement scheme as a matrix and writes the information in a predefined format to the file *seq_table_4L_indep.txt*. The sequence is generated in 3 byte groupings specifying the current injection layer, the source/sink electrodes and measurement

layer. *makeseq(..)* was adapted from the multilayer 3D sequence generation function *tomo_seq_gen_3d_v1(..)* written by [17].

3. This stage allows pre-computation of the ‘forward problem’ solution and ‘inverse matrix’ using EIDORS functions[3]. Mesh data required for the forward solution was extracted from the file *9_Layer_3D_mesh.mes* using the *LoadSequence(..), Load3DMesh(..)* and *CreateMeasurementPatterns(..)* functions, written by [18]. A description of these functions are shown in table 5.4 The EIDORS function

FUNCTION	DESCRIPTION
LoadSequence(..)	Inputs the formatted *seq_table_4L_indep.txt file* and returns the 3 byte groupings of (2) as zero offset vectors of drive layer, sensing layer and source/sink electrodes.
Load3DMesh(..)	This function returns 4 matrices. The matrices include information about the numbering of tetrahedral node elements and their connections to other nodes, cartesian positioning of each node within the mesh, the electrode dimensions in terms of node connections and mesh surface elements.
CreateMeasurementPatterns(..)	This function transforms the generated sequence table into a binary based table for and EIDORS reconstruction functions.

Table 5.4: Functions used to extract mesh data

get_3d_meas is used to calculate the full predicted voltage distribution matrix for every element in the mesh. The mesh implemented in this thesis had 3456 spatial elements within the electrode vessel. In order to calculate the error term[4] for reconstruction only the predicted voltages at the boundary electrodes are required, thus it is necessary to extract the correspomding vaules from the predicted voltage distribution. Coding used for the extraction *(figure 5.3)* was adopted from [18]. Error terms from subsequent captured fames are multiplied by the same ’inverting matrix’ in the single step reconstruction algorithm and thus only a single computation of the ’inverting matrix’ is required. EIDORS code to calculate the ’inverting matrix’ with added regularisation is given in figure 5.4. A description of the EIDORS input variables is given in [13]. Finally, the sequence table created in *makeseq(..)* is concatenated with the *HC12ERT.bin* binary executable to form the ’modified executable’. The MSB of the last byte of the ’modified executable’ is set to allow for the HC12 to continuously loop the measurement sequence until halted by the server PC.

[3]See section 10

[4]See section 3.1.2

```
frame1ref = [];
frame2ref = [];
frame3ref = [];
frame4ref = [];
m_ind_check = [];
for i = 1:61:3904     % loop through VrefH in 61 blocks
    if i < 977        % injection on the first plane
        frame1ref = [frame1ref; VrefH( i:i+12)];
        m_ind_check = [m_ind_check; indH(i:i+12 , : )];
    elseif i < 1953   % injection on the second plane
        frame2ref = [frame2ref; VrefH( (i:i+12)+16)];
        m_ind_check = [m_ind_check; indH((i:i+12)+16 , : )];
    elseif i < 2929   % injection on the third plane
        frame3ref = [frame3ref; VrefH( (i:i+12)+32)];
        m_ind_check = [m_ind_check; indH((i:i+12)+32 , : )];
    else              % injection on the fourth plane
        frame4ref = [frame4ref; VrefH( (i:i+12)+48)];
        m_ind_check = [m_ind_check; indH((i:i+12)+48 , : )];
    end
end
v_calc = [frame1ref' frame2ref' frame3ref' frame4ref'];
```

Figure 5.3: Extraction of the predicted boundary voltages from the full voltage distribution matrix

```
[J] = jacobian_3d(I, elec, vtx, simp, ground_node, mat_ref, zc, v_f, df, forward_tol, '{y}');
[reg] = iso_f_smooth(simp,vtx,3,1);
Inverting_Matrix = (J'*J + tfactor*reg'*reg)\J';
save Inverting_Matrix Inverting_Matrix;
```

Figure 5.4: Code segment for calculation of the ’inverse matrix’

4. The ’modified executable’ is written to the server *(opcode 0x0102)* where is decoded into assembler for use with the HC12 MCU.

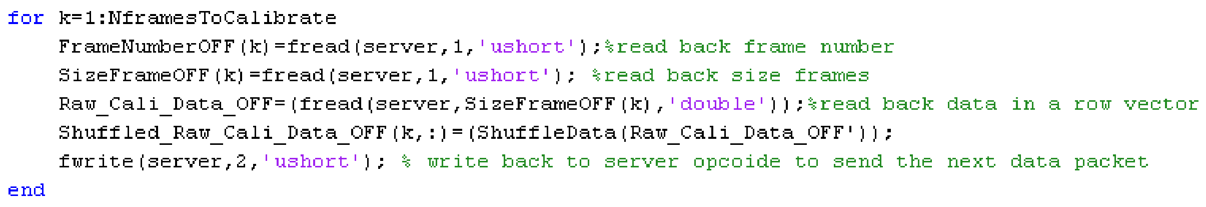
5. Operating parameters of current level *(opcode 0x0101)* and frames to average *(opcode 0x0202)* are sent to the server. The current level is inversely proportional to the conductance properties of the aqueous solution in the measurement vessel. The Eagle Card ADC’s used in the ERT instrumentation saturate at 2.5V and therefore the current level must be set to constrain the measured voltages within this boundary. Measured voltages should typically be in the order of 1V. Averaging of the *M* data frames, averages out noise within the combined frame. Noise can be further minimised by making the time required to acquire *M* frames an integer factor of the mains oscillating period. Positive and negative interference terms should cancel within one mains’ cycle.

6. The number of samples per frame *(opcode 0x0200)* is written to the server. An internal variable from the *makeseq(..)* function is returned with this value.

7. The HC12 MCU is reset *(opcode 0x0201)* and the embedded EPROM downloaded with the machine code version of the ’modified executable’ and current level via serial communications between the C++ server and HC12. Reset also sets the DAC output on the Eagle Card to 0 Volts *(opcode 0x0303)*.

8. The user is asked if the continuous capture mode must be entered on the server *(opcode 0x0006 or 0x000A)*. It has been observed that the voltage measured between the first electrode pair *E1* and *E16* is biased on the first data frame recorded after startup. This is believed to be related to the static conditions occuring during the idle state of the ERT system which may result in polaristation at the electrodes *E1* and *E16* which are selected by the multiplexor during the rest state. The phenomenon is currently being investigated. It was found that by continuous sampling and discarding the first few frames this problem is avoided. Time between start of continuous capture and returning of frames to the client is user-defined.

