# A DATA ANALYZING SOFTWARE TOOL FOR THE OMNIDIRECTIONAL RFI SPECTRUM MONITORING SYSTEM OF GMRT

SHUBHENDU JOARDAR
&
ARIJIT BANERJEE

GMRT Khodad
Tata Institute of Fundamental Research
Narayangaon, Pune - 410504

# PREFACE

There was a lot of demand for a tool to analyze the data from the RFI monitoring system of GMRT which could make the 3 dimensional plots, contours, percentage RFI occupancy plots and directional averaged spectrum plots. I had earlier made these as on a scripting language (MATLAB) but because of restricted user license, this tool was not accessible to the normal users. Therefore it was felt that there is a necessity of the software to be MATLAB independent. The present work has been initiated with this in mind. The software analyzer tool has been made independent of MATLAB. The first version of the software has been made for Windows 2000/XP/etc. and is released. The next step would be to convert the tool for a UNIX/LINUX operating system. New features shall be added with the growing demand and feed-backs from the users. I am thankful to Sri Arijit Banerjee, third year B.Tech. (Electronics and Communication) student of Kalyani Government Engineering College who has also put a tremendous effort in completing this work to its final details.

Shubhendu Joardar

# Acknowledgements

*I am thankful to Mr. Shubhendu Joardar for his guidance encouragement and involvement throughout the work. I am also thankful to the Director Professor Rajaram Nityananda and staff members of GMRT.*

*Arijit Banerjee*

# CONTENTS

# Chapter 1

# OVERVIEW OF THE OMNI DIRECTIONAL RFI MONITORING SYSTEM FOR GMRT

## Introduction:

All terrestrial radio sources are categorized as RFI from the radio astronomical point of view. These could be narrow band type like communication signals or wide band type like arc-welding, automobile ignition, high voltage power lines etc. Terrestrial radio spectrum monitoring assists radio astronomy by providing information like free cosmic frequency bands, out-of-band and spurious emissions form transmitters, power line interference and the growth of new transmitters in the region. It has other applications like studying radio propagations and radio direction finding. Magnitude spectrum of radio signals available from a region is useful for generating radio availability statistics and studying RFI properties over time, etc. The RFI monitoring system of GMRT [1], [2] is a magnitude spectrum monitoring system but posses the capability of radio direction finding [3] [4] using software. The system has been developed for assisting radio astronomy.

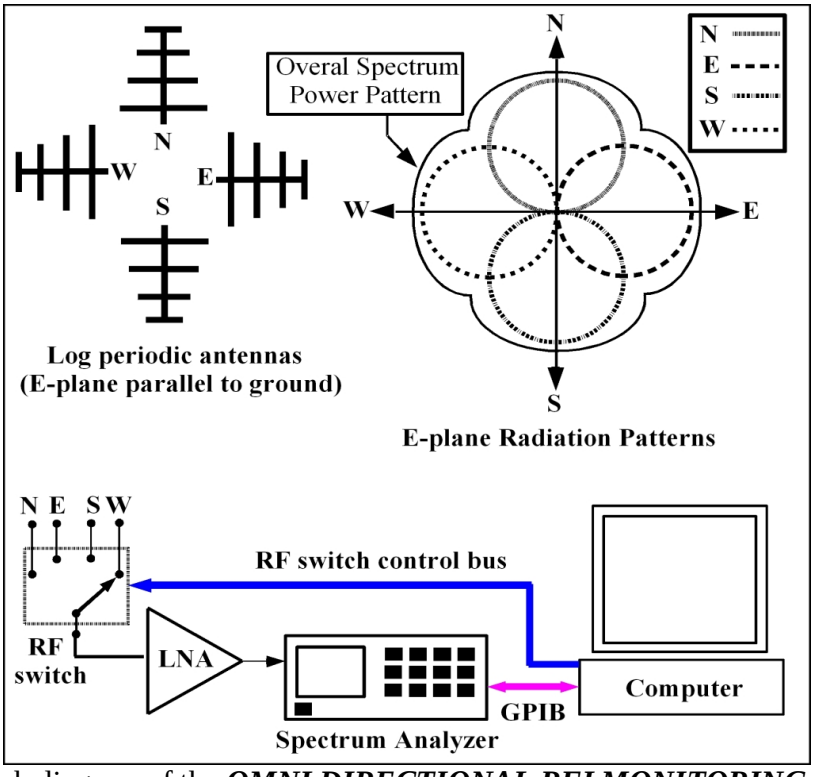



Fig.1: Block diagram of the ***OMNI DIRECTIONAL RFI MONITORING SYSTEM.***

## System Hardware:

Fig.1 shows the top view of the antenna mountings on a tower and the block details of the omni-directional RFI monitoring system of GMRT [1] [2]. It consists of four identical log periodic dipole arrays with their maximum directivity pointed to east, west, north and south directions, and are mounted over a tower of twenty meters height. The antennas are positioned in the E-plane parallel to the ground. The computer selects the antennas in a cyclic order, viz. East, West, North and South. The corresponding multiplexed RF powers from the antennas are amplified suitably and fed to the spectrum analyzer kept inside the lab building through an RF-cable. The output from the spectrum analyzer is displayed and saved by the computer, which is a 401-point data with time stamp for individual directions. Presently, the frequency range of the system is from 70 MHz to 1500 MHz. Settings of the spectrum analyzer and observation period are user defined and taken care by the computer. The computer continuously cycles the antennas over the entire period of observation and records the data in different files corresponding to the antenna direction.

## The new software tool:

The data from the four antennas are recorded in four individual files corresponding to the antenna directions, viz. East, West, North and South. Additionally there is a separated single file containing the data from all directions in a cyclic order. All the files begins with a header containing information of the spectrum analyzer settings, viz. ***center frequency***, ***span***, ***RBW***, ***VBW***, ***attenuation***, ***amplitude scale***, ***number of averages***, etc.The data files are written in ASCII and can be edited with any text editor if necessary. The instantaneous time of the data is recorded first followed by the 401 points of data. In the ****AVG.rfi*** file the data is written sequentially as East, West, North, South, East, West …. so on. This is a file containing the complete data from all the four directions. In order to work with the data, the header is extracted first and then the data is related with the header. In the present work we have made a software tool to analyze the data content of the ****AVG.rfi*** files and thereby saving in various image formats. Provision for browsing the plots with a mouse cursor is also added for the advantage of the users. The tool has been named as the ***Omni directional RFI Monitoring System Data Analyzer*** (**ORMSDA**).

## References:

[1] Shubhendu Joardar, “The Omnidirectional RFI monitoring system of GMRT-TIFR”, Summer School on Spectrum Management, June 2002, NRAO, West Virginia, USA.
[2] Shubhendu Joardar, Lecture presentation “The E-plane Omnidirectional RFI monitoring system for GMRT”, NCRA Engg. Committee-2003 Intenal report, GMRT, Pune.
[3] Shubhendu Joardar, “Radio Direction finding using 'N' switchable antennas and RFI estimation”, INC-URSI, November 2003, NPL, New Delhi.
[4] Shubhendu Joardar, “RFI monitoring system of GMRT and radio interference analysis on various radio-astronomy bands”, URSI, Oct. 2005, NPL New Delhi (paper accepted for publication on May 2005).

# Chapter 2

## DESCRIPTION OF THE OMNI DIRECTIONAL RFI MONITORING SYSTEM DATA ANALYZER WITH BLOCK DIAGRAM & FLOWCHART

We will discuss on the following points in this chapter:-

1)GUI Concepts.
2)System Calls.
3)Callback Objects.
4)Push-button Callback Objects.
5)Menu-item Callback Objects.
6)Dialog box.
7)Message Dialog box.
8)Wait bar Dialog box.
9)Input Dialog box.
10) File-open Dialog box.
11) Exit Dialog box.
12) Figure Dialog box.
13) Input-text Callback Objects.
14) Output-text Callback Objects.
15) Static-text Objects.
16) Exit Callback Objects.
17) Main Form.
18) Flowchart of the Operation of the ***ORMSDA***.
19) Description of the Flowchart of Operation.

# GUI Concepts:

The word ***GUI*** is the abbreviation of 'Graphical User Interface'. It signifies that the user can have an access to the interface of computational task graphically. We usually write programs to execute in 'Command Mode', which is difficult to a new user as well the processing speed is slow enough along with 'Single Tasking' capabilities are present, that means one task in one time. But ***GUI*** provides the scope of graphical interface, easy & friendly to use environment, 'Multi Tasking' capabilities, that means many a task at a time, 'Multi Threading' capabilities, that means multiple flow of control, high speed of performance. In general ***GUI*** is provided on 'Multi Tasking' operating systems like 'Microsoft Windows 2000', 'Microsoft Windows XP', 'Microsoft Windows 2003 Server', 'Red hat Linux', 'Linux Fedora', 'Linux Mandrake', 'Linux Suse', 'Linux Ubantu', 'Mac Os' etc. There exist usually no pointing devices in command mode of operation. ***GUI*** provides 'Mouse' as the pointing device. Here in ***GUI***, maximum work is accomplished by mouse other than inputting a text. ***GUI*** provides the components like buttons, menu bar, task bar, dialog boxes, popup windows, balloons, check boxes, sliders, radio buttons, list boxes, toggle buttons, frames, editable text, static text etc.

In general ***GUI*** works on an event-based model. This means we don't just go on executing codes after codes, the same way we do in procedural programming. Our program waits, in an idle state, until the user specifically gives an input message to the program. The message may be given by clicking a push button or entering a text in an editable field etc. Hence the message is a certain event. Our program then acts on the event by executing whatever code we have associated with it.

Our ***GUI***, ***ORMSDA*** is an Object oriented approach along with event-based model. Objects are everywhere in ***ORMSDA***, from a simple button to complex objects like callback objects. Objects can be meant by separate routines that have a unique identification number to the main program. Generally it is defined as: - ***"An object is an identifiable region of memory that can hold a fixed or variable values or set"***. What ever it may be, the unique identification number is called the handle of that object. Whenever the main program calls an object, it calls by handle. Objects have some features like Inheritance, Polymorphism etc. Objects can be inherited to form new objects. Any object oriented programming language like C++, Java etc provides the scope of creating ***GUI***'s. Our program, ***ORMSDA*** is a fully functional ***GUI*** created with combined C & C++ codes generated by MATLAB. Basically it contents a main 'Form' and many a child object. Hence ***ORMSDA*** is also can be regarded as ***Parent-Child*** type of model. If the main form is considered to be the ***Parent*** object then menu bars, buttons, editable texts, figures etc sub-items are considered to be as ***Child*** objects. It is a multi-tasking, multi-threaded ***GUI***. It is user friendly, fast enough to process megabytes of data in minutes. Most of all, the ***GUI*** does not go hang if some problems occur in input data files i.e. corrupted data files.

Screen shot of ***ORMSDA*** is given in Fig.2 below.

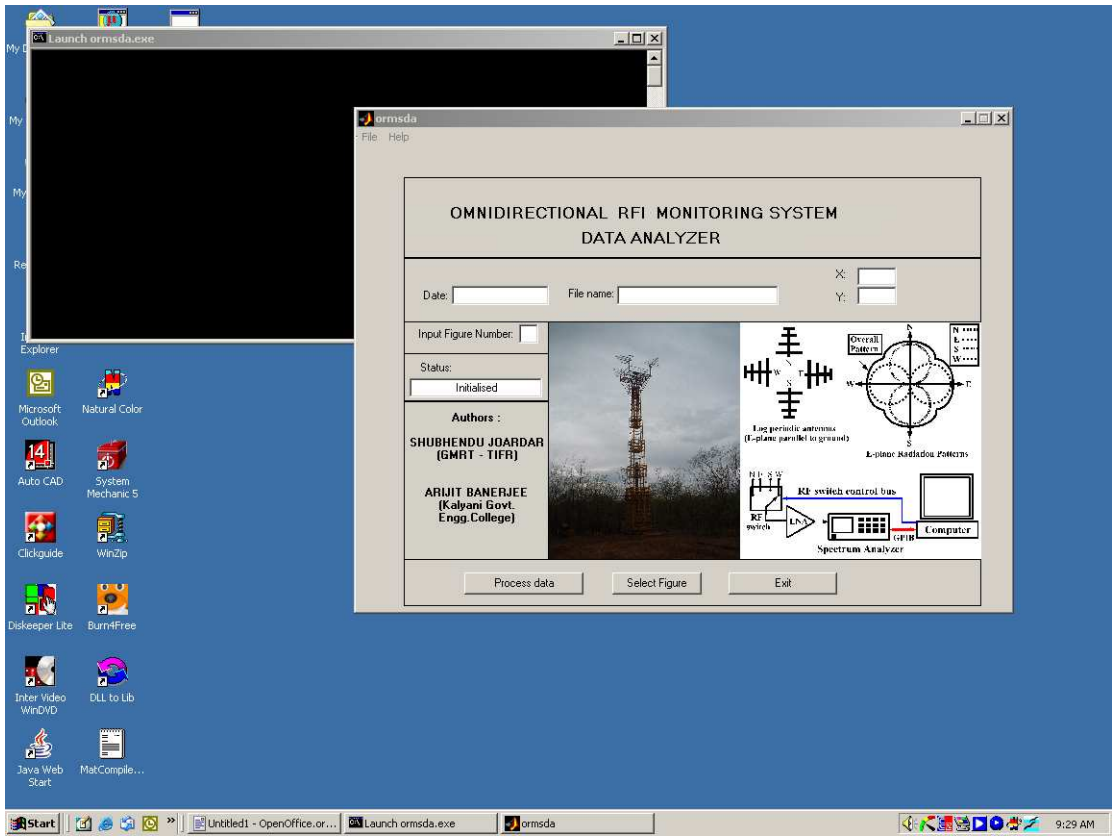



Fig.2: Screen shot of ***GUI of ORMSDA.***

Here we can see that as a ***GUI***, ***ORMSDA*** has main application window named ***ormsda***.exe. It has a menu bar with ***File*** & ***Help*** menu objects. The text ***OMNI DIRECTIONAL RFI MONITORING SYSTEM DATA ANALYZER*** is a static text object, i.e. it can not be modified or changed during runtime execution by writing codes. The ***Date***, ***File name***, ***X***, ***Y***, ***Input Figure Number*** and ***Status*** are the dynamic text objects i.e. these texts can be edited either by key board input or by writing codes. The ***Authors :..*** is again a static text object. One of the pictures, we see, is a tower containing four LPDA which is a picture object and the block diagram is also the same. Below we can have a vision of three buttons. One of them, named ***Process data*** is a push button object. The other two buttons named ***Select Figure*** & ***Exit*** are also push button objects. In the menu-bar, if we glance at, we can find a submenu child object named ***Open*** which is under the parent ***File*** object. Another submenu under ***File*** object is ***Exit*** object. On the other hand, in the ***Help*** menu, an object is there. ***Help*** is a parent object to the ***Documentation*** submenu object. The ***About Authors*** submenu child object is also under the parent ***Help*** menu object. Many a hidden object, other than ***GUI*** objects, is in the ***ORMSDA*** which has not a graphical feature, like callback objects. Most of all we say that ***ORMSDA*** application ***Form*** is the main parent object in the entire program which has the described ***Child*** objects or ***Child*** processes or events.

# 2) System Calls:

In figure 2, we can have a glance that there exists another command window named ***Launch ormsda.exe***. This command window has a great importance in the whole ***GUI***. In General, as far as the term ***System Call*** is considered, different operating system has different system calls for a same operation to be accomplished. Thus if we write different codes for doing the same jobs in different operating systems, which are equivalent to system calls, it will be a laborious as well as a huge coding along with low performance of executables and slow speed of operation. Hence we have to find an alternative solution. The solution is system call. The command window is linked with the application ***ORMSDA*** in such a manner that whenever there is a requirement of indirect system call, the operating system should provide the necessary facility to the application program ***ORMSDA***. Here we have 'Windows 2000' and we find that the 'DOS' prompt to be there for service. If we make the application ***ORMSDA*** for 'LINUX' or 'Mac OS' their own command prompt window should be there. Thus if there is an error, the command window will definitely show it. Even if the error, in such a case be not defined in the application code, will be showed in the command window. Thus the command window associated with ***ORMSDA*** is like a debugger which will trace the bug there in the execution of the ***ORMSDA***.

So far, the only thing we've done was to use well defined kernel mechanisms to register `/proc` files and device handlers in Linux. This is fine if we want to do something, the kernel programmers thought we have want, such as write a device driver. But what if we want to do something unusual, to change the behavior of the system in some way? Then, we are mostly on our own. The *real* process to kernel communication mechanism, the one used by all processes, is system calls. When a process requests a service from the kernel (such as opening a file, forking to a new process, or requesting more memory), this is the mechanism used. If we want to change the behaviour of the kernel in interesting ways, this is the place to do it. By the way, if we want to see which system calls a program uses, run **strace <arguments>**.

In general, a process is not supposed to be able to access the kernel. It can't access kernel memory and it can not call kernel functions. The hardware of the CPU enforces this (that is the reason why it is called `protected mode').
System calls are an exception to this general rule. What happens is that the process fills the registers with the appropriate values and then calls a special instruction which jumps to a previously defined location in the kernel (of course, that location is readable by user processes, it is not writable by them). Under Intel CPUs, this is done by means of interrupt 0x80. The hardware knows that once we jump to this location, we are no longer running in restricted user mode, but as the operating system kernel is there we are allowed to do whatever we want.

The location in the kernel a process can jump to is called *system_call*. The procedure at that location checks the system call number, which tells the kernel what service the process requested. Then, it looks at the table of system calls (`sys_call_table`) to see the address of the kernel function to call. Then it calls the function, and after it returns, does a few system checks and then return back to the process (or to a different process, if

the process time ran out). If we want to read this code, it's at the source file **`arch/$<$architecture$>$/kernel/entry.`** after the line `ENTRY (system_call)`.

So, if we want to change the way a certain system call works, what we need to do is to write our own function to implement it (usually by adding a bit of our own code, and then calling the original function) and then change the pointer at `sys_call_table` to point to our function. Because we might be removed later and we don't want to leave the system in an unstable state, it's important for `cleanup_module` to restore the table to its original state. This all about 'Linux' or 'Unix' system calls.

