NEW CONTROL SOFTWARE FOR CERBERUS 3D NANOINDENTATION SYSTEM

by

Bhakt Vatsal Trivedi
(2010041)

SUPERVISORS
Mr. Saket Sourav
Research Engineer
IIITDM - Jabalpur

Dr. Graham L. W. Cross
PI, CRANN Nanotechnology Institute
Trinity College Dublin

Computer Science and Engineering Discipline

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DESIGN AND MANUFACTURING JABALPUR

Interim Report 2

INTRODUCTION:-

After the completion of the initial version of the GUI, the next target was to complete the GUI and connect the GUI to the hardware. The initial strategy for connecting the GUI was through the Ethernet port but on more discussion and thinking another solution came to light- serial port (RS-232). So finally this option was selected and connection was established between the old control system and the modern GUI.

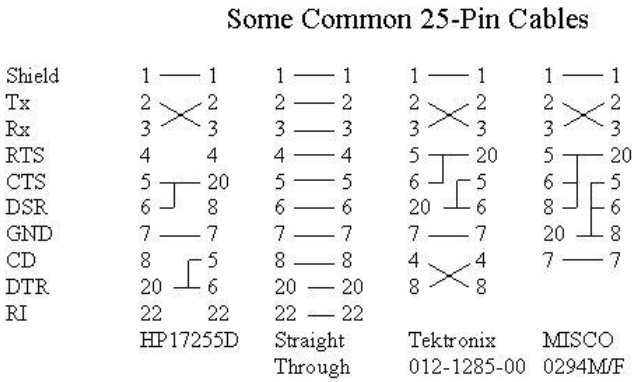
Another task that was accomplished during this phase was updating the GUI. Many new features were added to the GUI, which are discussed in the following sections.

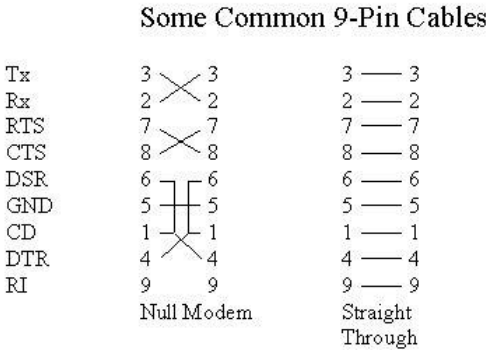
PROGRESS:

SERIAL (RS-232) PORT

There are three main issues regarding serial communication:

1. CABLING:
   We have options of 9 pin or 25 pin serial communication.





2. HANDSHAKING:
   There are two main handshaking protocols in serial communication:

XON/XOFF Handshaking:

1. When the buffer is about full, XOFF is sent.
2. When there is room again in the buffer, XON is sent.

Hardware Handshaking:

1. If the receive buffer is empty, turn on DTR and wait for a character to arrive.
2. Get a character from the receive buffer and turn off DTR.

3. COMMUNICATION PARAMETERS:
   Plugging the cable between the computer and the device is not all that is required to make them talk. The devices must speak the same language in order to understand each other. It doesn't matter what it is, but it is essential they be the same! The language consists of the baud rate, data bits, parity, and stop bits. We must determine which of these parameters can be set on the device, what they can be set to, and how they are set. Typically, you would choose the highest baud rate that both the device and the computer support, one stop bit, eight data bits, and no parity.

4. DATA FORMATS:
   Once the computer and device are communicating correctly, it is still possible to get incorrect data if the data is formatted by the device in one way and interpreted by the computer in another or vice versa. The easiest way to get things working is to instruct the device to send data in ASCII, with CR/LF terminating each data item. However, the fastest way to exchange data is to do so in binary.

After lot of discussion and thought the final parameters were decided, which are as follows:

1. Use 9 pin serial interface.
2. XON/XOFF Handshaking.
3. Com Params:
   Baud Rate : 9600
   Data Bits: 8
   Parity: 1
   Stop Bits: None
4. Since data in Cerberus is binary so binary data format is chosen.

NEW FEATURES ADDED TO THE GUI:

1. DATA EXPLORER:
   After discussion with the people who will be using the GUI, the need for a data explorer came to light. The data explorer should display all the calculated values in accessible way without creating a mess in the GUI.