9. This command requests *N* raw data frames of both current off and current on - where *N* is user-specified - *(opcode 0x0003 or 0x0008)* for the calibration of the instrument. The code segment for the general acquisition of data set with current on or off is shown in figure 5.5. For every captured frame the server writes back the frame number and frame size followed by the raw data frame. The last line of code instructs the server to send the next frame of data. Frames captured for instrument calibration are known as *calibration frames* and are required to find amplifier offsets and discrepancies in the gain constants between the predicted and measured voltages [10]. Frames are returned to the client as a column vectors. They are averaged separately to aquire two calibration frames, one with current off and one with current on. Differential amplifier offsets are obtained by data



```
for k=1:NframesToCalibrate
    FrameNumberOFF(k)=fread(server,1,'ushort');%read back frame number
    SizeFrameOFF(k)=fread(server,1,'ushort'); %read back size frames
    Raw_Cali_Data_OFF=(fread(server,SizeFrameOFF(k),'double'));%read back data in a row vector
    Shuffled_Raw_Cali_Data_OFF(k,:)=(ShuffleData(Raw_Cali_Data_OFF'));
    fwrite(server,2,'ushort'); % write back to server opcoide to send the next data packet
end
```

Figure 5.5: Code segment for receiving a *N* raw data frames

capture with current off as only the ambient voltages in the measurement vessel are measured with frames of zero current injection. It is assumed these offsets are time independent and thus there is no need to conduct further experimentation with current injection disabled. Calibration data is usually acquired with a homogenous saline solution in the region of investigation.

10. The forward problem algorithm expects the data to be formatted in particular manner, however the measured data from the ERT instrument does not match this format. The difference is that the EIDORS code expects the first measurement samples on each injection to come from electrodes *E1* and *E2* where as the first measurement taken by the ERT hardware on each injecion is between electrode *E16* and *E1*. Therefore the measured data must be shuffled to match the format of the EIDORS forward problem solver. The mismatch problem can be solved by the software reshuffling on every captured frame. The function *ShuffleData(..)* written by [18] shuffles the raw data to coincide with the EIDORS frame structure. *ShuffleData(..)* inputs a raw data frame from *(9)* as a column vector and returns it as a row vector. Both data frames captured in *9* are passed through this function for reshuffling.

11. In the injection of current on an electrode pair, the measurements from the adjacent two electrodes are invalid due to the affect of cross coupling. Raw voltage measurements include these unwanted pairs. The removal of these pairs is accomplished in software on the remote client by the function *RemoveCorrupted(..)*, written by [18]. It can be shown that for every 16 samples in a 4 layer independent measurement scheme there are three invalid pairs thus resulting in the new data frame having 832 samples. Frames from *(10)* are passed through this function and offset data is stored in memory for use on subsequent raw frame captures.

12. Offsets are subtracted from the data frame with current injection enabled from *11*.

13. The calibration factors are calculated. These factors are constant terms needed to adjust the gain constants between the measured calibration frame (*after the necessary manipulations)* and the predicted EIDORS voltages. The ratio is given by

$$CalibrationFactors = \frac{Vpredicted}{Vcalibration} \tag{5.1}$$

The calibration factors are assumed to remain constant unless the medium inside the measurement vessel is permanently disturbed. Under these conditions the calculation of the calibration factors would have to be repeated. Calibration factors are stored in memory for use on subsequent frame captures. The code for the calculation of the calibration factors from steps *11-13* is shown in figure 5.6

```
save Shuffled_Raw_Cali_Data_OFF Shuffled_Raw_Cali_Data_OFF;
save Shuffled_Raw_Cali_Data_ON  Shuffled_Raw_Cali_Data_ON;

raw_cali_data_OFF=mean(Shuffled_Raw_Cali_Data_OFF);
raw_cali_data_ON=mean(Shuffled_Raw_Cali_Data_ON);

save raw_cali_data_OFF raw_cali_data_OFF;
save raw_cali_data_ON raw_cali_data_ON;


Uncorr_raw_cali_current=RemoveCorruptedSamples(raw_cali_data_ON, no_of_layers, drive_lay, drive_elec, sense_lay);
Uncorr_raw_cali_NO_current=RemoveCorruptedSamples(raw_cali_data_OFF, no_of_layers, drive_lay, drive_elec, sense_lay);

save Uncorr_raw_cali_NO_current Uncorr_raw_cali_NO_current;

UnCorr_Cali_Data_NoOffsets=(Uncorr_raw_cali_current-Uncorr_raw_cali_NO_current);

save UnCorr_Cali_Data_NoOffsets UnCorr_Cali_Data_NoOffsets;

[Calibrating_Factors] = (v_calc)./UnCorr_Cali_Data_NoOffsets;

save Calibrating_Factors Calibrating_Factors;
```

Figure 5.6: Code extract for calculating the calibration factors

14. Each iteration of the loop requires a new raw data frame for image reconstruction in order to implement continuous remote imaging. The user specifies the time between frame captures. MATLAB client code logic allows the user to enter the sequence at this stage *(figure 5.2)* if calibration has previously been completed with the same measurement sequence. The forward solution of the predicted voltages would be inherently different of the measurement sequence is altered.

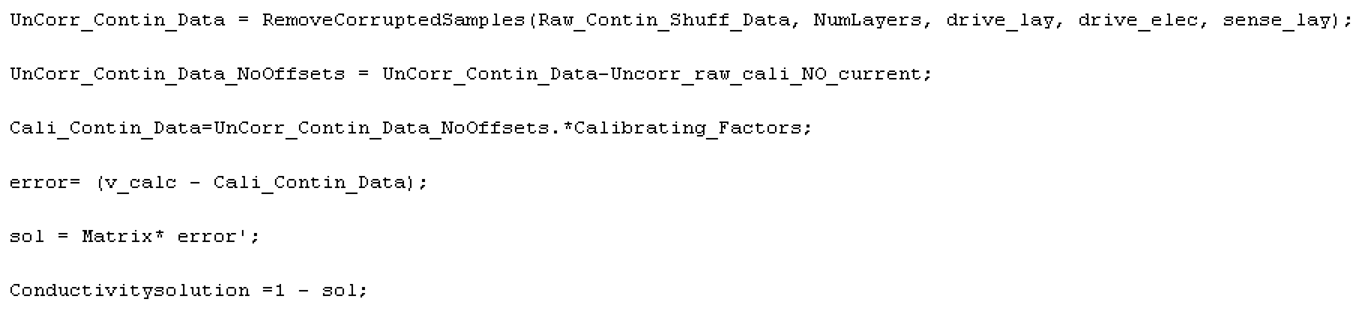
15. The data manipulation of sequence steps *10-12* are applied to the new raw data frame.

16. The data is then calibrated by vector multiplication with calibration factors and thus any deviations in the amplitude of the calibrated frame from the calibration frame inherently represent changes in the conductivity.

17. Deviations are found by finding the error between the predicted forward model values and the calibrated data frame.

18. The 'inverting matrix' with added regularisation computed in *(3)* is applied to the error from *(17)* and the vector is the error between the actual and reference conductivity distributions[5]. The actual conductivity solution is found by subtracting this conductivity error from the reference conductivity. Example code for the execution of steps *15-18* is shown in figure 5.7.