Now if we consider about 'Windows' system calls, we can have the **DOS3Call** API function in Windows 3.*x* must be called from assembly language. It is typically used to perform file I/O. In Win32, we should replace assembly language code that calls **DOS3Call** with the appropriate Win32 file I/O calls. Other (non-file) INT 21H functions should be replaced with the portable Windows API call as shown in the following table.

| **INT 21H subfunction** | **MS-DOS operation** | **Win32 API equivalent** |
|---|---|---|
| 0EH | Select Disk | **SetCurrentDirectory** |
| 19H | Get Current Disk | **GetCurrentDirectory** |
| 2AH | Get Date | **GetDateAndTime** |
| 2BH | Set Date | **SetDateAndTime** |
| 2CH | Get Time | **GetDateAndTime** |
| 2DH | Set Time | **SetDateAndTime** |
| 36H | Get Disk Free Space | **GetDiskFreeSpace** |
| 39H | Create Directory | **CreateDirectory** |
| 3AH | Remove Directory | **RemoveDirectory** |
| 3BH | Set Current Directory | **SetCurrentDirectory** |
| 3CH | Create Handle | **CreateFile** |
| 3DH | Open Handle | **CreateFile** |
| 3EH | Close Handle | **CloseHandle** |
| 3FH | Read Handle | **ReadFile** |
| 40H | Write Handle | **WriteFile** |

| | | |
|---|---|---|
| 41H | Delete File | **DeleteFile** |
| 42H | Move File Pointer | **SetFilePointer** |
| 43H | Get File Attributes | **GetAttributesFile** |
| 43H | Set File Attributes | **SetAttributesFile** |
| 47H | Get Current Directory | **GetCurrentDirectory** |
| 4EH | Find First File | **FindFirstFile** |
| 4FH | Find Next File | **FindNextFile** |
| 56H | Change Directory Entry | **MoveFile** |
| 57H | Get Date/Time of File | **GetDateAndTimeFile** |
| 57H | Set Date/Time of File | **SetDataAndTimeFile** |
| 59H | Get Extended Error | **GetLastError** |
| 5AH | Create Unique File | **GetTempFileName** |
| 5BH | Create New File | **CreateFile** |
| 5CH | Lock | **LockFile** |
| 5CH | Unlock | **UnlockFile** |
| 67H | Set Handle Count | **SetHandleCount** |

These are all about 'Windows' system calls. As we mention we are using indirect system calls in ***ORMSDA*** i.e. not calling the Kernel's processes directly but with the help of command line interpreter. In the case of 'Windows', the command line interpreter is 'command.com', in 'Linux' it is terminal, what ever it may be, we are using system calls through the system commands, interpreted by command line interpreter or terminal. A block-diagram of indirect 'System Calls' implemented in ***ORMSDA*** is shown in Fig.3.

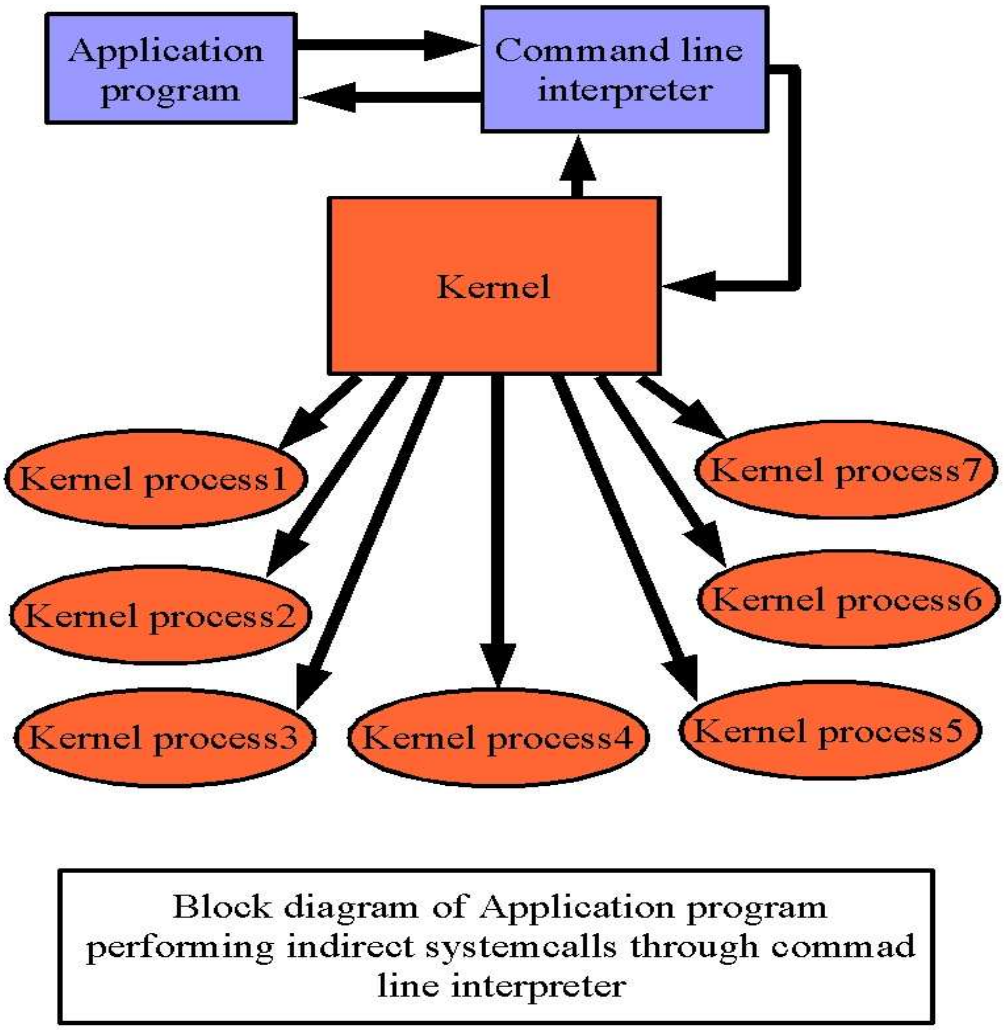



Fig.3: Block diagram of ***ORMSDA application program's performing system calls.***

# 3) Callback Objects:

***Callback*** is some thing like: - execution of some event after the happening of another event. We know about objects any how; hence ***Callback Objects*** are one type of object those can be called by the end time of execution of an event object. Suppose we have a ***GUI*** button. If the button has an embedded object associated with it, it can call the object after being clicked by mouse or being rolled over by mouse or by some other possible events to be considered. Simply we can embed any kind of embedded object into another object. Let us have the view of ***ORMSDA***'s ***Callback Objects*** in Fig.4.

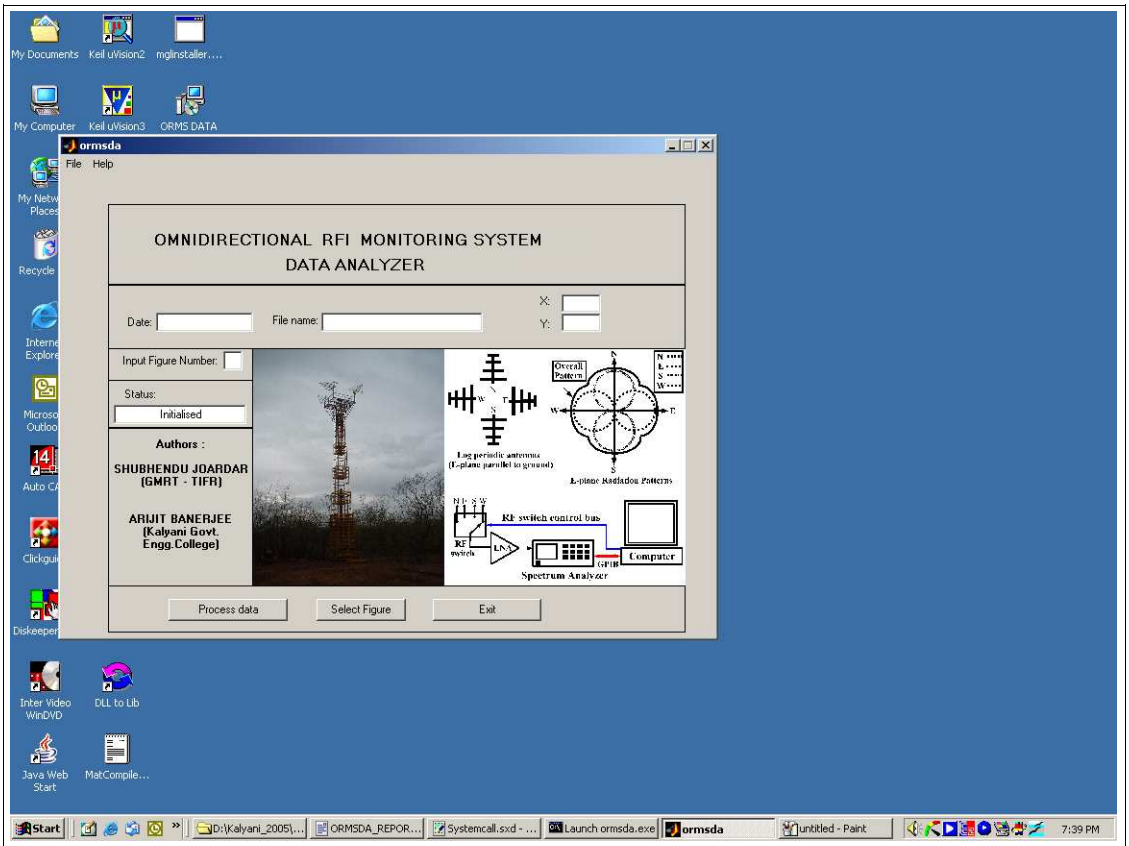



Fig.4 : Screen shot of ***Callback objects of ORMSDA***

Here we can see some of the callback objects, like push buttons, menu items etc. Now if we click the button named ***Process data,*** a dialog box appears. This is because the ***Process data*** button has an embedded 'Callback Object' to call the dialog box object. Whenever the push button is clicked the embedded 'Callback Object' is executed to create the dialog box objects. The ***Select Figure*** button has a complex callback object. ***Select Figure*** button callback object has a link with ***Input Figure Number*** callback object. Whenever a number which is greater than equal to 1 as well as less than equal to 4 is given as a text input to the embedded text input object and the ***Select Figure*** callback object is executed, it calls the figure object specified by the figure handle number and the specific figure object is displayed in a figure window. Now we can have a glance in the figure 3 that there is a push button named ***Exit***. Here the ***Exit*** push button also has an embedded callback object which executes, after a mouse click is found to be there on the ***Exit*** button. Something is special about the ***Exit*** push button callback object. It is due to that reason that the ***Exit*** push button has an embedded callback object to destruct all objects or may be considered to be as an object which deletes all objects which is of course the ***main form*** object and the child objects of the main form also. But the figure objects are out of reach of the ***Exit*** callback object. They are separate objects or not child objects of the main form. Whatever it may be, the menu bar has objects like ***File*** menu object and ***Help*** menu object. ***File*** menu object has two callback child objects which are designed by us. One of the ***File*** menu child object is ***Open*** submenu object which has also an embedded callback object to open files having ****AVG.rfi*** extensions from user

specified directories. In the ***File*** menu object, there is also an embedded callback object named ***Exit*** which is a submenu object of ***File*** menu object. Telling about ***Help*** menu object, we can say that it has two child submenu objects. One of the ***Help*** menu object is ***Documentation*** submenu object which has an embedded callback object linked with ***Adobe's Acrobat Reader*** software that invokes the documentation files to be loaded. On the other hand, the second submenu object of ***Help*** menu object is ***About Authors*** submenu object which has also a callback embedded object to a dialog box displaying the name of the authors. This is all about callback objects in ***ORMSDA***. A block diagram of how callback objects are executed is shown in Fig.5 below:-

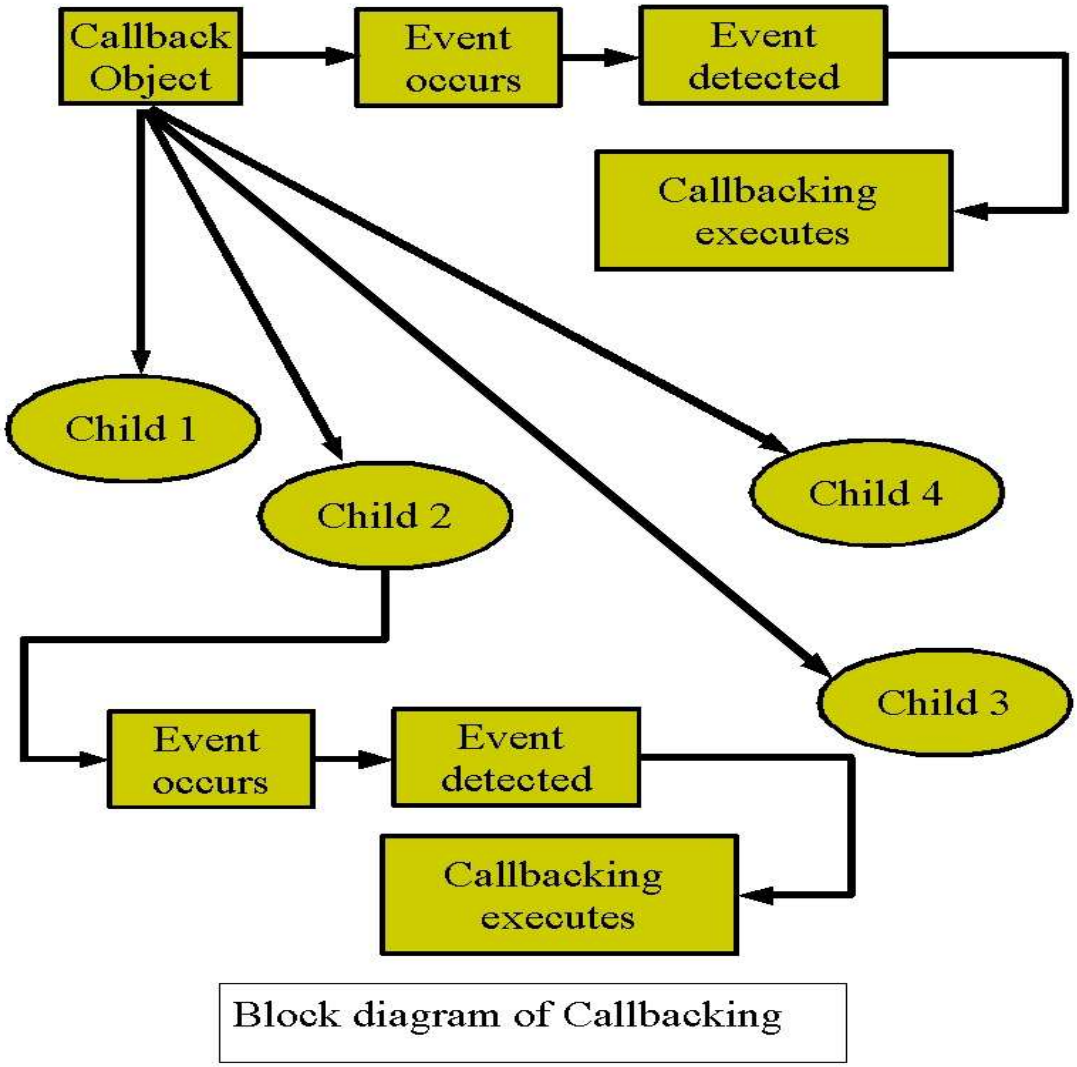



Fig.5 : Block diagram of ***ORMSDA's Call backing***

# 4) Push-button Callback Objects:

Push buttons are such a kind of object which can serve many a subroutines in a single callback. Push button objects have a graphical embedded object which has the look of the button. We can change the embedded graphics object of a button by some extent, as per the power of the programming language. In general the embedded graphical object has two states. One is the normal view and another is the view after a button is pressed or the object being executed. The normal view is the three dimensional general look of the button and the exceptional look is the look after the button is pressed with dotted rectangular box inside the button. We can have the screen shot of a push button named ***Process data*** in Fig.6. The ***Process data*** push button object has an embedded object that opens the files, i.e. ****AVG.rfi*** files, for processing. Here the call backing is implemented by linking a ***file-open dialog box*** to the push button object named ***Process data***. Whenever the button is clicked, the callback object executes and the ***file-open dialog*** object, linked with it, is also triggered.

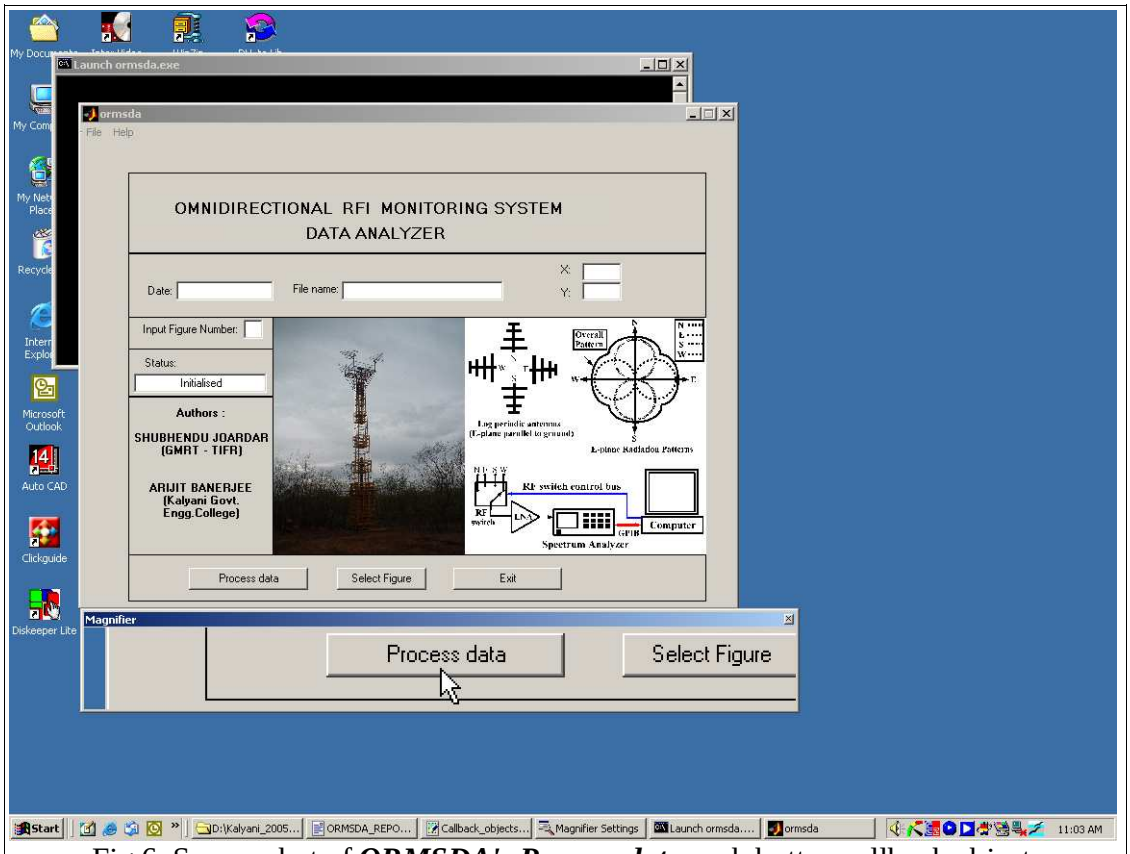


Fig.6: Screen shot of ***ORMSDA's Process data*** push button callback object.