To achieve this goal, I used QTableWidget of PyQt which is designed for the systematic data view. The following image shows the code for the data explorer.

```
def settable(self):
    rows = len(self.file_objects[self.tabarea.currentIndex()].calc.Time)
    self.datatable.setRowCount(rows)

    if self.see_all_channel == False:
        self.datatable.setColumnCount(2)
        xchannel = self.channellist[self.xaxisbox.currentIndex()]
        ychannel = self.channellist[self.yaxisbox.currentIndex()]
        self.updatedata()
        listc = []
        listc.append(xchannel)
        listc.append(ychannel)
        self.datatable.setHorizontalHeaderLabels(listc)
        for j in range(2):
            for i in range(rows):
                item = self.dataset[listc[j]][i]
                self.datatable.setItem(i,j,QTableWidgetItem(QString("%1").arg(item)))

    if self.see_all_channel == True:
        self.datatable.setColumnCount(15)
        self.datatable.setHorizontalHeaderLabels(self.channellist)
        for j in range(15):
            for i in range(rows):
                item = self.dataset[self.channellist[j]][i]
                self.datatable.setItem(i,j,QTableWidgetItem(QString("%1").arg(item)))
```

2. MULTIPLE FILES:
   The task of opening multiple files is achieved through QTabWidget of PyQt. Each tab has its associated Dataset object which contains all the data and other details.

3. SAVING A FILE:
   To save a file I used the standard python module named **Pickle,** which stores any object to a file. The object is stored serially as byte stream. To retrieve back the object, the module has a concept of unpickling which gets the object back.The following code shows how a file in saved (pickling):

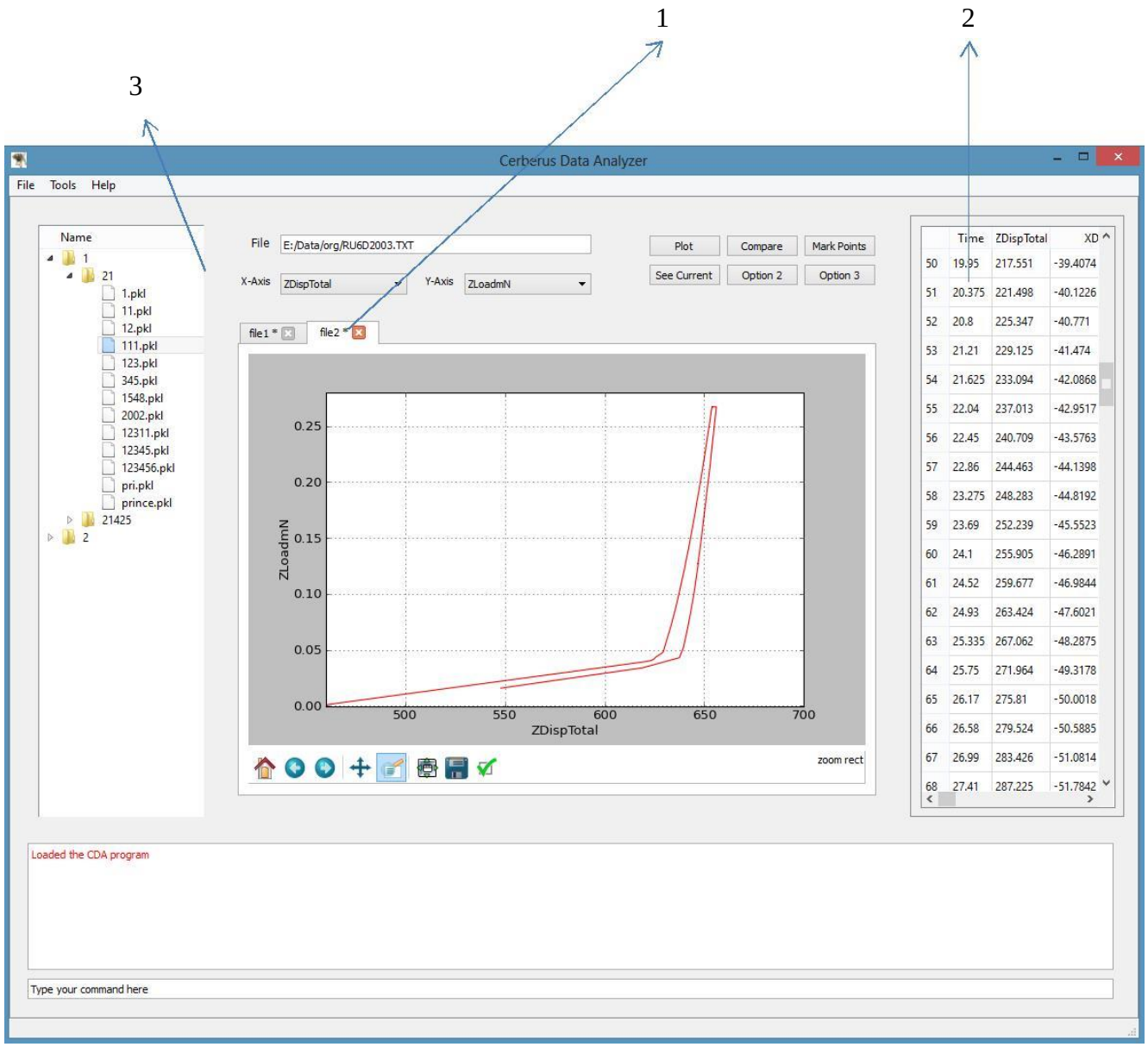
```
output = open("projects\\" + self.current_project + "\\" + self.current_exp + "\\"+ name , "wb")
pickle.dump(self.file_objects[self.tabarea.currentIndex()], output, -1)
output.close()
self.file_objects[self.tabarea.currentIndex()].filename = name

self.tabarea.setTabText(self.tabarea.currentIndex(), _translate("MainWindow", self.file_objects[self.tabarea.currentIndex()].filename, None))
self.file_objects[self.tabarea.currentIndex()].saved = True
```