[5]See section 3.1.2



```
UnCorr_Contin_Data = RemoveCorruptedSamples(Raw_Contin_Shuff_Data, NumLayers, drive_lay, drive_elec, sense_lay);

UnCorr_Contin_Data_NoOffsets = UnCorr_Contin_Data-Uncorr_raw_cali_NO_current;

Cali_Contin_Data=UnCorr_Contin_Data_NoOffsets.*Calibrating_Factors;

error= (v_calc - Cali_Contin_Data);

sol = Matrix* error';

Conductivitysolution =1 - sol;
```

Figure 5.7: Code extract for finding the actual conductivity distribution

19. Reconstructed images are updated on the output screen as they are calculated. The EIDORS plotting function *slicerplot(..)* is used to plot 'sliced' planes through the measurement vessel at different heights. The magnitude of each element in the conductivity solution is represented by a linear colour mapping. Mesh elements that have their conductivities altered by a disturbance will result in a different represented colour.

Chapter 6

The Client GUI and Image Analysis

This chapter shows the final implementation of the MATLAB GUI and dicusses the obtained reconstructed images.

6.1 GUI Operating Window

The *TomoClient* operating window with default parameter values is shown in figure 6.1. User defined parameters can be updated in the specified edit textboxes before the specified button is invoked. Sequential operations of figure 5.2 are executed via button callback functions with corresponding names. Operations that require a user input are conveyed via the implementation of popup windows. The user is shown the status of the operations by updates on the *Status/Messages* textbox.

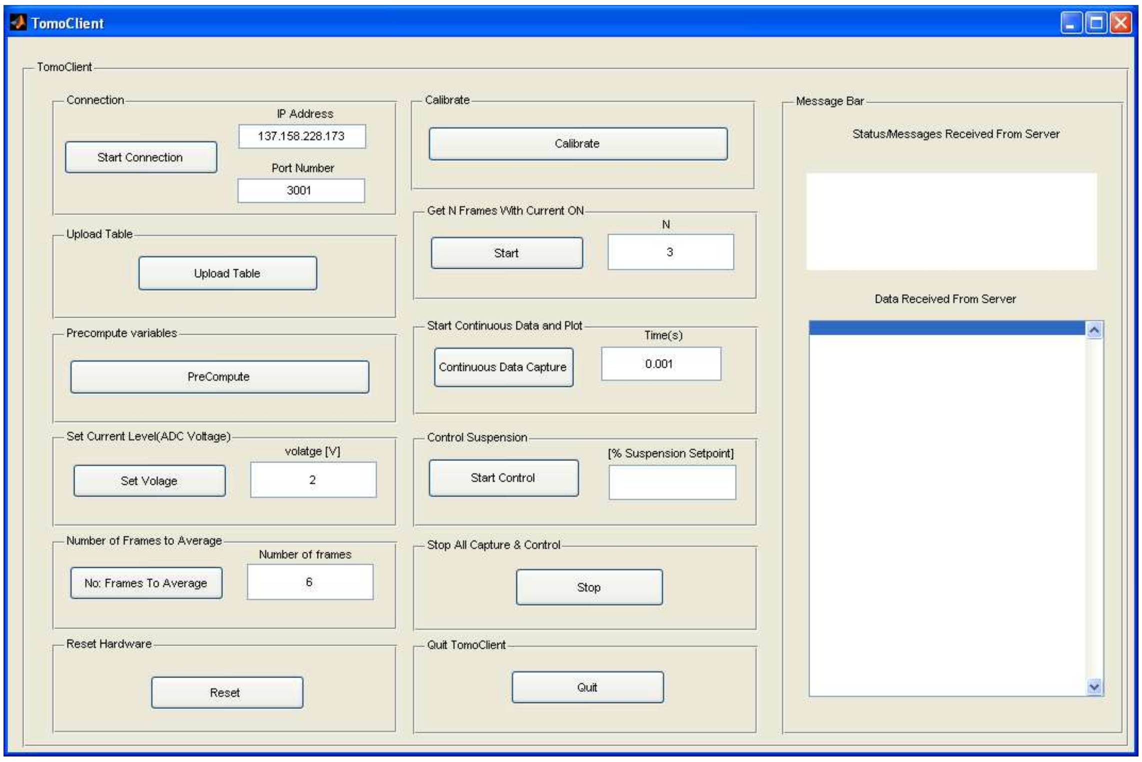
The start button under the *Get N frames With Current ON* heading retrieves *N* frames of raw data and stores them into memory. This data can be loaded into MATLAB for manipulation by the entering of *load RawDataON* onto the MATLAB command line.

The *Continuous Data Capture* and *Start Control* buttons implement the necessary operations to continuously capture, reconstruct and plot a data frame. In the case of the later feedback control is also implemented. These looping procedures are stopped by the pressing of the *Stop* button.

A note must be made that:

- The *UploadTable* callback function performs both the transmitting of the sequence table and the samples/frame parameter to the server.
- Precompute calculations of mesh parameters, forward solution, 'inverting matrix' with added regularisation are all performed by the *PreCompute* button.
- Data manipulation functions of *ShuffleData(..)* and *RemoveCorrupted(..)* for operations aquiring data sets are executed internally in the appropriate callback function and are hidden from the user.

- Operations needed to calculate and store the calibrating factors are executed by the *Calibrate* callback function.



IP Address 137.158.228.173
Port Number 3001
N 3
Time(s) 0.001
volatge [V] 2
[% Suspension Setpoint]
Number of frames 6

Figure 6.1: TomoClient

6.2 Image Plots and Analysis

This section shows screen shots of the data frames and EIDORS *slicerplot(..)* image reconstructions.

A raw data frame waveform is shown in figure 6.2. The observed ’hooped’ behaviour can be explained in terms the current density within the electrode vessel during current injection. Current density *(J)* is given by the formula

$$J = \sigma \frac{V}{d} \tag{6.1}$$

where d and V are the distance from the injection source and the measured potential difference respectively.

With the use of a constant current source (constant current density) voltages measured on the boundary of the measurement vessel will vary according to the distance between sense and drive electrode pairs . The ’hoops’ are ’periodic’ as the measurements are repeated around a circular ring of electrodes. It can be seen that there are indeed discontinuities in the waveform from the measurements taken from adjacent pairs to the drive pair.

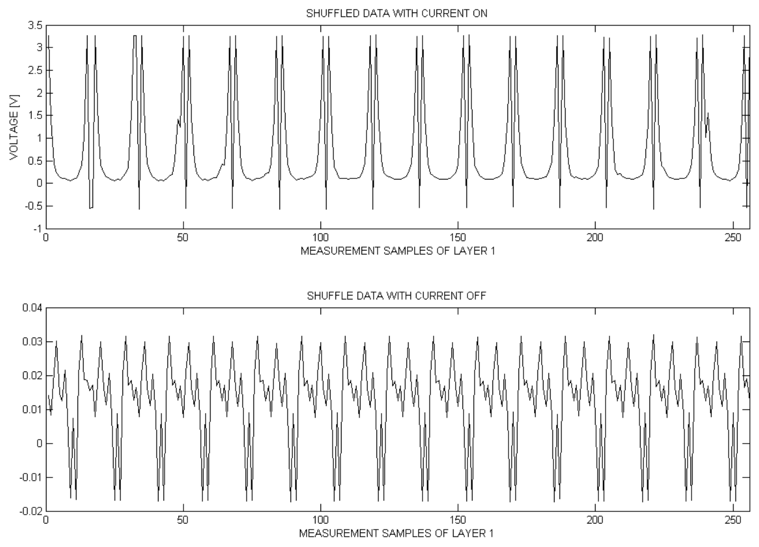


Figure 6.2: A raw frame of measurements on the first layer with current on and off

The EIDORS predicted voltages are compared to the a calibrated data frame from measurement with only the homogeneous solution in the measurement vessel in figures 6.3 and 6.4 (The voltages are with respect to the measurement scheme). Through calibration of the raw data frame, they are almost identical. The 'hoops' are seen to have a higher magnitude in the first layer and last layer of figure . This is caused by the increasing of the current density along upper and lower boundaries of the measurement vessel and hence an increase in the sensed potential difference *(equation 6.1)*.