Next we have another push button callback object named ***Select Figure***. See Fig.7. The specified push button object has an embedded graphical object which has a three dimensional looks also. The button has two state of operation as stated on the previous

case. But the call backing object is the most complex one and completely different from that of the previous one. To have a look of the call backing of this object let us deeply analyze it. The callback object of the push button named ***Select Figure*** is not triggered at the 'Initialize' state of 'Status', because there is a filter object which search for whether the data is processed. If any how, the data is processed, the filter object gives access to the other call backing objects to be executed. But, in case of the data being not processed, the filter object hinders all callback objects under the ***Select Figure*** push button object and sends control to the error handler object. Error handler object is such a kind of object which maintains whenever an error occurs in the application. In this case the error handler object passes a message to a warning dialog object and the warning dialog object is executed and displayed. The most complex type of callback occurs in the case of status being 'Processing Completed'. In this case, if we trigger the callback object with proper values for operation, the callback object grasps the handle of the specified figure and the axis handle is also transferred to the callback object. The axis handle of the figure object is linked with the scaling of the axis. Hence we can easily have the control over the x and y coordinates of the axis. The x and y coordinate values associated with a mouse click object, is passed to the ***X*** and ***Y*** fields of ***ORMSDA***.

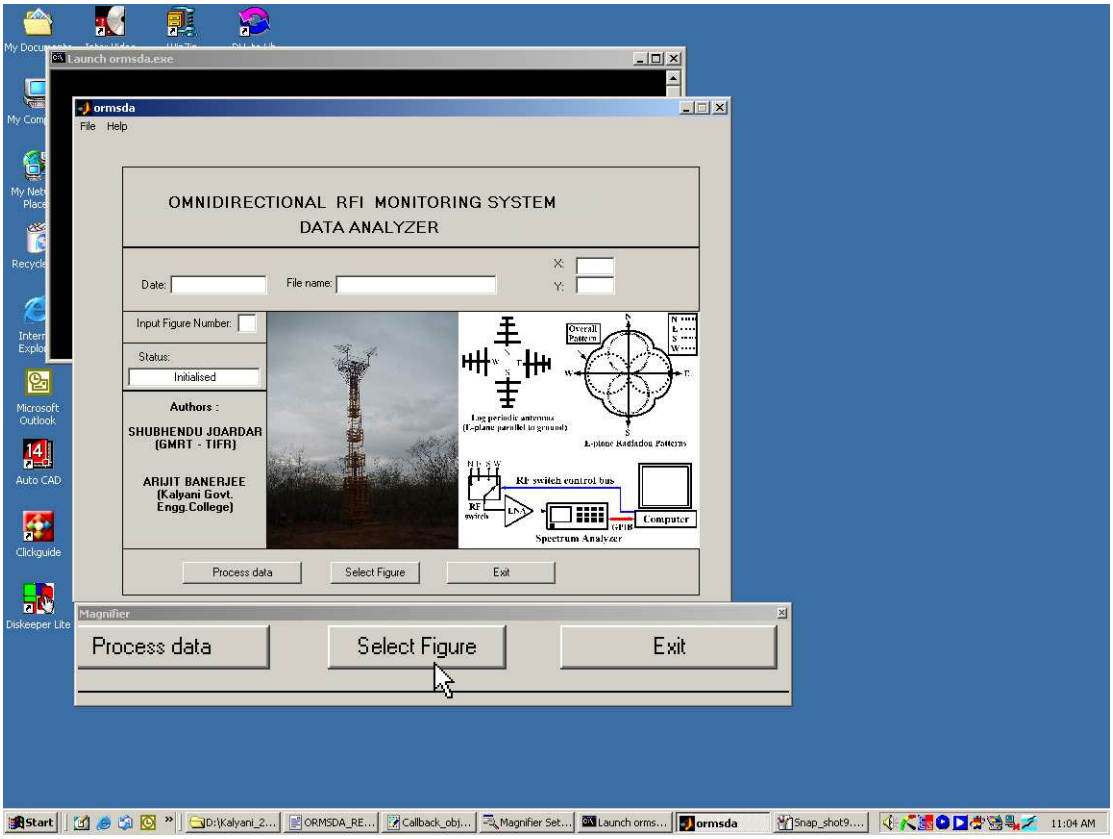


Fig.7: Screen shot of ***ORMSDA's Select Figure*** push button callback object.

The ***Exit*** button object has the other features of a push button object also. It has an embedded graphical object which is a bitmap, looks like a three dimensional figure. The

***Exit*** push button has two states, one is general state and other is pressed state which has a look of dotted rectangle within the main bitmap. The push button has an embedded static type text object. See Fig.8. We give the name ***Exit*** in the same object to display the name in the button. In the screen shot below, we have the highlighted push button object named ***Exit***. As the name suggests, it is an object that perform the operation of quitting the main application. The push button has a callback object which indirectly has a control over the main application window object. The callback object of the push button has an embedded object which calls a dialog box object. The dialog box object has two push button objects. One is for quitting the application named ***Yes*** and another is ***No*** for canceling the dialog box object. If ***No*** is clicked, the push button has an inbuilt call backing object which destroys the dialog box from the memory. On the case that ***Yes*** is clicked, the call backing object gets the handle of the main application window and executes a destructive operation to unload the application from memory. Hence the main window goes quitted. Here we see that the ***Exit*** push button don't have the direct access to the deletion of the main application object from the memory, but it executes another call backing to unload the main application. Here it seems to be the example of a callback object which is again under a callback object. We can associate many a callback object under a single callback object or something like that.

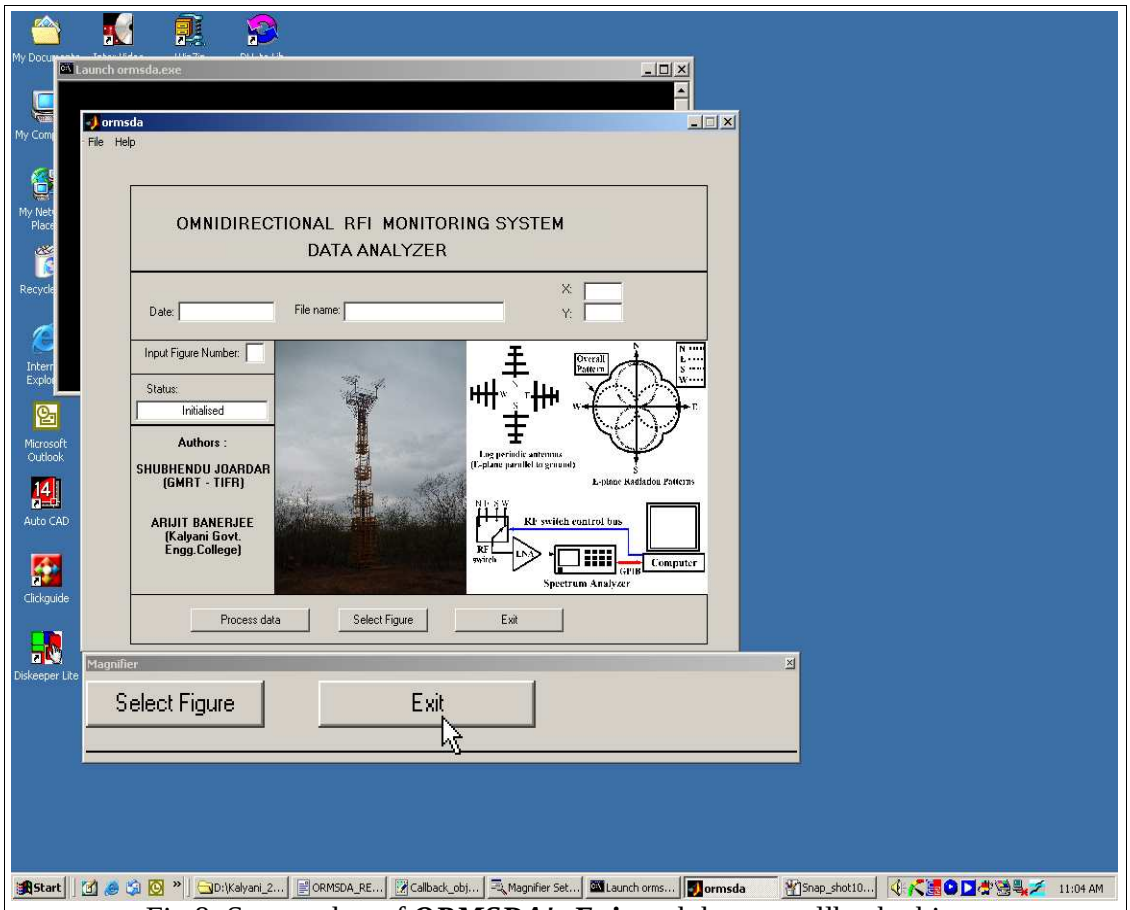


Fig.8: Screen shot of ***ORMSDA's Exit*** push button callback object.

# 5) Menu-item Callback Objects:

Menu items are the pull down type objects i.e. on a mouse click it will opens up in the downwards direction. They have an embedded graphical bitmap object. There is also a static text object associate with it. The name of the menu is displayed in the static text object field. Whenever a mouse roll-over is detected, the static text is highlighted with a blue, in general, patch of rectangle. Below, the magnifier window highlights the ***Open*** sub menu object in Fig.9. The word suggests the obvious operation of the sub menu. It is to open a ****AVG.rfi*** file from a user specified directory. But the sub menu has not had the direct control over opening a file. It has an embedded callback object that creates an open-file type dialog box. With the help of the open-file dialog box user can specify the desired file to be loaded in the memory. The file-open dialog has two embedded call backing object which are ***Open*** and ***Cancel*** named push buttons. On clicking the ***Cancel*** push button the file-open dialog object get destroyed. But after selecting the specific file, if the user clicks on the ***Open*** button, it callbacks and executes another object which grasps the path specified and load the file in memory.

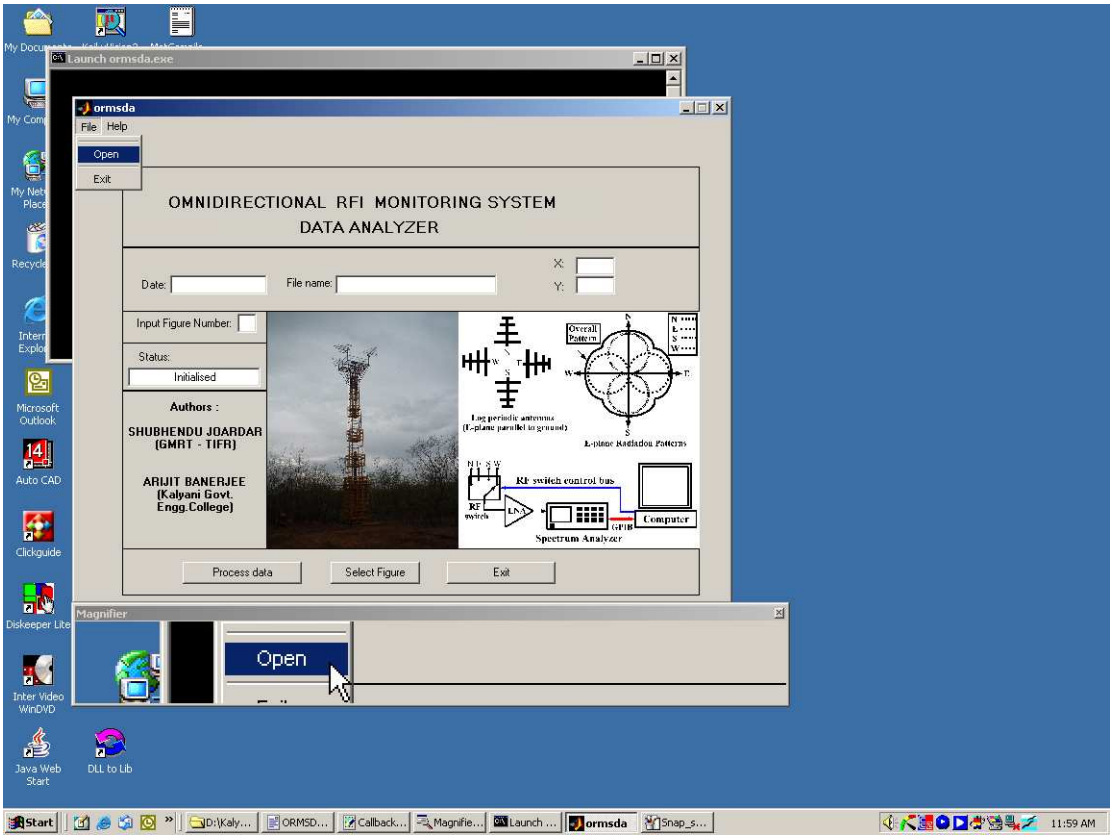


Fig.9: Screen shot of ***ORMSDA's Open*** sub menu callback object.

Here the screen shot of the ***File*** menu's sub menu ***Exit*** is given in Fig.10. The ***Exit*** sub menu is used for quitting the main application or to unload the main application from the

memory. It has a static type of text embedded object which shows the name of the sub menu. Whenever a mouse roll-over is detected, the static text is highlighted with a blue, in general, patch of rectangle. The sub menu has a callback object which indirectly has a control over the main application window object. The callback object of the sub menu has an embedded object which calls a dialog box object. The dialog box object has two push button objects. One is for quitting the application named ***Yes*** and another is ***No*** for canceling the dialog box object. If ***No*** is clicked, the push button has an inbuilt call backing object which destroy the dialog box from the memory. On the case that ***Yes*** is clicked, the call backing object gets the handle of the main application window and executes a destructive operation to unload the application from memory. Hence the main window goes quitted. Here we see that the ***Exit*** sub menu don't have the direct access to the deletion of the main application object from the memory, but it executes another call backing to unload the main application. Here it seems to be the example of a callback object which is again under a callback object. We can associate many a callback object under a single callback object or something like that.

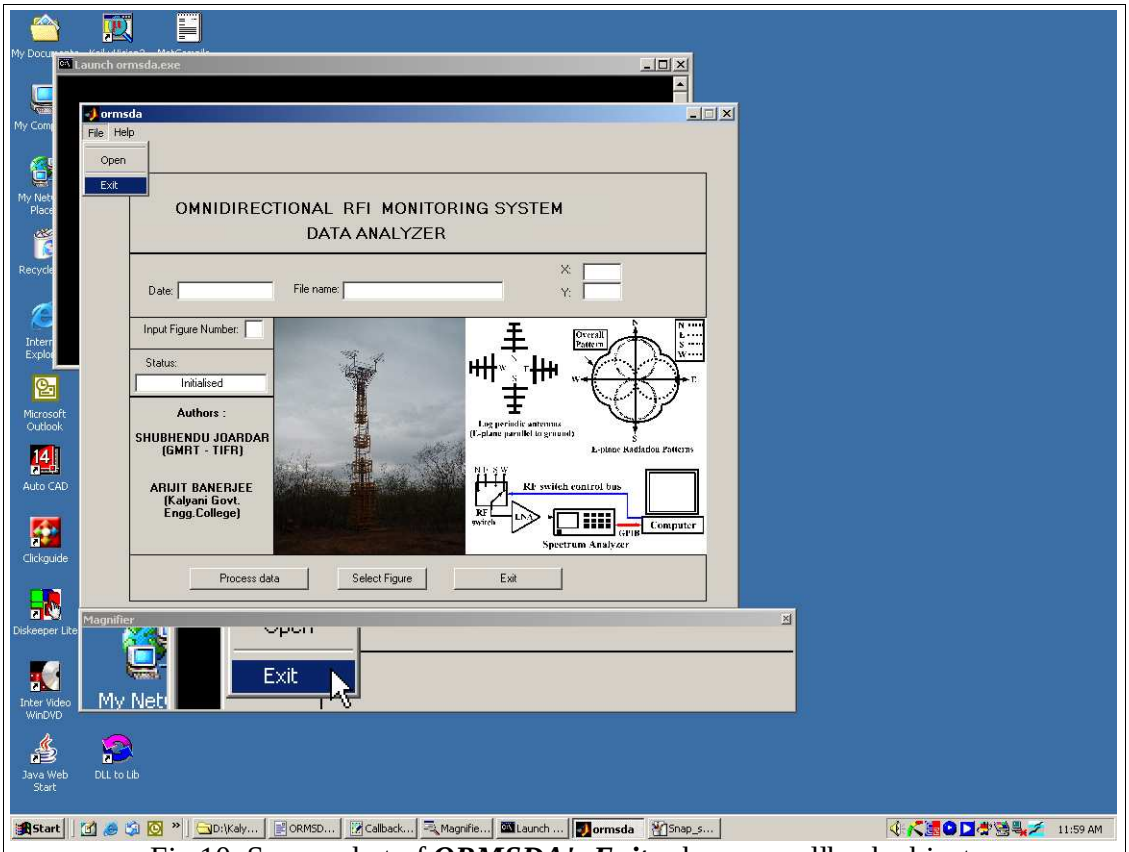


Fig.10: Screen shot of ***ORMSDA's Exit*** sub menu callback object.