The following code shows how a saved file is opened (unpickling):

```
indexItem = model.index(index.row(), 0, index.parent())
filePath = self.model.filePath(indexItem)
self.file_to_load = filePath
inputfile = open(filePath, "rb")
self.file_objects.append(pickle.load(inputfile))
```

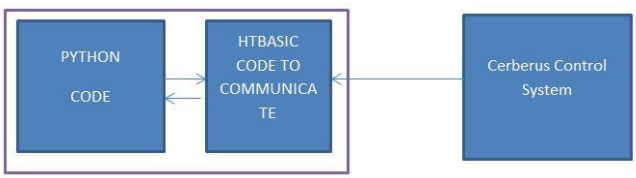
The final GUI is shown in following image:



1. Multiple tabs to open any number of files at the same time.
2. Data explorer to see the data of the file in the current tab.
3. Saved files in ".pkl" format.

CONNECTION:

The connection has been made using this model but it is in the final phase of testing. Final Model is:



A part of HTBasic code to communicate is shown in the following figure:

```
OPTION BASE 1
COM /Ind_data/Logged_data(20,5000) BUFFER,Ctrl_seq$(12,10,6)[10],Zero_displ,Data_point$(20)[30]
COM /Bsdm/T$[80],Vn$(50)[15],Sn$(20)[10],Sc(20),No,Nv,Ns,Tmp_seq$(20)[10]

CLEAR SCREEN                        !This test requires two machines connected by
PRINT "*** Transfer Test ****"      !serial cables.  One running this program, and
DIM A$[50]                          !the other outputing information to this one.

LOAD BIN "SERIAL"
ASSIGN @Buf TO BUFFER [2000]
ASSIGN @In TO 9

DISP "Waiting for data .... .. ... ... "

TRANSFER @In TO @Buf
WAIT FOR EOT @Io
ENTER @Buf;T$
DISP "T$ = ",T$

TRANSFER @In TO @Buf
WAIT FOR EOT @Io
ENTER @Buf;No
DISP "No = ",No

TRANSFER @In TO @Buf
WAIT FOR EOT @Io
ENTER @Buf;Nv
DISP "Nv = ",Nv

TRANSFER @In TO @Buf
WAIT FOR EOT @Io
ENTER @Buf;Vn$(*)
DISP "Vn$ = ",Vn$(*)

TRANSFER @In TO @Buf
WAIT FOR EOT @Io
ENTER @Buf;Ns
DISP "Ns = ",Ns

TRANSFER @In TO @Buf
WAIT FOR EOT @Io
ENTER @Buf;Sn$(*)
DISP "Sn$ = ",Sn$(*)

TRANSFER @In TO @Buf
WAIT FOR EOT @Io
ENTER @Buf;Sc(*)
DISP "Sc = ",Sc(*)

DISP "Calculating headers ... "
IF Ns<20 THEN
    Sc(Ns+1)=No+1
END IF

ALLOCATE T_data(Nv,No)

REDIM Logged_data(Nv,No)

MAT Logged_data=(0)

ENTER @Cfile;T_data(*)
```

CONCLUSION:

The next and the last target is to test this model for different types of tests and create a help manual.