A noticable difference between the calibration frame *(figure 6.4)* and a calibrated frame with a non-conductive rod placed in the measuement vessel *(figure6.5)* can indeed be seen . The plastic rod decreased the the overall conductivity of the medium and thus increased the measured boundary voltages *(equation 6.1)*.

Actual reconstructed conductivity distributions with only a homogeneous solution in the measurement vessel were observed to have random fluctuations around the reference conductivity. These fluctuations were attributed to drifting of the ERT instrumentation and are presently under investigation. The conductivity distribution changes to figure 6.6 as the non-conducting rod is placed in the tank.

The EIDORS *slicerplot(..)* function was used to create a reconstruction slice of a plane in the vessel at different heights. A non-conducting rod was placed inside the measurement vessel. The *slicerplot* of figure 6.7 shows the rod as a black circle through otherwise grey slices. The grey areas correspond to the reference conductivity colour mapping and the black sections represent major changes in conductivity. In theory, the insertion of an object into the measurement vessel only changes the grey area affected by the disturbance. However, it was observed that the entire colour mapped region changed as an object was placed into the vessel. The cause of this was attributed to the internal auto-scaling of the *slicerplot(..)* function. The heights of the respective planes are shown.

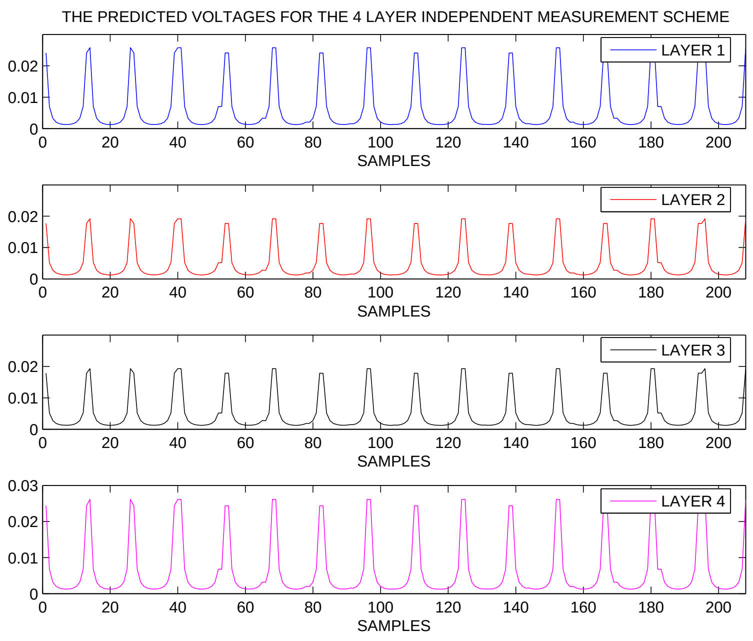


Figure 6.3: Predicted voltages

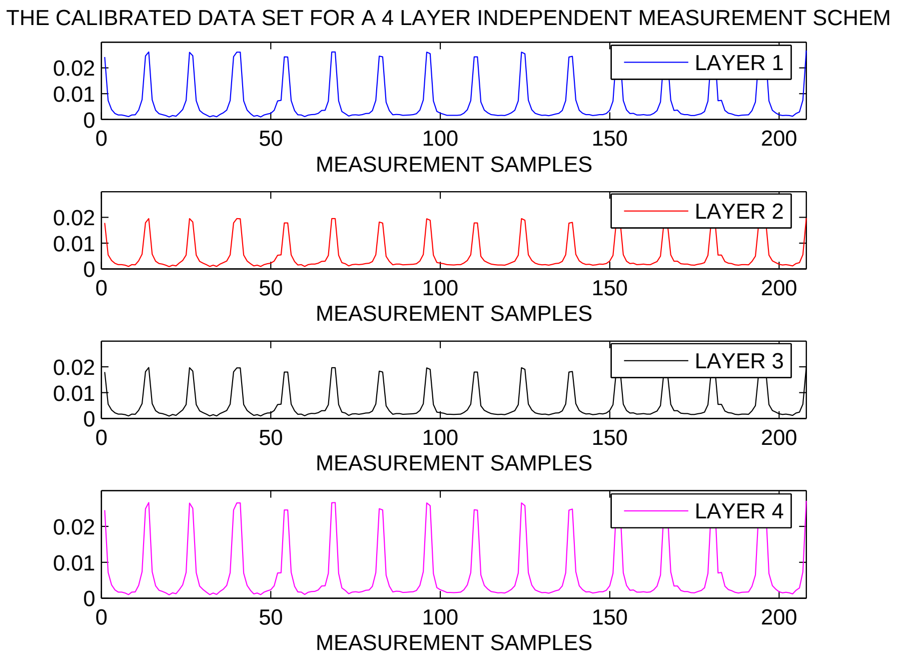


Figure 6.4: A calibrated data frame

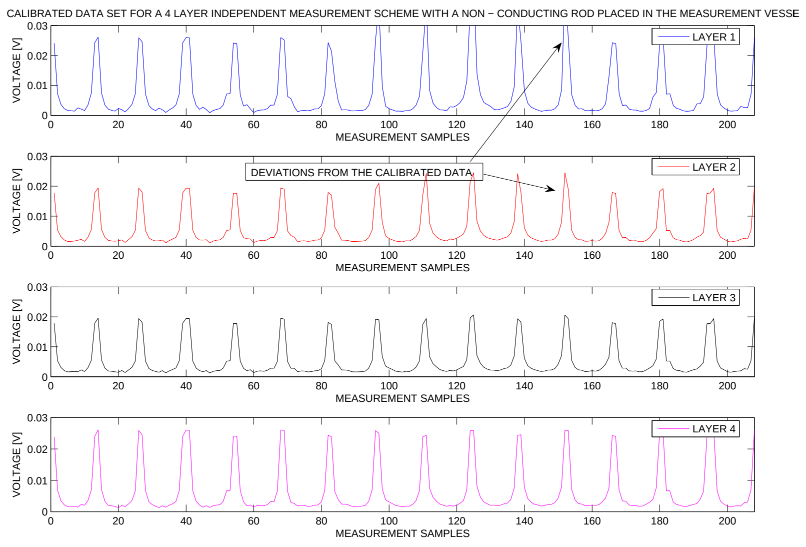


Figure 6.5: The deviations from the calibration frame

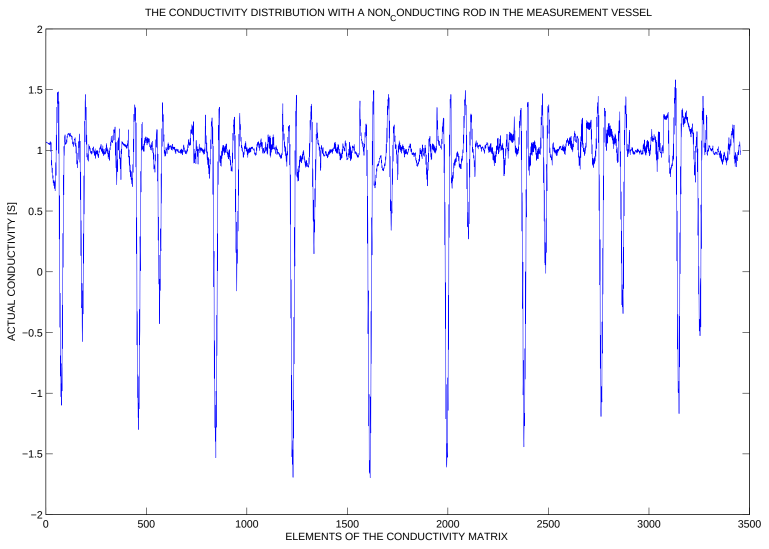