Here we have the ***Documentation*** sub menu in Fig.11. This is completely different from all sub menus. The reason is it has a callback object that performs indirect ***system-call***. The embedded object has an indirect linkage with another application named ***Adobe's Acrobat Reader*** through the command window. As the documentation is written in PDF

file format, we here use the Acrobat Reader to access the documentation. Here one thing should be notified that this call backing object is completely different from others, because the call backing object performs another application to be loaded through system call.

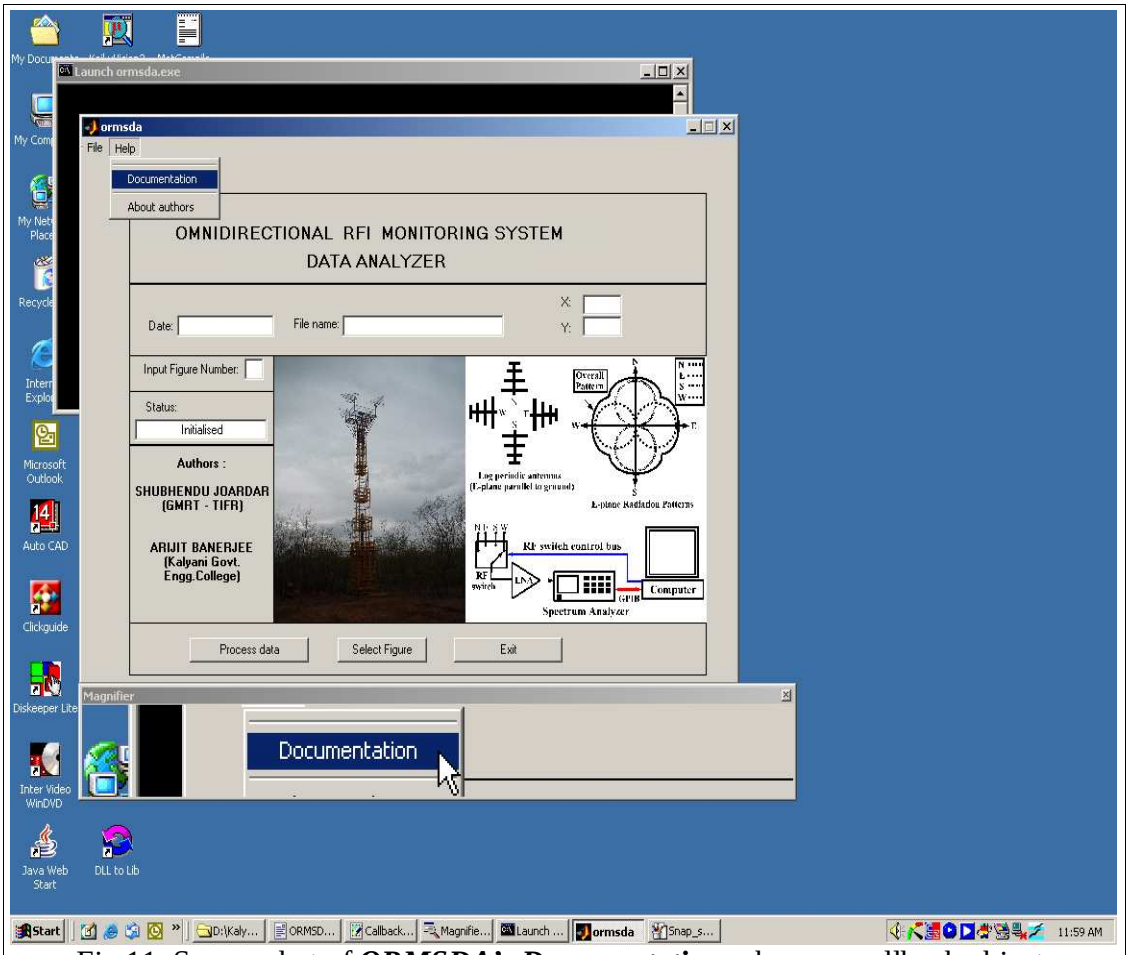


Fig.11: Screen shot of ***ORMSDA's Documentation*** sub menu callback object.

The screen shot shows the highlighted object which is a sub menu named ***About Authors***. See Fig.12. As the name suggests, it displays the name of authors along with some other information. It has an embedded graphical bitmap object along with a static type text object. Whenever the mouse is rolled over the sub menu, it is highlighted by an inbuilt object causing a blue rectangular patch. The sub menu has a callback object, when it is executed; it creates a message dialog box. The message dialog box has a child push button object named ***OK***. The message dialog object carries the authors' information and shows the same. On clicking the ***OK*** button the message dialog box unloads itself from the memory.

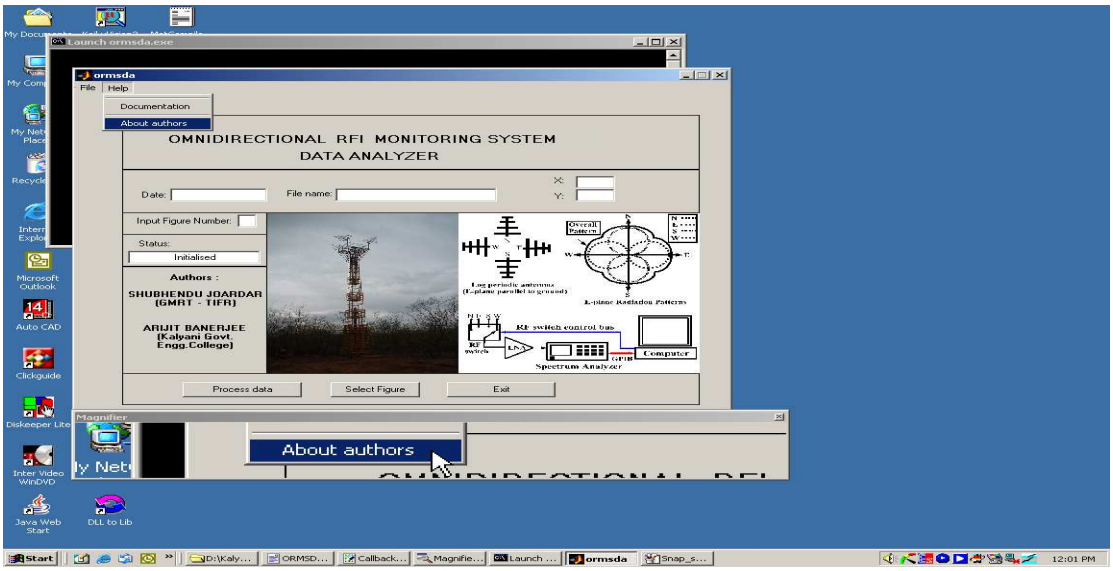


Fig.12: Screen shot of ***ORMSDA's About authors*** sub menu callback object.

# 6) Dialog-box:

Dialog boxes are considered to be the most exciting features of ***GUI***. Basically dialog boxes are child windows and they have certain handle numbers to the parent controls. Dialog boxes are also ***GUI*** objects and they have child objects too. Mainly dialog boxes have child objects like push buttons, toggle buttons, radio buttons, check boxes etc. Dialog boxes can be classified into many a number like 1) Message dialog boxes, 2)Help dialog boxes, 3)Error dialog boxes, 4)Wait bar dialog boxes, 5)File-open dialog boxes, 6)Warning dialog boxes, 7)Exit dialog boxes etc. We have incorporated all the seven types of dialog boxes mentioned above in ***ORMSDA*** application. Let us have a view of how dialog boxes work in ***GUI***'s. Now if we consider about the flow of control in a dialog box then we have the following view:-

A dialog box is mainly a child object under a parent window. The push button is of unidirectional flow of control because it has to notify that whether the mouse is clicked, whether the mouse is rolled over, whether the mouse is moved over it etc. Then the push button is there only to inform the dialog object of a message that is a mouse event has occurred. Check boxes are of simple type of event-catchers, that is if we check the box by clicking a mouse on it, it will send a message to the parent dialog box. Considering the Static text objects we have in general no flow of control. But we can embed an object associated with the very Static text that it has a flow of control in one direction to the parent dialog. Next we consider Toggle button. Toggle buttons generally have unidirectional flow of control towards the parent dialog. Radio buttons also have unidirectional flow of control. But Dynamic type text objects are considered to be bi directionally controlled objects. It is due to that fact that if we use the Dynamic type of text as input text objects then the flow of control will be from Dynamic text object to parent dialog object. But depending on the usage of the program writer it may have a look of output text object. There the flow of control will be from parent dialog to the

child type Dynamic text object. Hence Dynamic text type objects are with bidirectional flow of control. It is important that the dialog box itself is a child object of a main window. Hence it has to obey the bidirectional control mechanism with the main window. Dialog boxes are associated with icons. A help dialog has a specific type of icon like thinking etc. An error dialog box has cross type of icon associated with it. A warn type of dialog box has exclamatory icon within it. These icons are bit-maps with embedded properties. In general there are three types of buttons available in main window on the upper right corner of the window for minimize, restore and close operation of windows. But in child dialogs it is a completely different scenario. There exist only two button objects, one is for minimizing and another is for close operation of the child dialog box. In general the resize object of the dialog box is disabled by default. If some has to resize the dialog then the embedded resize object must be made on.

The flow of control of the dialog boxes is given in Fig.13:-

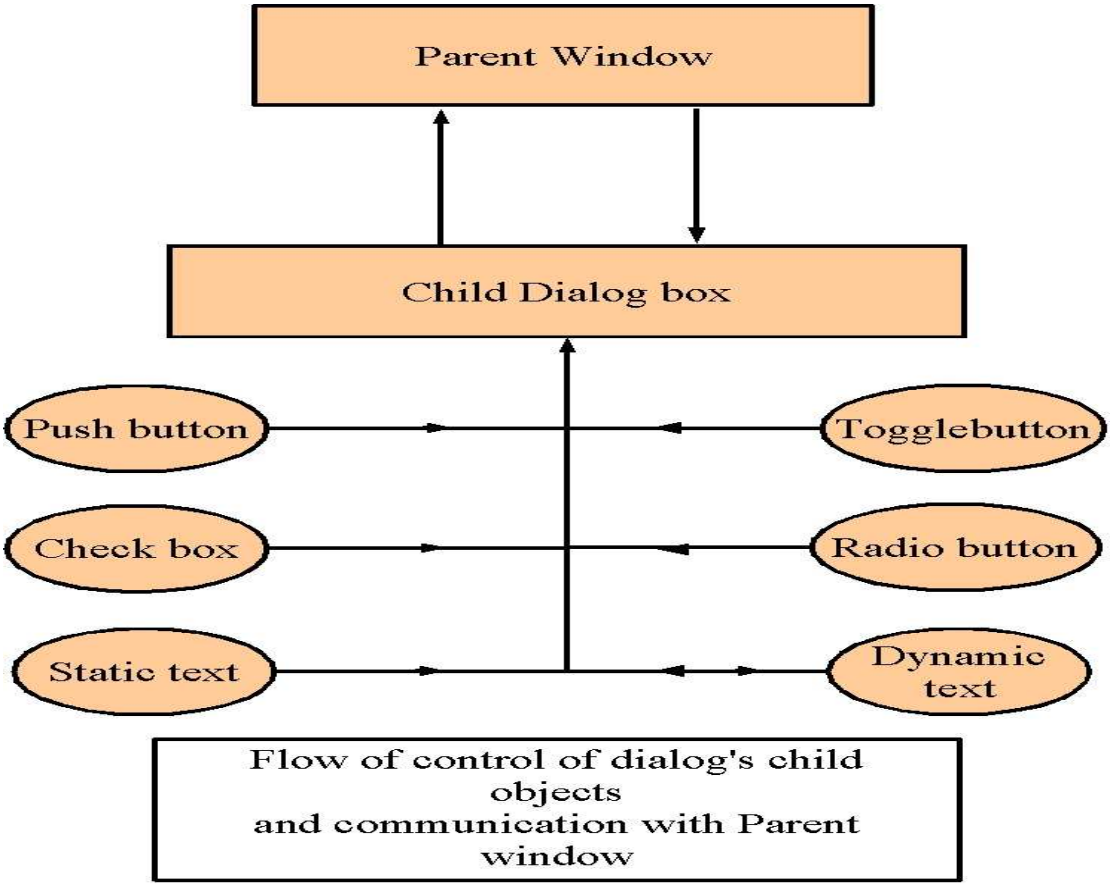



Fig.13: Block diagram of ***Flow of control of dialog's child object and communication with parent window***.

The diagram is self explanatory.

# 7) Message Dialog box:

Message dialog box has the basic characteristics of dialog boxes. See Fig.14. The name suggests the cause of nomenclature. This kind of dialog box conveys some message to the user on particular event being occurred. As we see below that the message type dialog object has embedded information for the user which is to covey the information that the authors of the application are Shubhendu Joardar & Arijit Banerjee. Message dialog box object has in general only one push button is that ***OK*** button object. Whenever the push button object ***OK*** is executed, the message dialog box object is deleted from memory. Hence ***OK*** push button is an object which calls a destructive routine to delete the message box object. Message type dialog objects can be classified into 1) help message box object, 2) warn message box object, 3) Error message box object, 4) Welcome message box object etc. Message box objects are always child objects to a main program. Whenever an event occurs, and the programmer wants to give a message to the user, the main application program calls the message box object. Message box objects

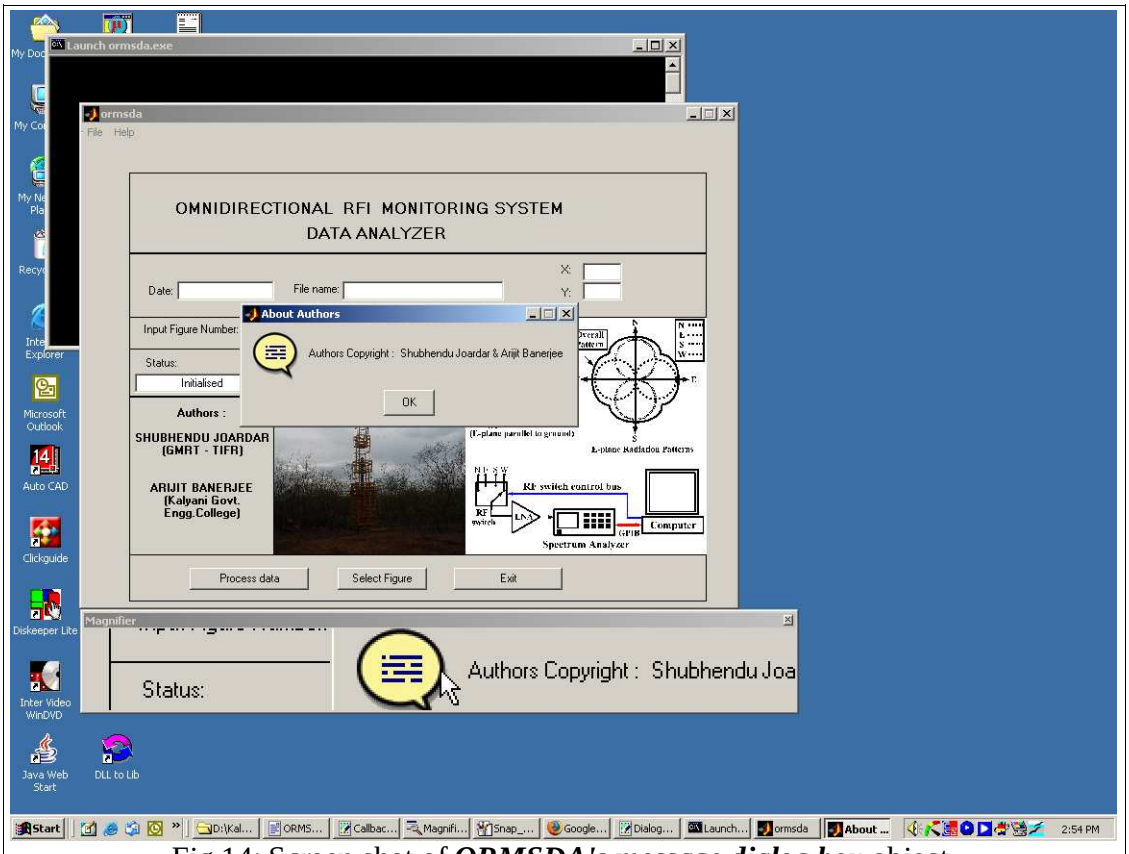


Fig.14: Screen shot of ***ORMSDA's message dialog box*** object.

generally have a bidirectional flow of control, because if the message is sent by application program to the message box, the user is accepting the message by executing the ***OK*** push button object. On the previous page we gave the example diagram of help

message box. In this page we have shown a screen shot of waning type of message box. Here the basic criteria for constructing the message dialog object are same as previous, but the difference lies in the fact of information. The warn dialog box object serves for giving warning messages only. There is an embedded graphical object, an exclamatory sign, to specify the warning. Warning type message dialog box object has in general only one push button i.e. ***OK*** button object. Whenever the push button object ***OK*** is executed, the warn dialog box object is deleted from memory. Hence ***OK*** push button is an object which calls a destructive routine to delete the warn message box object. This type of message dialog objects are always child objects generated from an exception in the program to aware the user that the user is making some mistake or going to make a mistake or the program is suffering from some ill codes or execution. Whatever it may be, warning type of message dialog box are generally can not be resized because the object is disabled to be resized by default. But a programmer's trick can enable the resizability by turning on the resize object property. On the upper right corner, not only in a warn message dialog object but also in any ordinary message dialog object, there are three buttons. One of the objects is a minimizing type object and the other is a closing type object. In here the restore down object is disabled**.** See Fig.15.