Figure 6.6: Actual conductivity distribution with a rod is placed in the homogeneous solution

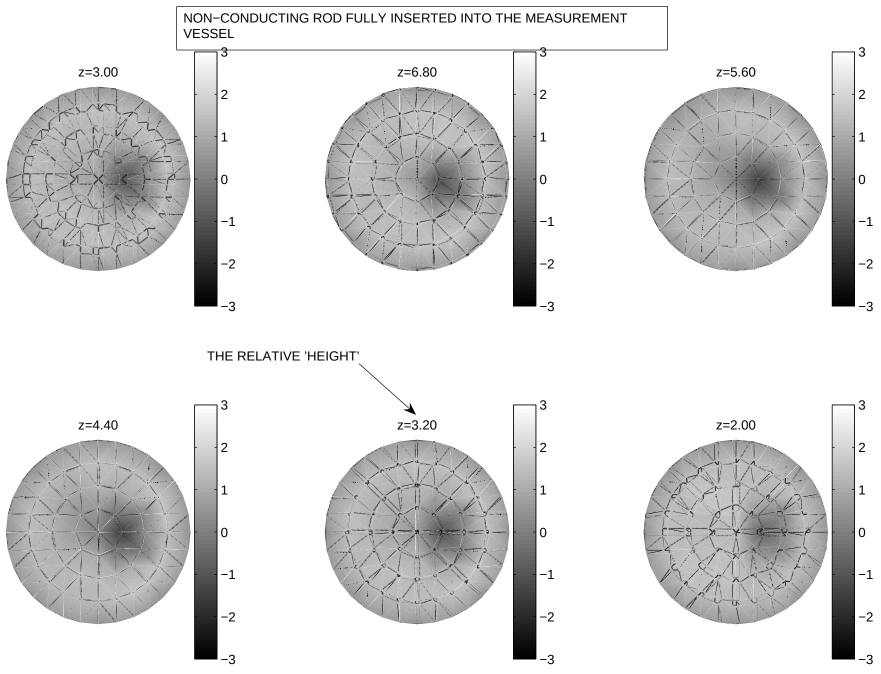


Figure 6.7: Non-conducting rod fully inserted

A *slicerplot* of the measurement vessel with a rod inserted halfway into the measurement vessel is shown in figure 6.8. It can indeed be see that the rod has passed through the two uppermost planes at 'height' planes *z=6.8* and *z=5.6* . The cause of the white centre in some of the plots is unknown. It was suggested that this problem could be solved if a finer mesh of more elements were used in the forward model, but due to time constraints of this thesis project this idea was never implemented.

A maximum image refresh rate of three refreshed updated screens per second was observed with the imaging of the element wise conductivity distribution and one refreshed updated per second with image application with the *slicerplot(..)* function. The *Log-File.txt* suggested that the time for image plotting using either imaging schemes is in the order of 50ms and thus there was nothing to suggest that the MATLAB client coding needed to be optimised. The root cause was the performance of the IP networking strategy between client and server and the C++ server coding not being optimised for IP data transfer.

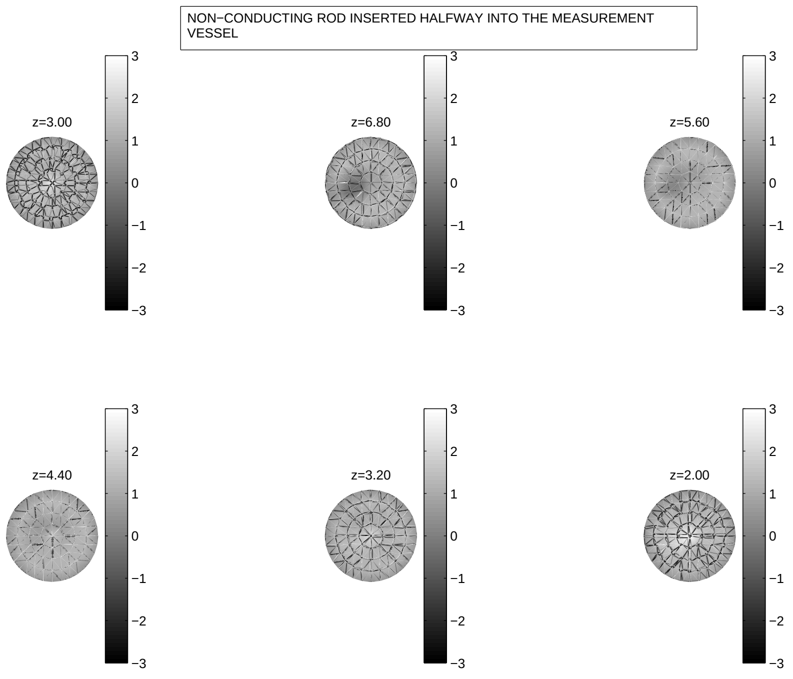


Figure 6.8: Non-conducting rod inserted halfway

Chapter 7

Conductivity Feedback Control

This chapter describes the use of the remote *TomoClient* in a feedback control application. The control of the conductivity within a region of the measurement vessel is investigated.

7.1 Feedback Control Design

An impeller was attached to a DC motor and placed in the tank. Plastic beads were then added. The rotation of the impeller causes the beads to be suspended in the homogeneous solution and is shown in figure 7.1.