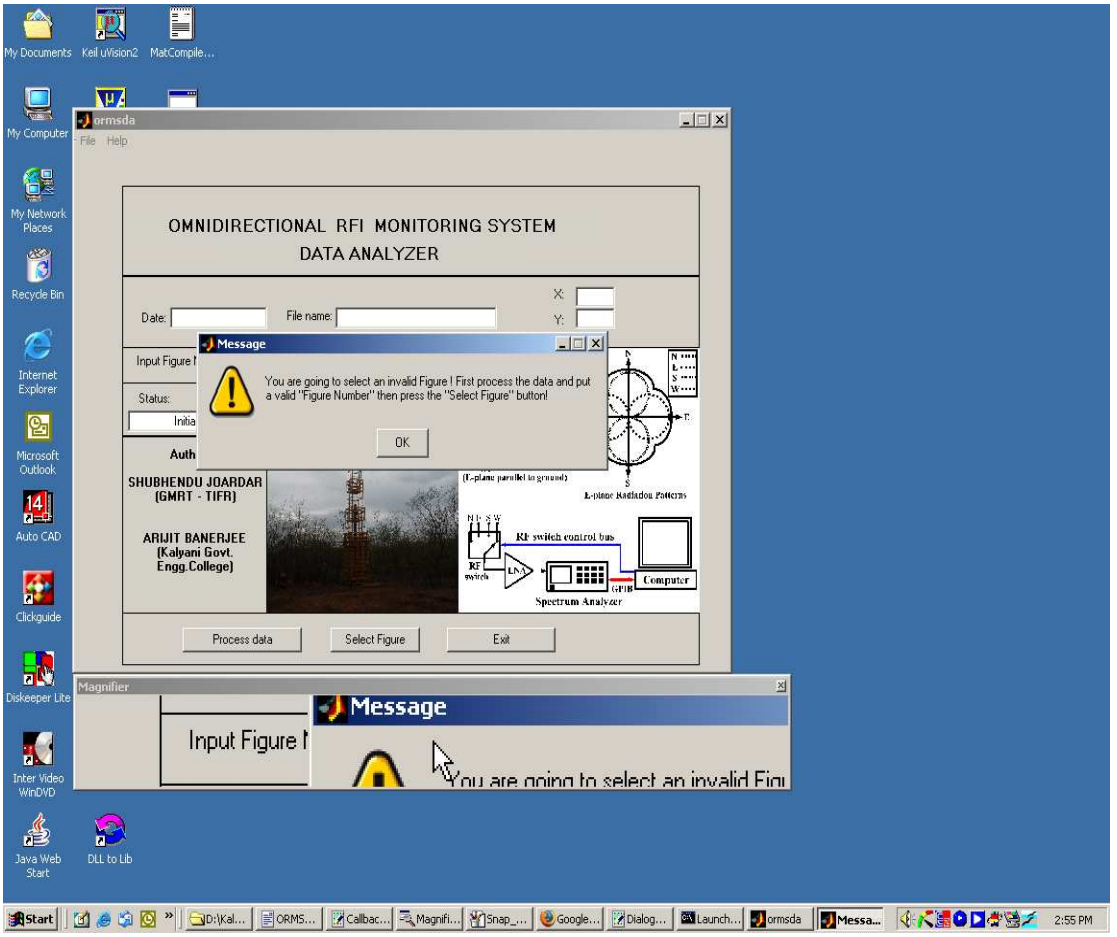


Fig.15: Screen shot of ***ORMSDA's Warn type message dialog box*** object.

# 8) Wait bar Dialog box:

The wait bar dialog box object is a simple dialog box object. It has an embedded graphical bitmap that gives the looks of the dialog object. The object has no child type push buttons. But it has an embedded static type text object. There is the main attraction which is a progressing status bar. The default color of the status bar is red. There can no callback object be made embedded in the wait bar dialog object, because it is a readymade object. The percentage of amount of task processed is linked with the progressing bar. See Fig.16.

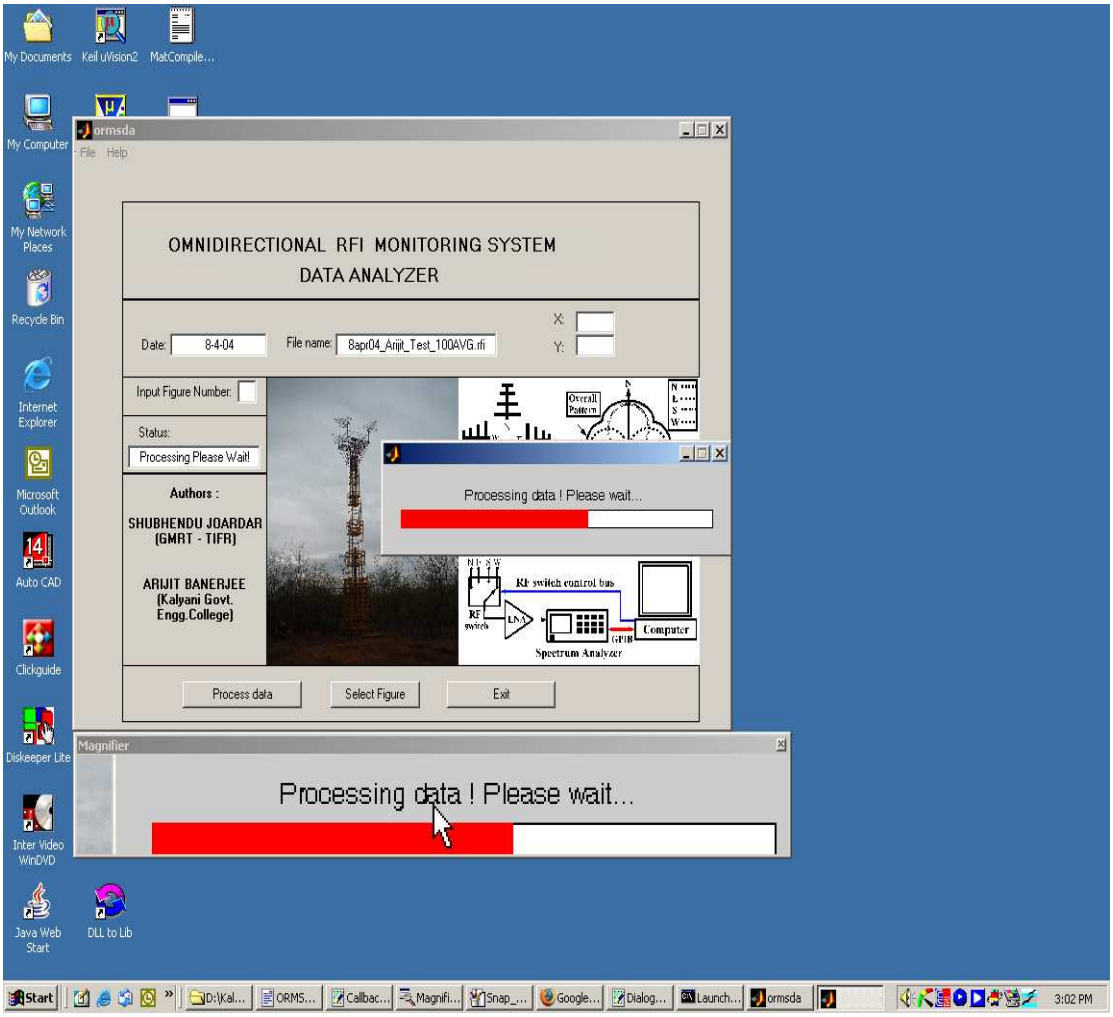



Fig.16: Screen shot of ***ORMSDA's Wait bar dialog box*** object.

# 9) Input Dialog box:

Input dialog box object is a special type of dialog object. It is an inbuilt object, but many other objects can be made embedded into it. The screen shot of an input dialog box is given in Fig.17. This type of dialog object, basically, if another object is linked with, is

the main input dialog object providing two fields for each object. One is a static text object to the linked object and another is an input text object to the linked object. Thus the name of the linked object can be specified as well the data from the user can be accessed by the linked object. There are two push button objects in the input dialog type objects. One is ***OK*** another is ***Cancel***. Clicking the ***OK*** button triggers the next object to which the flow of control is to be handed over. But the clicking of ***Cancel*** button triggers the callback object of the dialog itself and the dialog object is unloaded from the memory. See Fig.17.

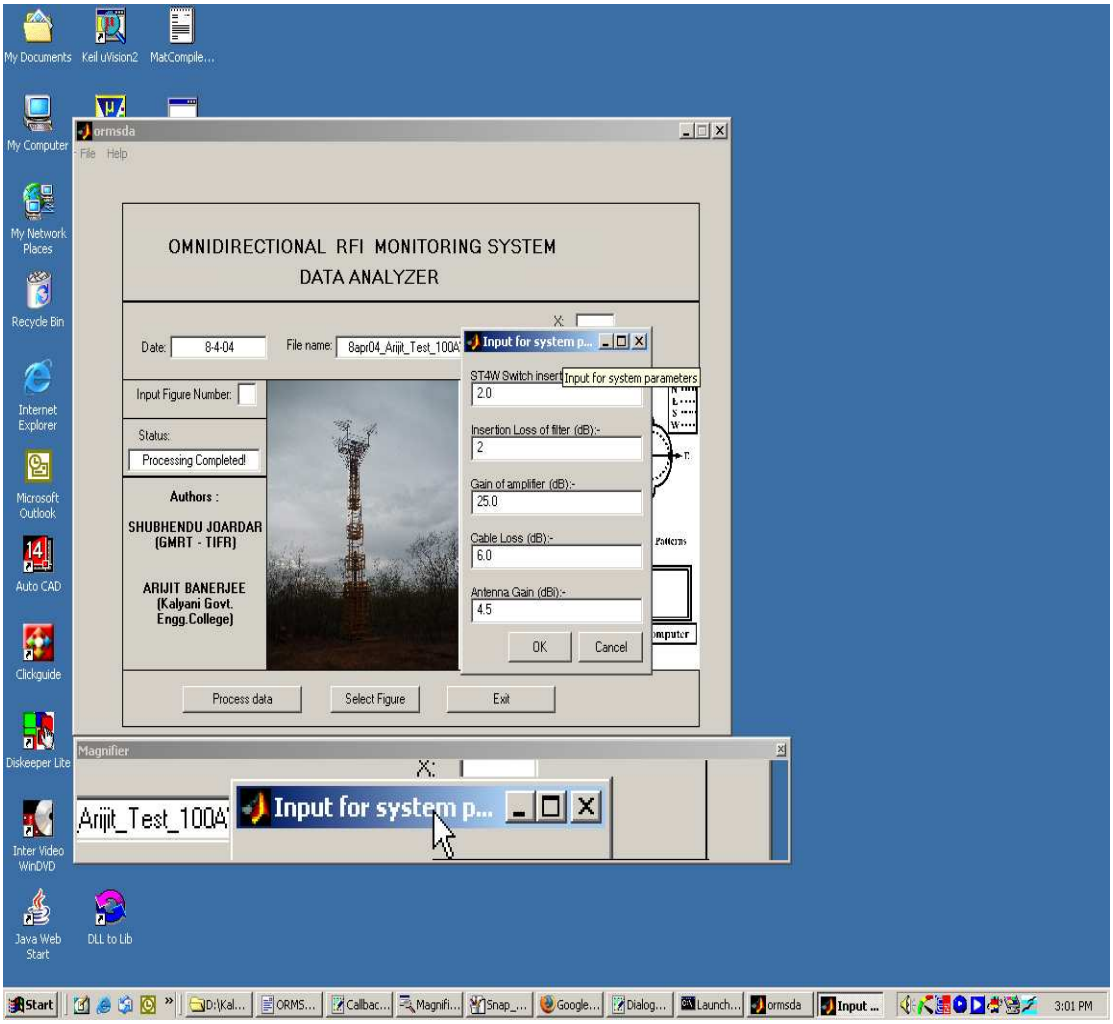


Fig.17: Screen shot of ***ORMSDA's Input dialog box*** object.

# 10) File-open Dialog box:

The file-open dialog box object is the most important one from the stand point of initialization of processing of the rfi data. The user have to provide the specified path for the rfi file and the path input should be known to the file-open dialog object. This type of object is readymade i.e. programmers can not have an access to contribute another object

to the file-open dialog or unable to modify the object settings. Taking a view into the inside of the file-open dialog box object, we have a lot of embedded objects to find. First of all let us consider the input static type text object named ***File name***. It is required to provide the path of the rfi file into the text input object associated with the ***File name*** object. Just below the ***File name*** object another static text object is there named ***File of type***. The associated input field for the object specified is an input text object as well as a popup menu object. It is basically a filter object which lets the user to find ****AVG.rfi*** files only. The other objects are push buttons, menu bar objects etc shown in the screen shot in Fig.18.

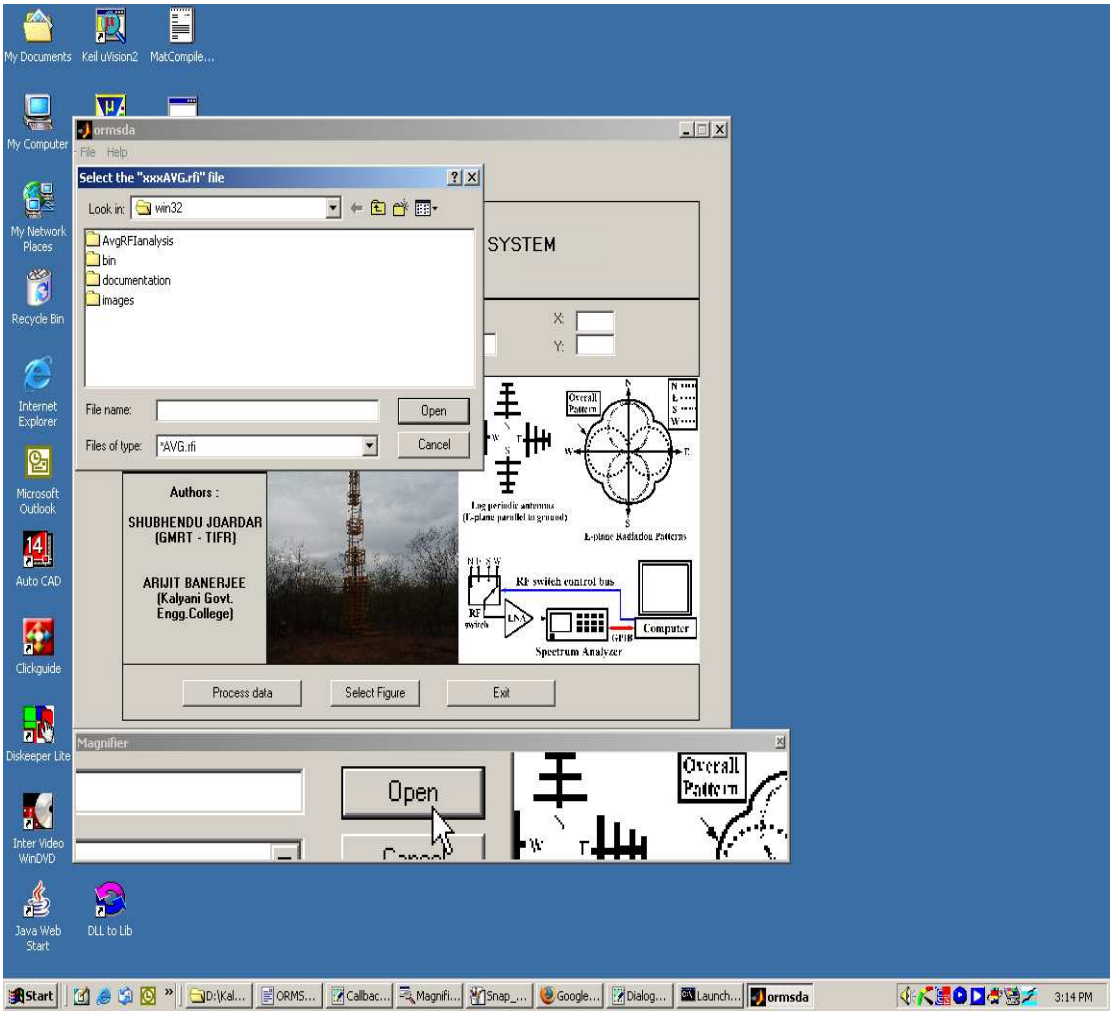


Fig.18: Screen shot of ***ORMSDA's File Open dialog box*** object.

# 11) Exit Dialog box:

Exit dialog object is an object that is use to unload the main application from memory. Simply it is used to quit the program in ***GUI*** mode. It is also a readymade object. Programmers can not change the same as well can not contribute some objects to the exit

dialog object. There are some objects embedded in the Exit dialog object. Those are push button objects, static text object, and graphical object. The static text object is to convey the information whether the user wants to quit the application or not. As there are two push buttons obviously, one is ***Yes*** for closing the application which will unload the main application from the memory and other is the ***No*** which callbacks and destroy the exit dialog box object. The graphical object is a bitmap which looks like the question mark '?' in a balloon. Exit type dialog box are always a child object to the main application window. See Fig.19.

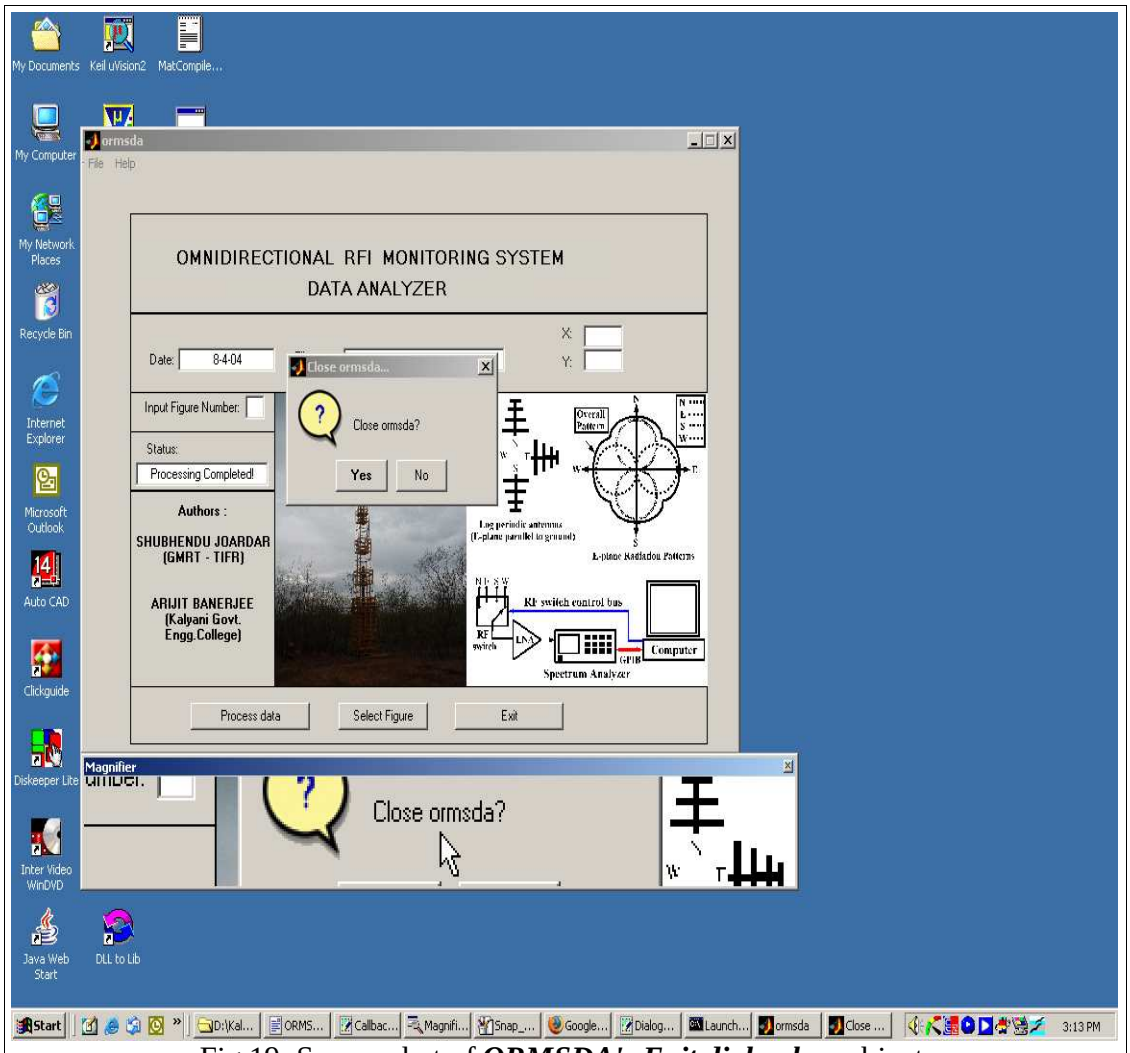


Fig.19: Screen shot of ***ORMSDA's Exit dialog box*** object.

# 12) Figure Dialog box:

Figure dialog box object are independent objects from the main application window, i.e. they are not the child of the main window. These are the most complex types object in ***ORMSDA***. Figure dialog objects, better to say figure windows have a lot of child objects

for various types of application. Only figure windows have no push buttons included inside the object. The child objects of the figure window are menu items like ***Open***, ***Close***, ***Save***, ***Save as*** etc options, function bar objects like ***new figure***, ***Open***, ***Save***, ***Print***, ***Zoom in***, ***Zoom out***, ***rotate 3-D*** etc objects. We will discuss each object later. Now in rfi analysis the output figures are four in numbers. The first or the figure number 1 shows the statistics plot of the rfi data file specified. Screen shot of the figure 1 of a given data file is provided. See Fig.20.

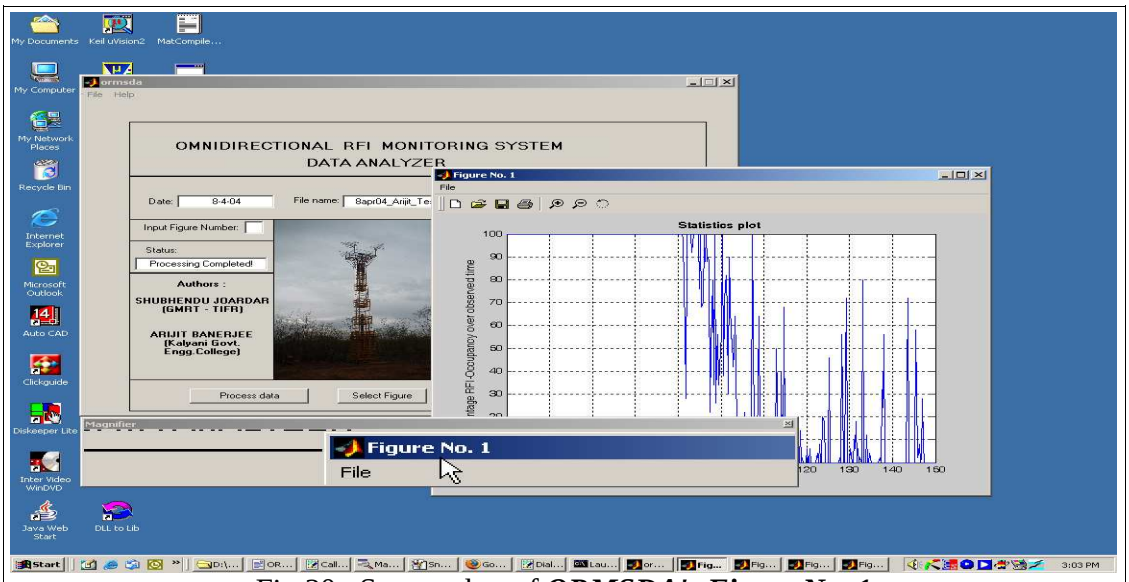


Fig.20 : Screen shot of ***ORMSDA's Figure No. 1***

The second figure i.e. the figure number 2 is the contour plot of the specified rfi data file. The contour plot screen shot is given in Fig.21.

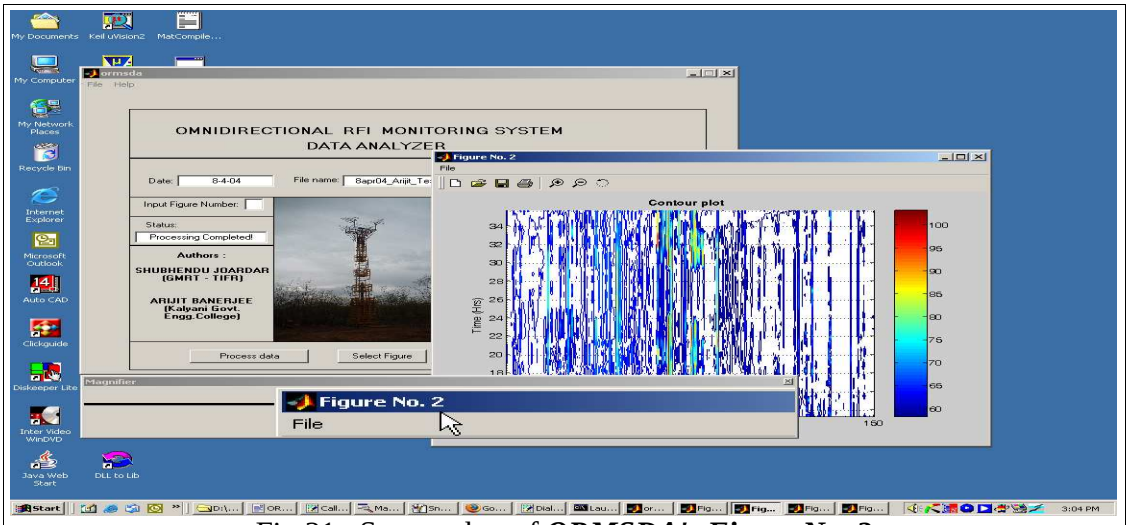


Fig.21 : Screen shot of ***ORMSDA's Figure No. 2***

The third figure i.e. the figure number 3 is the three dimensional plot of the specified rfi data file. The screen shot of 3-D plot is given in Fig.22.

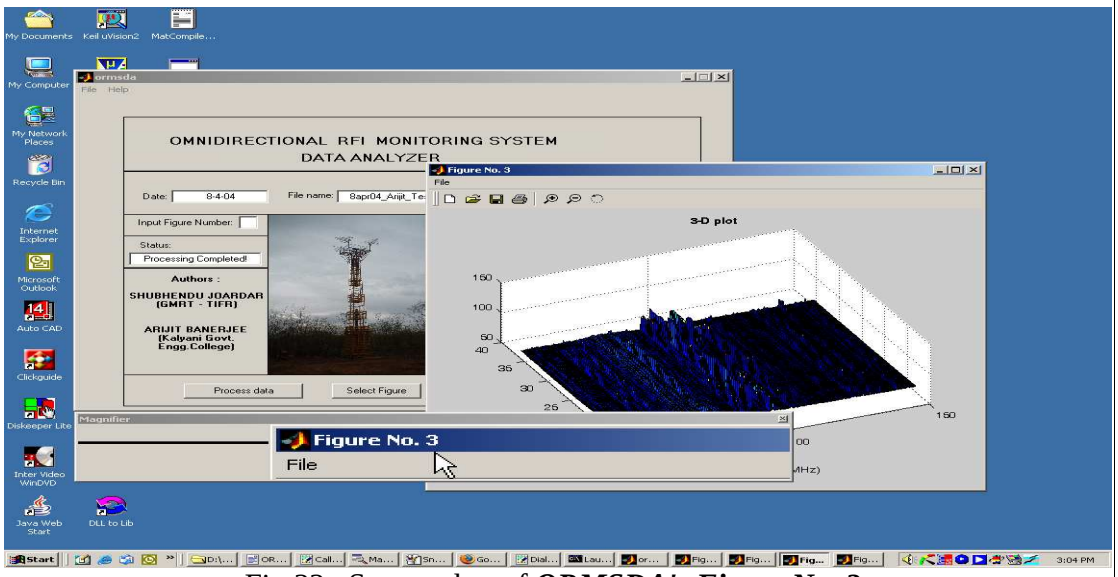



Fig.22 : Screen shot of ***ORMSDA's Figure No. 3***

The fourth figure i.e. the figure number 4 and last one is the average power plot of the specified rfi data file. The average power plot screen shot is given in Fig.23.

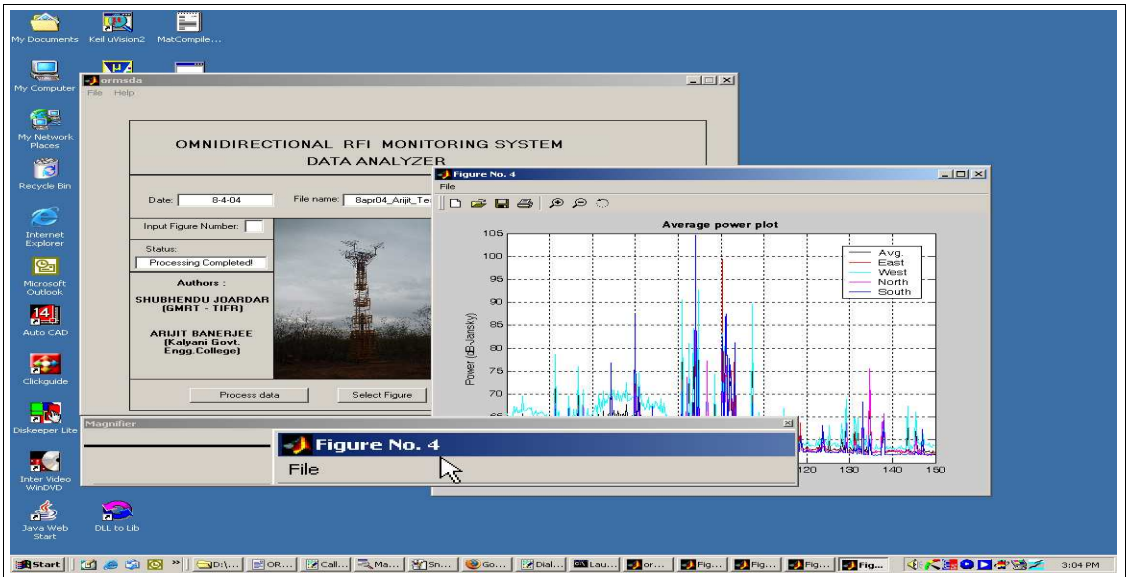



Fig.23 : Screen shot of ***ORMSDA's Figure No. 4***

Let us have the complete view of the child objects of the figure window. First component

of the function bar is ***New figure*** object. The object is to create new figures. The function of the object is to generate a free handle to the main application window and create the new figure to provide the handle for the same. The new figure is a blank figure object with no graphics object. If user wish to plot graphs, the handle of the figure should be grasped first then user may plot any thing into it. The screen shot of the object is given in Fig.24 along with magnifier window.

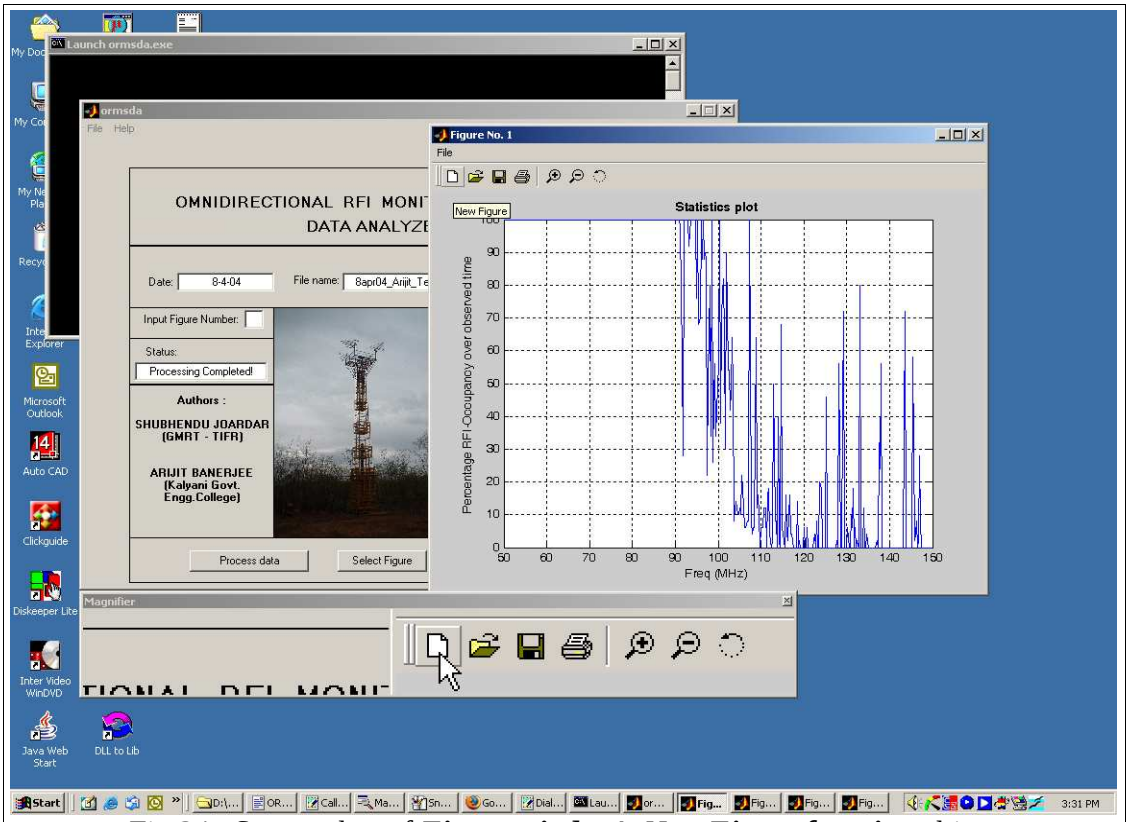


Fig.24 : Screen shot of ***Figure window's New Figure function*** object

The open file function object in the function bar is obviously designed for opening file specified by the user. After clicking the icon, the object is executed and a file-open dialog box is opened. Hence it is clear that the object has a callback embedded object for creation of file-open dialog box objects. The file-open dialog box object is the most important one from the stand point of initialization of processing of the rfi data. The user have to provide a specific path for the rfi file and the path input will be known to the file-open dialog object. This type of object is readymade i.e. programmer can not have an access to contribute another object to the file-open dialog or unable to modify the object settings. Taking a view into the inside of the file-open dialog box object, we have a lot of embedded objects to find. First of all, let us consider the input static type text object named ***File name***. It is required to provide the path of the rfi file into the text input object associated with the ***File name*** object. Just below the ***File name*** object another static text object is there named ***File of type***. The associated input field for the object specified is an input text object as well as a popup menu object. It is basically a filter object which let

the user to find ***AVG.rfi*** files only. The other objects are push buttons, menu bar objects etc. See Fig.25.

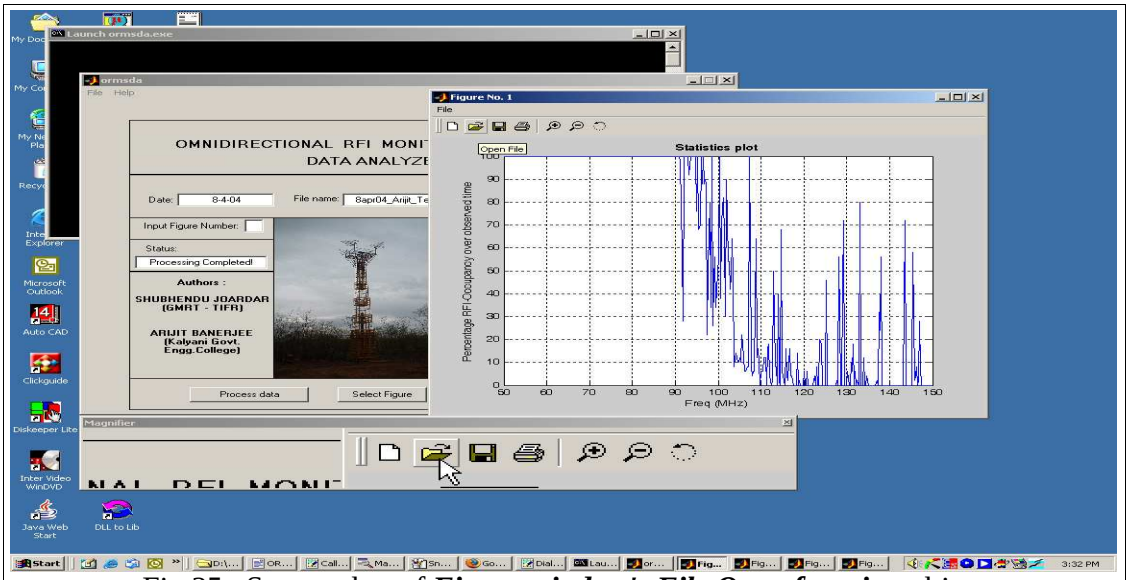


Fig.25 : Screen shot of ***Figure window's File Open function*** object

The save function object has a callback object which calls the system utility and saves the figure in different formats. It is also a readymade object. Programmer can not modify the object. See Fig.26.