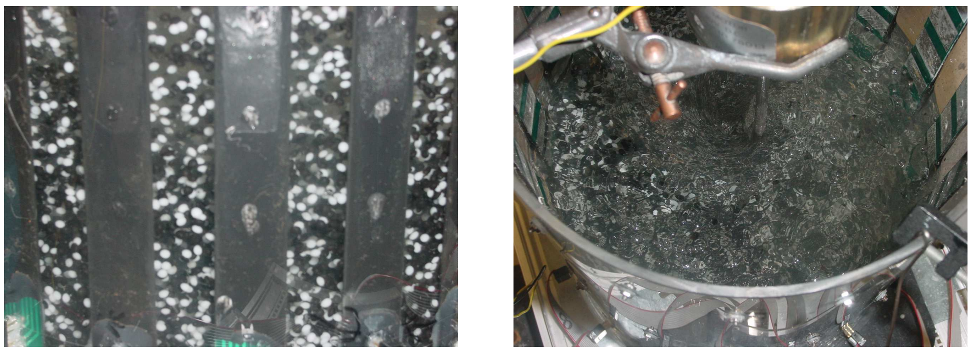


Figure 7.1: Mixing of the beads in the measurement vessel

Increased motor speed causes an increase in suspension but a decrease in conductivity values from the reference conductivity[1]. Feedback is needed to control the motor speed so that the setpoint conductivity within a specified volume was tracked by the controller.

A simplified block diagram of the feedback design is shown in figure7.2. The user specifies a control volume[2] in the vessel and inputs the conductivity setpoint. The setpoint conductivity is set in the *Control Suspension* block of figure 6.1 and must have a value less than the reference conductivity to ensure setpoint tracking when the beads are suspended.

[1]See section 3.1.2

[2]Volume within which condutivity control is implemented.

Updated conductivity values are acquired through raw data capture and manipulation using the functions described in section 4.2.1. The conductivity solution corresponding to the control volume is then extracted from the full conductivity distribution solution. The *conductivity error* is the error between the setpoint and actual conductivity within the control volume. This error is then fed into the software controller function *controller(..)* where a value, corresponding voltage needed to be written to the server DAC, is returned. The motor driver [3] converts the DAC output voltage to a proportional motor speed.

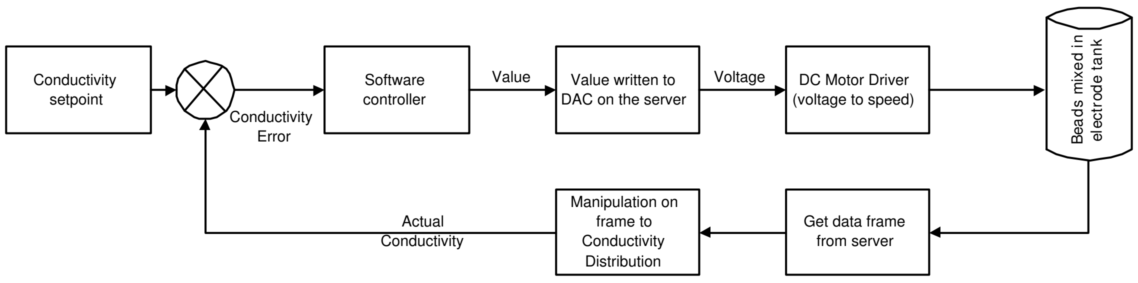

Conductivity setpoint
Conductivity Error
Software controller
Value
Value written to DAC on the server
Voltage
DC Motor Driver (voltage to speed)
Beads mixed in electrode tank
Get data frame from server
Manipulation on frame to Conductivity Distribution
Actual Conductivity

Figure 7.2: The control Block diagram

7.2 Step Test

A step test was conducted to attempt to model the system and design a suitable controller. The voltage is manually injected into the motor driver and the resulting conductivity within the control volume observed.

The step test result with a constant input voltage of 1.5V is shown in figure 7.3. The step test resembles a first order oscillatory system.

It was observed that the mixing beads exhibited a random oscillatory motion *(figure 7.1)*. The time taken to update the DAC output voltage was found to be approximately 0.65s per update.[4] Under certain conditions, a vortex formed in the centre of the tank, creating an air pocket within the medium. This further contributed to the randomness of the step test fluctuations. The slow update time and randomness of the system limits the control preformed in this thesis.

7.3 Control Functions

Control functions were added to the loop sequence in figure 5.2:

- The DAC is output voltage is updated via writing the new voltage value to the server *(opcode 0x0303)*. The function was implemented in the GUI with the same logic and error detection and handling techniques discussed in section 5.1.3.

[3]The motor driver circuit is given in appendix C.
[4]See LogFile in appendix B

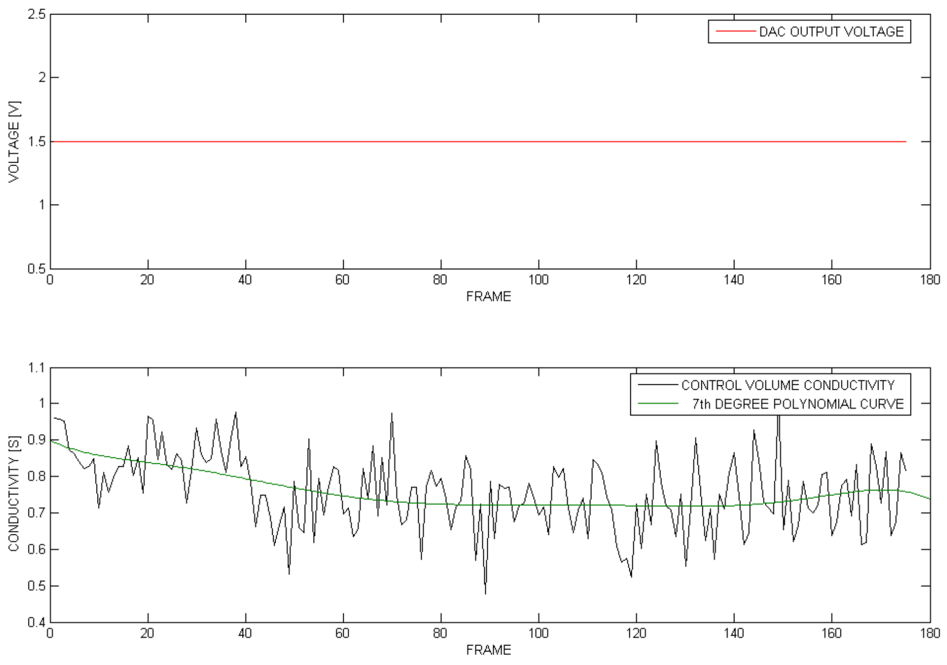


Figure 7.3: Conductivity of a control volume during a step test

- Control begins by asking the user which controller is to be implemented. Two software controller functions were coded in the function *controller(..)*. They are listed in table 7.1.

CONTROLLER	DESCRIPTION
Proportional Integral (PI)	Previous conductivity errors are integrated. If the new conductivity error is zero the voltage written to the DAC remains constant.
Proportion, Integral, Differential (PID)	A three mode control action where the controller output voltage has three terms: proportional, integral and derivative. The PID constants require tuning to achieve optimal response.

Table 7.1: The controllers

7.4 Reconstructed Responses

Proportions used in these controllers were found by an iterative adjustment method until the best performance was observed. The performance of the software controllers were tested and results concluded on the basis of oscillation variance and frame captures to ‘settle‘.