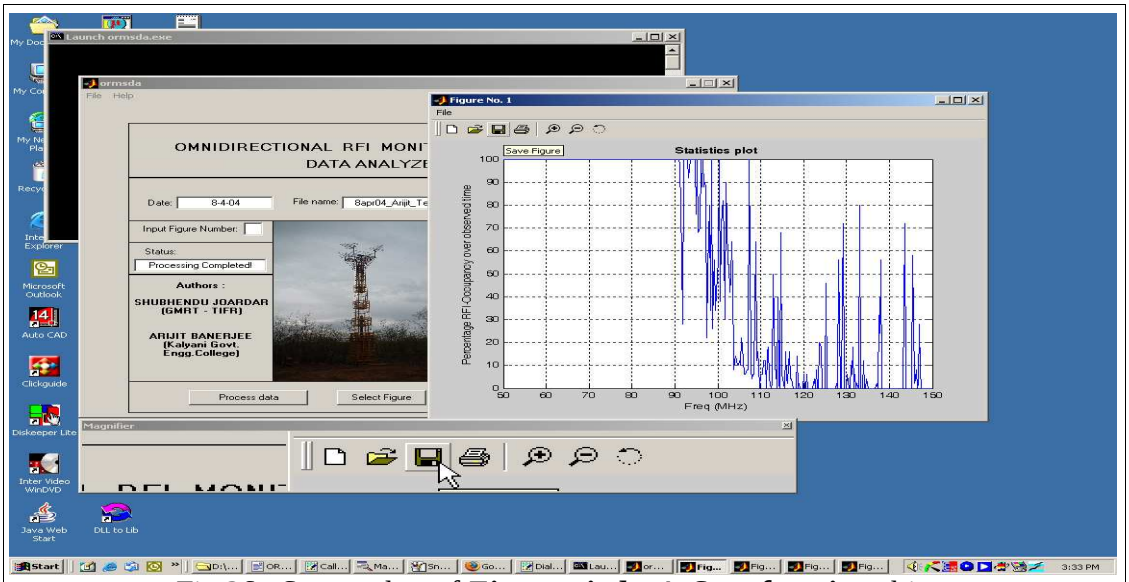


Fig.26 : Screen shot of ***Figure window's Save function*** object

***Print*** function object is to print the figure in printers avail to the system. Here also the object calls system utility to print the figure. See Fig.27.

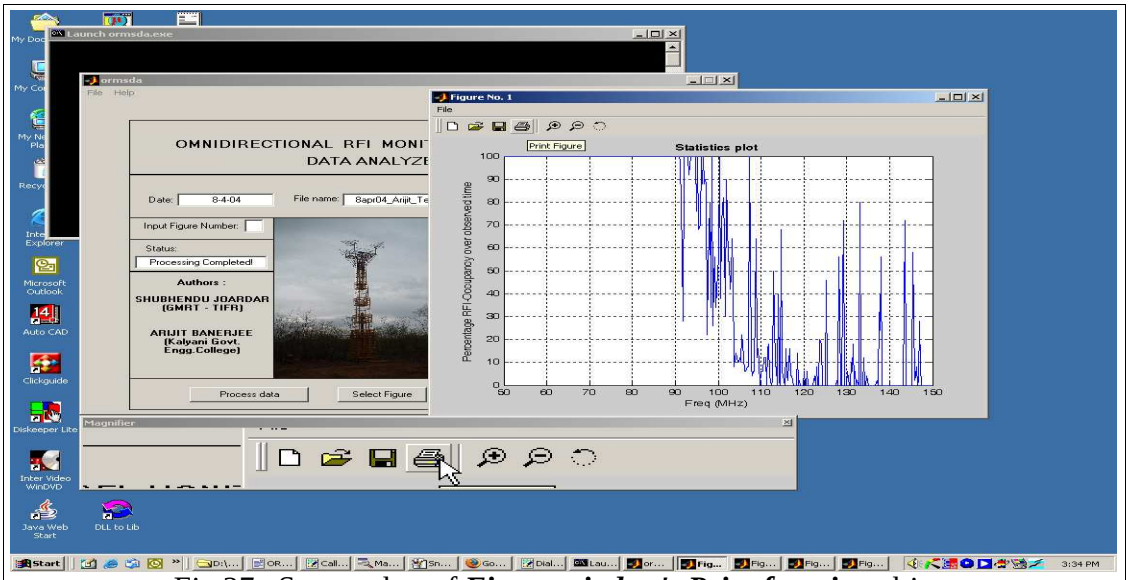


Fig.27 : Screen shot of ***Figure window's Print function*** object

The zoom in function object performs the operation of digital zooming of image. Here image processing algorithms are use in the callback object of the same. See Fig.28.

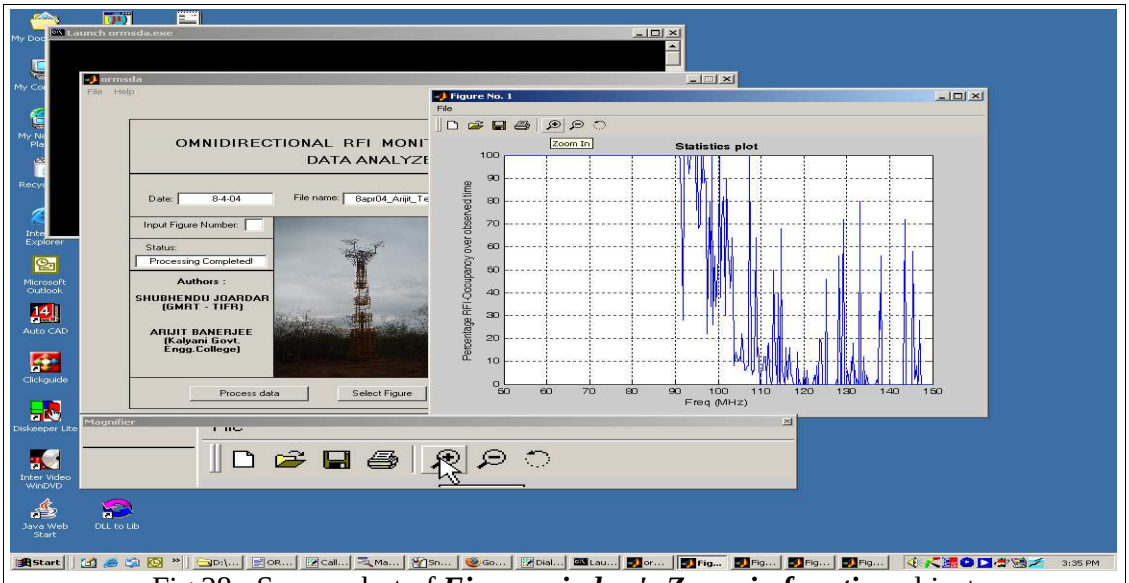


Fig.28 : Screen shot of ***Figure window's Zoom in function*** object

The zoom out function object has the similar but opposite properties of zoom in function

object. It has also an embedded callback object to implement image processing algorithms. See Fig.29.

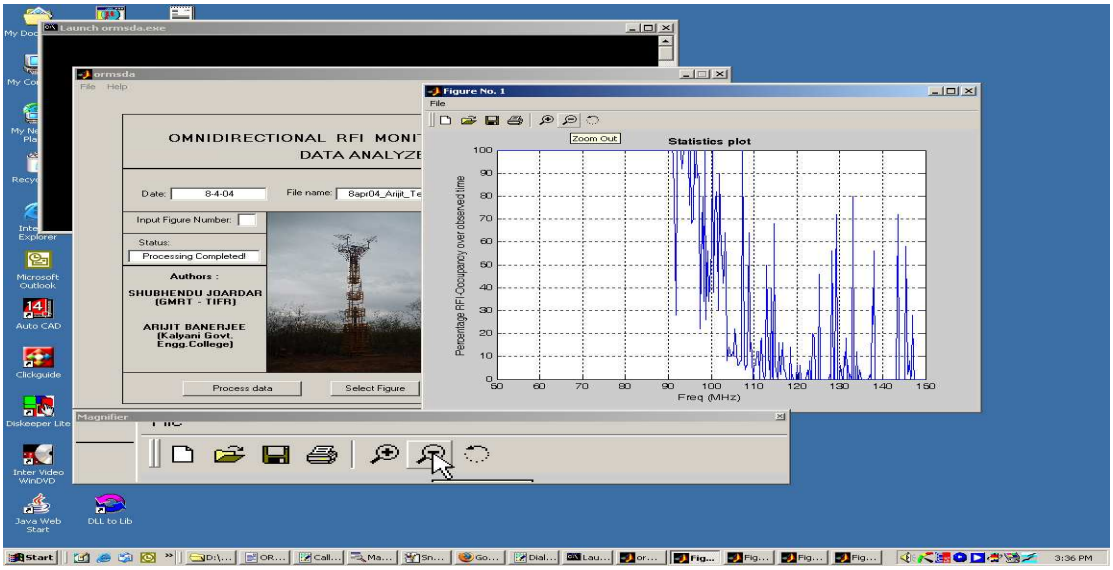


Fig.29 : Screen shot of ***Figure window's Zoom out function*** object

The rotate 3-D function object is the most complex image processing object. Here the callback embedded object is based on very complex image processing algorithms. See Fig.30.

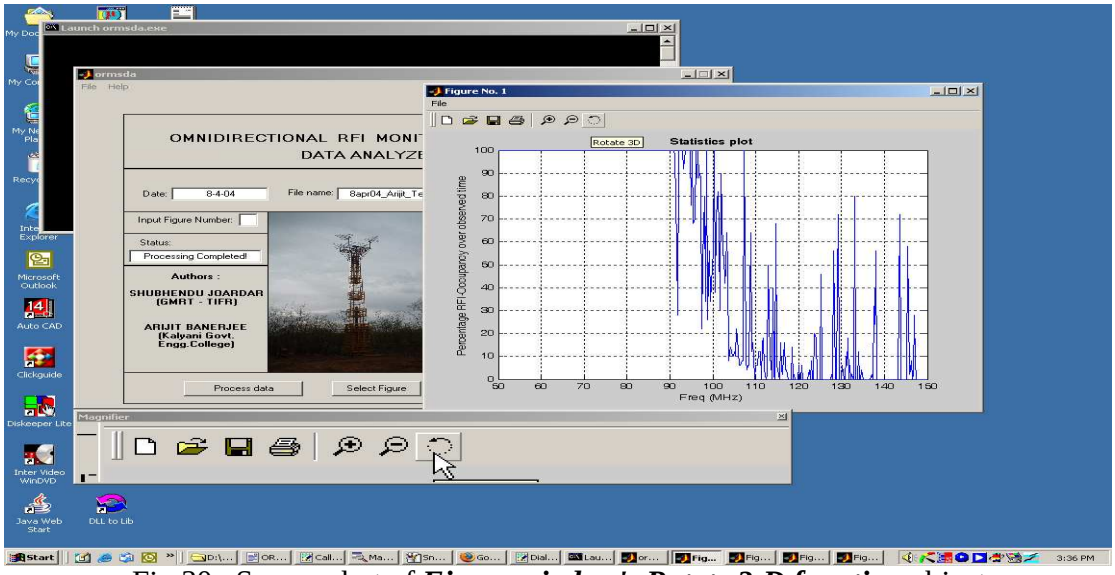


Fig.30 : Screen shot of ***Figure window's Rotate 3-D function*** object

The file menu of the figure object has the same objects like a general file menu object as shown in Fig.31.

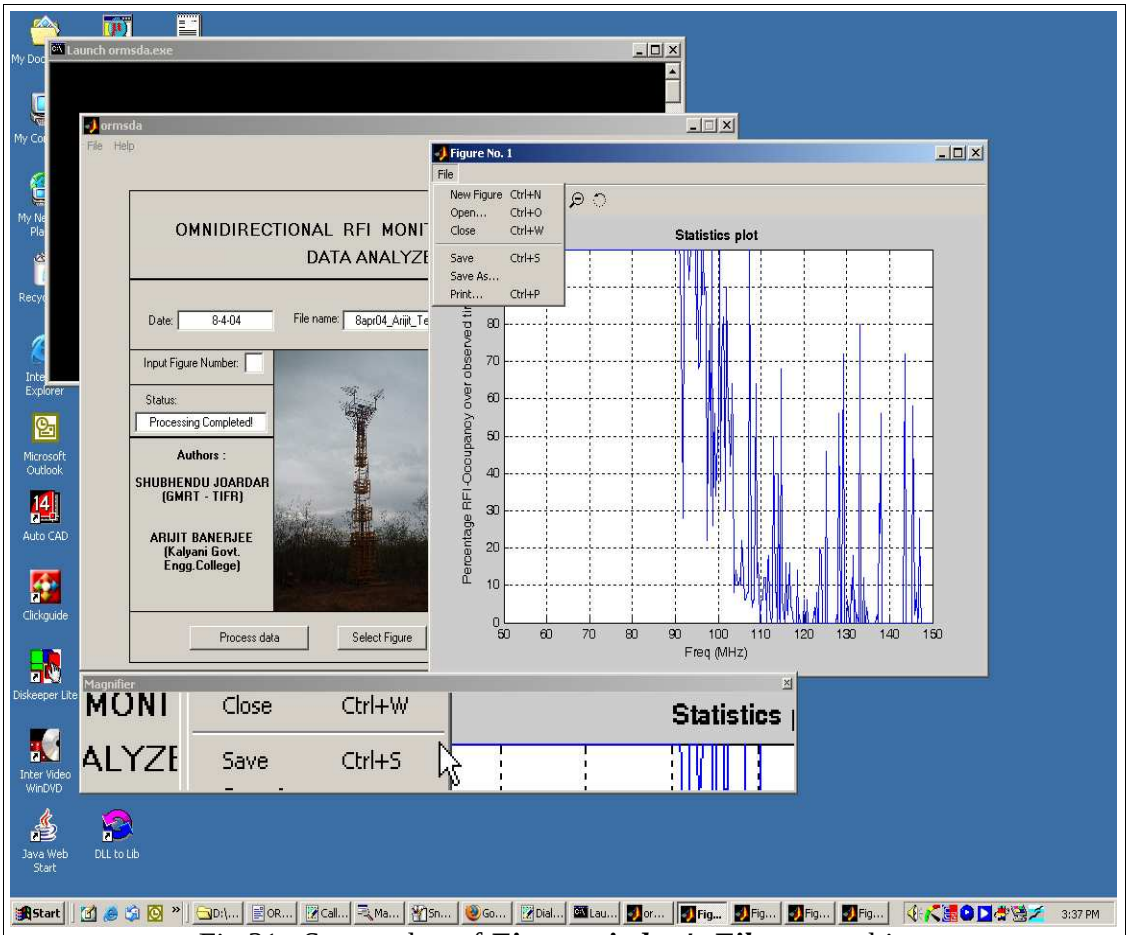



Fig.31 : Screen shot of ***Figure window's File menu*** object

# 13) Input-text Callback Objects:

Input text objects can be called back, hence they are also callback objects. Basically we have to give some input to the input text object. Whenever call backing occurs, the text is scanned by the help of the input text object and further processing progresses. In general the input text callback objects are linked with buttons. We mean the button triggers the callback text object to be scanned from the text field for input. On the other hand the input text object can be linked with mouse or keyboard operations, that is a mouse click or a key press may cause the input text object to callback. An example of this kind is a text editor. Here only one text object in the main window is programmed to behave like an input text object which is ***Input Figure Number***. See Fig.32.

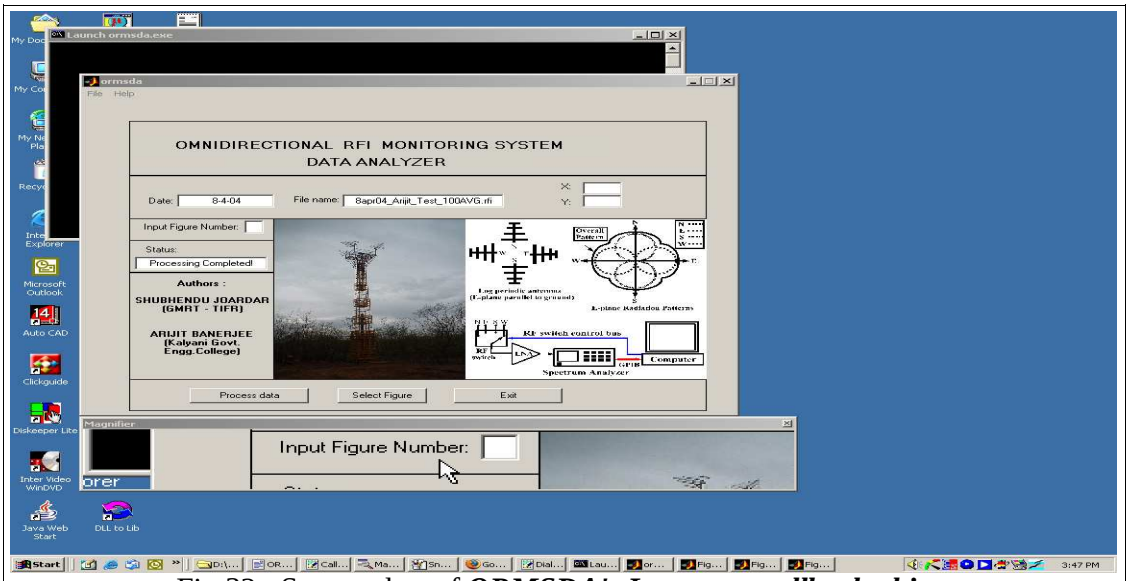



Fig.32 : Screen shot of ***ORMSDA's Input text callback object***

# 14) Output-text Callback Objects:

Output text callback objects are just opposite to the input text callback objects. Here a specific text is given for displaying it in the output text object. In ***ORMSDA*** there are many an output text callback object like ***Date:*** and ***File name:*** objects shown in the screen shot in Fig.33 and Fig.34.