7.4.1 Proportional Integral (PI)

The PI controller ’settled’ in approximately 100 frames with an oscillation variance of approximately 0.4 [Siemens]. An input distrubance - in the form of a non-conducting rod - was placed in the measurement vessel and the controller action to return the conductivity to setpoint can clearly be observed in figure 7.4.

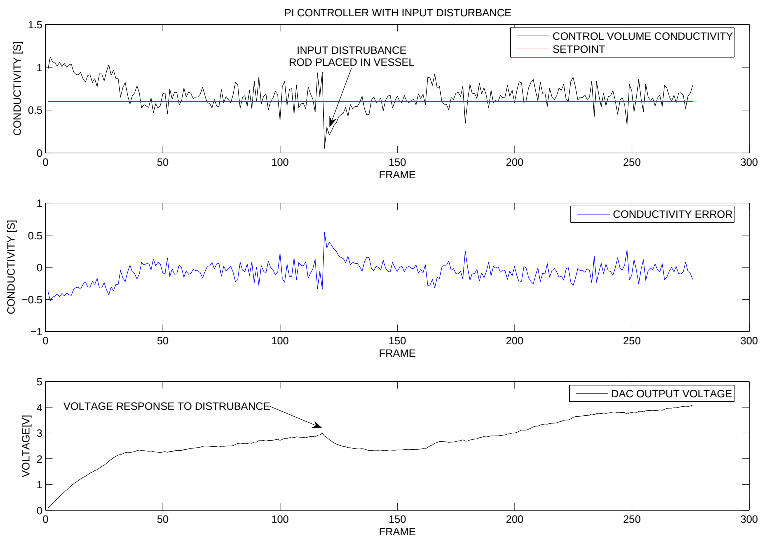


Figure 7.4: The PI controller

7.4.2 Proportional Integral Differential (PID)

Performance of the PID controller is shown below in figure 7.5 . The controller exhibited a faster ’settle’ time of approximately 50 frames but the oscillation deviance from the setpoint has deterioated to approximately 0.5 [Siemens].

The application chosen for feedback control could not be controlled by either of the designed controllers. In a discussion with E.W. Randall it was suggested that no evident control could be implemented because of the suspended beads’ random motion and slow update time of the DC motor.

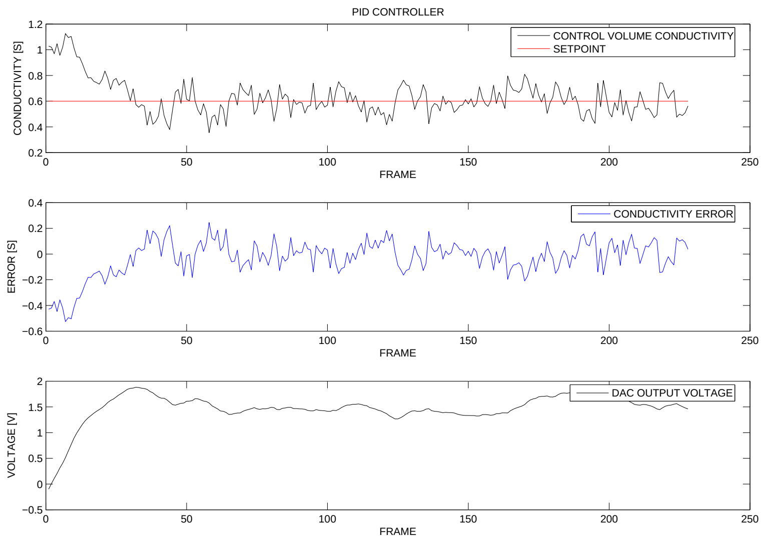


Figure 7.5: The PID controller

Chapter 8

Conclusions

The following conclusions were drawn:

1. Remote operation and imaging of the ERT instrumentation has been implemented with a combination of MATLAB and EIDORS functions interfacing with a C++ server.
2. An interactive MATLAB GUI provides a user-friendly, easily modifiable front end to the designed functionality of the remote client.
3. The data frame rate over TCP/IP only allows for the image reconstruction at an update rate of approximately one image per second and thus the developed system can be used for ’real-time’ imaging for processes having a time constant less that this value.
4. The MATLAB client has demonstrated its application in remote feedback control.

Chapter 9

Recommedations

Although the aims of this thesis have been satisfied and remote operation of the ERT instrumentation implemented, time constraints did not allow aspects of the system to be improved and developed. The following improvements are suggested:

1. **Increase image update speed by optimising the C++ server code**. The C++ server coding requires optimisation in terms of routines for IP interactions and MATLAB interfacing.

2. **Use a finer mesh for increased spatial accuracy.** The use of a finer mesh increases the spatial resolution of the forward model predicted voltages and thus of the conductivity solution. The increased resolution allows tracking of smaller disturbances in the measurement vessel.

3. **Investigate other control applications**. The potential for the remote control of a slower, more stable system with a time constant less than the update speed of the remote client requires investigation.

Appendix A

The Measurement Sequence

Appendix B

An extract from the LogFile.txt

```
START CONNECTION
Date:15-Oct-2005 10:17:58
Time Taken To Transcieve Data: 4.066209e-002 [s]
Action: open
UPLOAD TABLE
Date:15-Oct-2005 10:18:00
Time Taken To Transcieve Data: 2.356864e-001 [s]
Action: FFFF
UPLOAD TABLE
Date:15-Oct-2005 10:18:00
Time Taken To Transcieve Data: 2.678246e-001 [s]
Action: FINISHED ROUTINE
THE NUMBER OF SAMPLES PER FRAME
Date:15-Oct-2005 10:18:00
Time Taken To Transcieve Data: 3.279691e-002 [s]
Action: FFFF
NUMBER OF SAMPLES PER FRAME
Date:15-Oct-2005 10:18:00
Time Taken To Transcieve Data: 4.039932e-002 [s]
Action: FINISHED ROUTINE
SET CURRENT
Date:15-Oct-2005 10:18:00
Time Taken To Transcieve Data: 1.908894e-001 [s]
Action: FFFF
SET NUMBER OF FRAMES TO AVERAGE
```

```
Date:15-Oct-2005 10:18:01
Time Taken To Transcieve Data: 2.876023e-002 [s]
Action: FFFF
Reset
Date:15-Oct-2005 10:18:04
Time Taken To Transcieve Data: 3.383498e+000 [s]
Action: FFFF
CALIBRATE CURRENT ON
Date:15-Oct-2005 10:18:41
Time Taken To Transcieve Data: 2.814013e+001 [s]
Action: GOT THE CALIBRATION DATA WITH CURRENT ON
CONTINUOUS CAPTURE STOPPED
Date:15-Oct-2005 10:18:41
Time Taken To Transcieve Data: 3.629952e-001 [s]
Action: STOPPED CONTINUOUS CAPTURE WITH CURRENT [ON]
CALIBRATE CURRENT OFF
Date:15-Oct-2005 10:19:03
Time Taken To Transcieve Data: 2.230220e+001 [s]
Action: GOT THE CALIBRATION DATA WITH CURRENT OFF
CONTINUOUS CAPTURE STOPPED
Date:15-Oct-2005 10:19:04
Time Taken To Transcieve Data: 6.461370e-001 [s]
Action: STOPPED CONTINUOUS CAPTURE WITH CURRENT [OFF]
CALIBRATE
Date:15-Oct-2005 10:19:04
Time Taken To Transcieve Data: 1.210816e+000 [s]
Action: CALIBRATION FINISHED
CONTROL DATA
Date:15-Oct-2005 10:20:18
Time Taken To Transcieve Data: 1.086439e-001 [s]
Action: FRAMES FOR CONTROL CAPTURE [ON] MODE SENT
CONTROL
Date:15-Oct-2005 10:20:18
Time Taken To Transcieve Data: 3.597305e-001 [s]
Action: GOT DATA FRAME FOR CONTROL
CONTROL
```

```
Date:15-Oct-2005 10:20:18
Time Taken To Transcieve Data: 7.483223e-003 [s]
Action: DATA MANIPULATION FOR CONTROL COMPLETED
CONTROL
Date:15-Oct-2005 10:20:18
Time Taken To Transcieve Data: 2.647415e-003 [s]
Action: CONTROL VOLTAGE COMPUTED
CONTROL
Date:15-Oct-2005 10:20:18
Time Taken To Transcieve Data: 7.159672e-002 [s]
Action: CONTROL PLOT DONE
CONTROL
Date:15-Oct-2005 10:20:18
Time Taken To Transcieve Data: 1.103794e-001 [s]
Action: VOLTAGE SUCCESSLY WRITTEN TO DAC
```

Appendix C

The motor driver control circuit

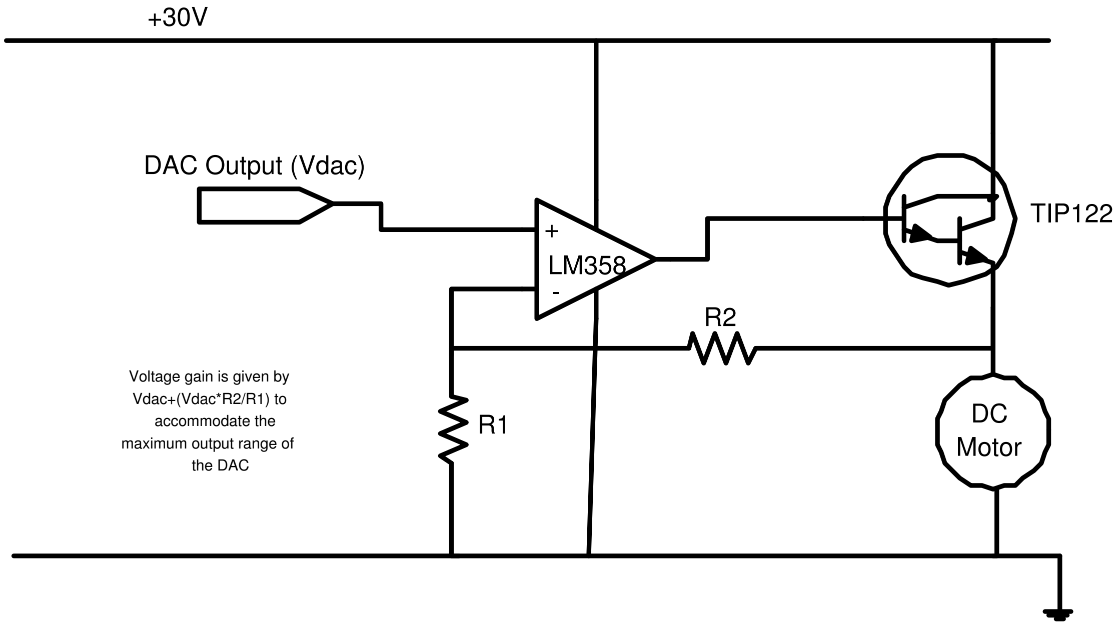

+30V
DAC Output (Vdac)
+
LM358
-
TIP122
R2
Voltage gain is given by Vdac+(Vdac*R2/R1) to accommodate the maximum output range of the DAC
R1
DC Motor

Figure C.1: The motor driver control circuit

Bibliography

[1] Dickin F and Wang M 1996 ‘*Electrical Resistance Tomography for process applications*‘, Measurement Science and Technology. 7 247-260

[2] Cilliers, J.J., Xie, W., Neethling, S.J., Randall, E.W., Wilkinson, A.J. 2001. *‘Electrical resistance tomography using a bi-directional current pulse technique’*. Meas. Sci. Technol., 12:997-1001.

[3] Wilkinson et al. *’A 1000-Measurement Frames/Second ERT Data Capture System With Real-Time Visualisation’*, IEEE sensor Journal 2005

[4] Beck,M.S.,Williams,R.A. 1996. *’Process tomography: a European innovation and its applications’*, Meas. Sci. Technol. 7 215-224

[5] Wilkinson, A.J., Randall, E.W., Durrett, D., Naidoo, T., Cilliers, J.J. 2003 ‘*The Design of a 1000 frames/second ERT Data Capture System and Calibration Techniques Employed’*. 3rd World Congress on Industrial Process Tomography. Banff.

[6] Randall,E.W.,Wilkinson,A.J, Long,T.M, Sutherland, A., Manchester U.K. 2004, *’A high speed speed current pulse electrical resistance tomography system for dynamic process monitoring’* ESDA 2004-58219

[7] EW Randall, AJ Wilkinson, TM Long, A Collins, 2005. *‘The design of a flexible multi-layer ERT system and an evaluation of its performance’*, 4th World Congress on Industrial Process Tomography, Aizu, Japan.

[8] Wx Windows embedded help documentation.

[9] Hua, P. Woo EJ, 1990. ‘*Reconstruction Algortihms - Electrical Impedance Tomography*‘. ed. J.G. Webster, The Adam Hilger Series on Biomedical Engineering, Bristol: IOP Publishing Ltd. pp.97-136.

[10] Naidoo, T. 2002. ’*Signal & Image Processing for Electrical Resistance Tomography*’ Masters Thesis, Department of Electrical Engineering, University of Cape Town.

[11] POLYDORIDES,NP,LIONHEART WRB, *’A MATALAB toolkit for three dimensional electrical impedence tomography: A contribution to the EIDORS project’*, Meas. Sci. Technol. 13 (December 2002) 1871-1883

[12] Long, T.M. 2003. *'A Software Front End for a Real-Time Electrical Resistance Tomography Imaging System'* Undergraduate Thesis, Department of Electrical Engineering, University of Cape Town.

[13] Parbhoo, M., *'A Guide to Using EIDORS for Three Dimensional Electrical Impedance Tomography'* (December 2004)

[14] EIDORS embedded help documentation.

[15] Arnett,M.F et al .1995 . *'Inside TCP/IP Second Edition'*, New Rider Publishing, Indianapolis,IN

[16] MATLAB embedded help documentation.

[17] Wilkinson, A.J. 2005 'ERT measurement scheme code'

[18] Long, T.M. 2005 'Data manipulation code'

Appendix A

Software Source Code