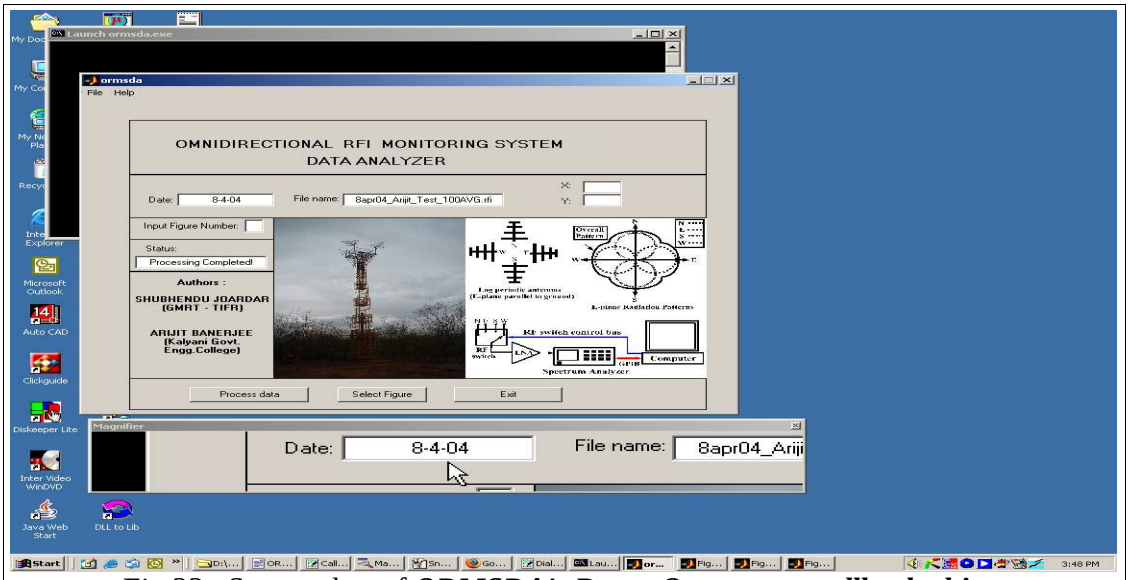



Fig.33 : Screen shot of ***ORMSDA's Date: Output text callback object***

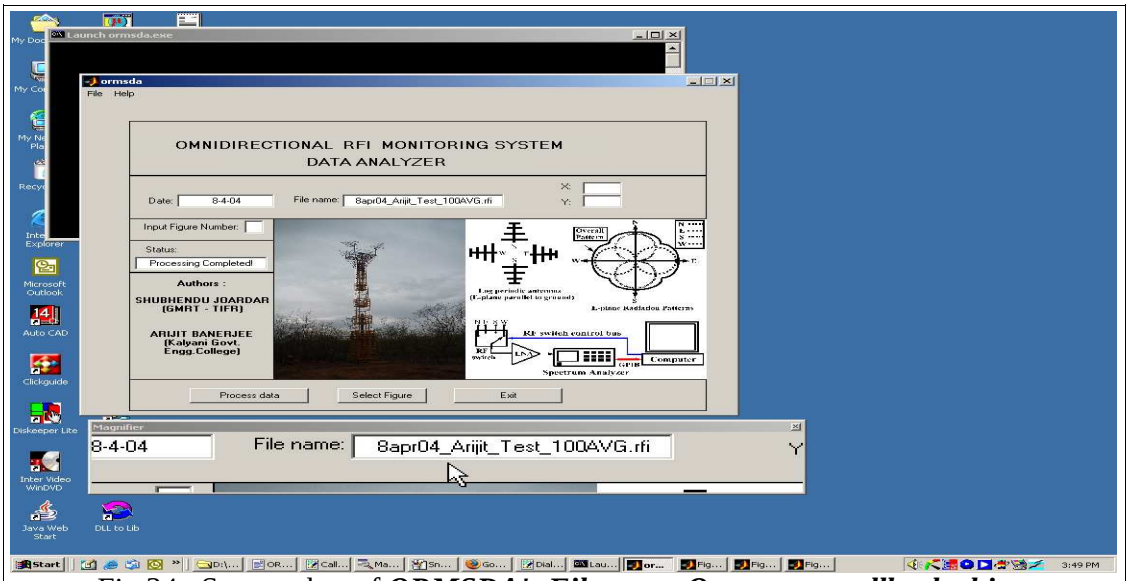



Fig.34 : Screen shot of ***ORMSDA's File name Output text callback object***

The ***X:*** and ***Y:*** fields are the x and y coordinates fields of the application window for analyzing the figures. This output text callback object is some bit complicated. The object is linked with mouse's left-click, right-click and key press of ***Esc*** key. The figures being generated and selected, a mouse left click causes the coordinates to be returned in the x & y fields. See Fig.35.

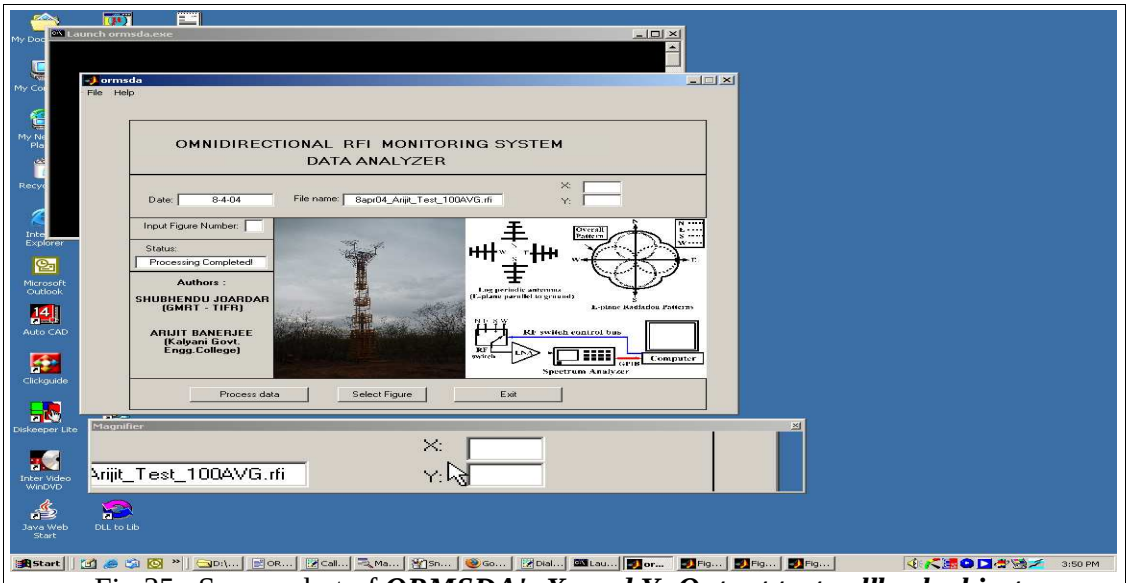



Fig.35 : Screen shot of ***ORMSDA's X: and Y: Output text callback objects***

The ***Status*** callback output text object is directly linked with the current state of flow of processing. The object displays the processing status in Fig.36.

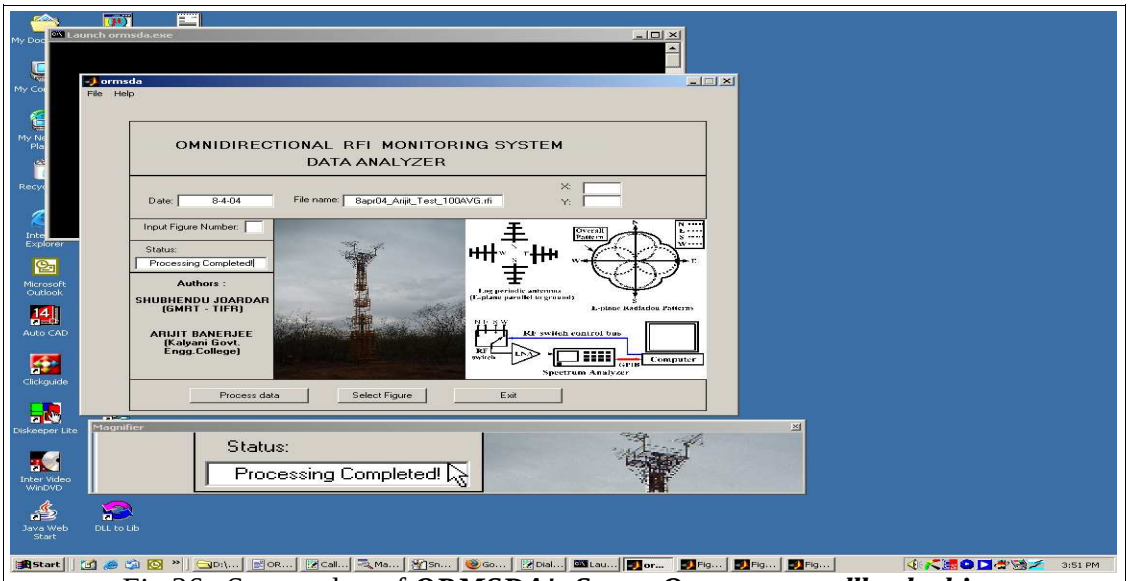


Fig.36 : Screen shot of ***ORMSDA's Status Output text callback object***

# 15) Static-text Objects:

The static text objects are very simple objects. Programmer can modify the text object at the time of compilation. But the object can not be used for inputting purpose.

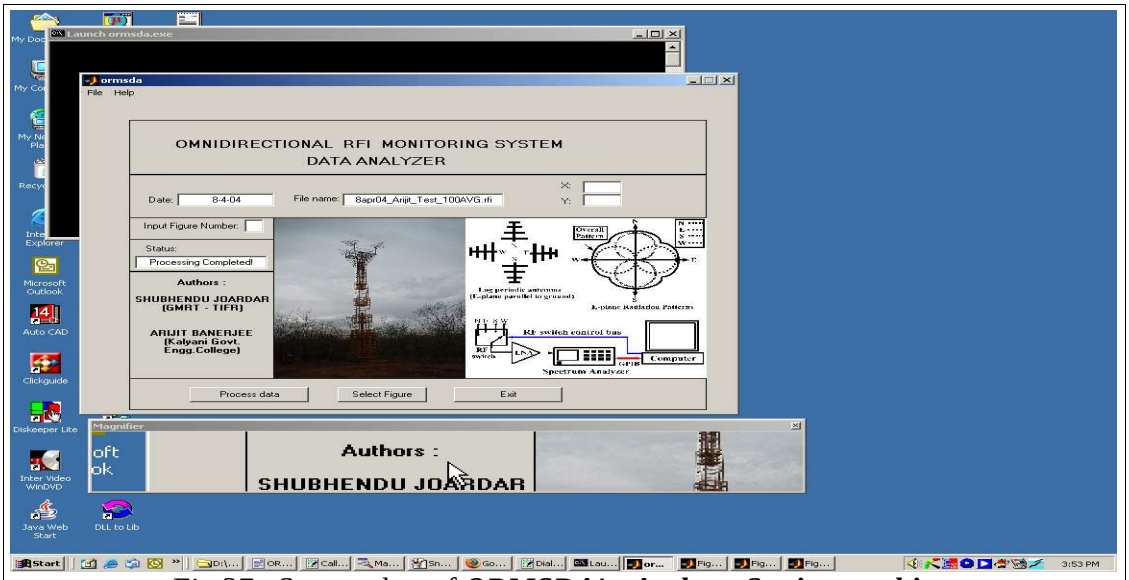


Fig.37 : Screen shot of ***ORMSDA's Authors Static text object***

***ORMSDA*** has two main static text objects given in the screen shots in Fig.37 & Fig.38.

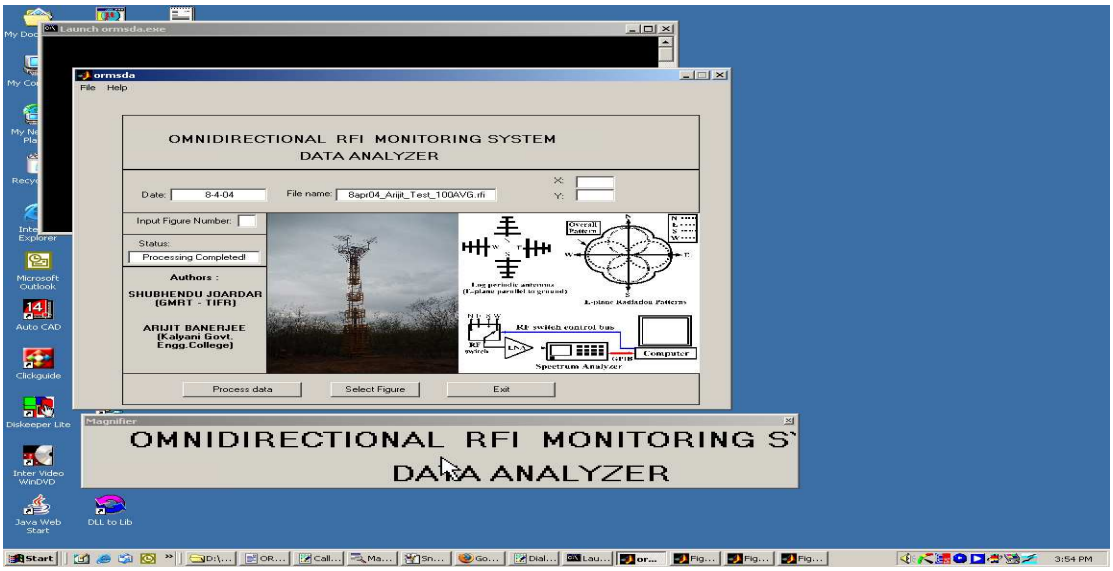



Fig.38 : Screen shot of ***ORMSDA's Title Output text callback object***

# 18) Flowchart of the Operation of the *ORMSDA*:

The flowchart of ***ORMSDA*** is given in Fig.39.

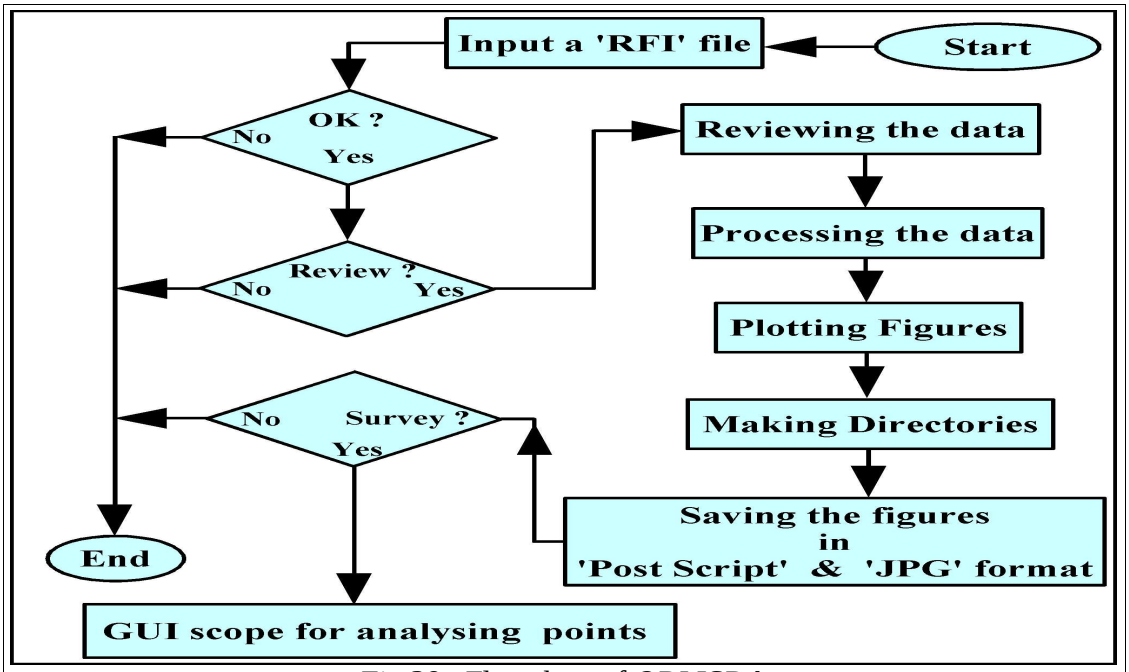



Fig.39 : Flowchart of ***ORMSDA***

The flow chart is self-explanatory.

# Chapter 3

# STANDARD OPERATING PROCEDURE OF THE OMINIDIRECTIONAL RFI MONITORING SYSTEM DATA ANALYZER

In this chapter we will give a vivid view of how analysis can be done by ***ORMSDA***.
We will discuss the following topics below:

**1) Installation of *ORMSDA*.**

**2) Running the application.**

**3) Main screen overview.**

**4) Opening a **AVG.rfi* file.**

**5) Settings of input parameters.**

**6) Analyzing the data.**

**7) Generating the figures.**

**8) Automated saving of figures.**

**9) Analyzing the figures.**

**10) Printing the figures.**

**11) The command window.**

**12) General user errors.**

**13) Uninstalling the application.**

# 1) Installation of ORMSDA:

To install the ***ORMSDA,*** just follow the steps below:-

a) Locate the ***ORMS DATA ANALYZER.msi*** named windows installer file. See Fig.40.

b) It will look like the icon below in the screen shot named ***ORMS DATA ANALYZER.msi***.

c) Double click the icon named or select the icon of ***ORMS DATA ANALYZER.msi*** and press the enter key.

d) A window will appear named ***ORMS DATA ANALYZER – Install shield wizard***. As the set up program for ***ORMSDA*** is created through install shield, the name is there.

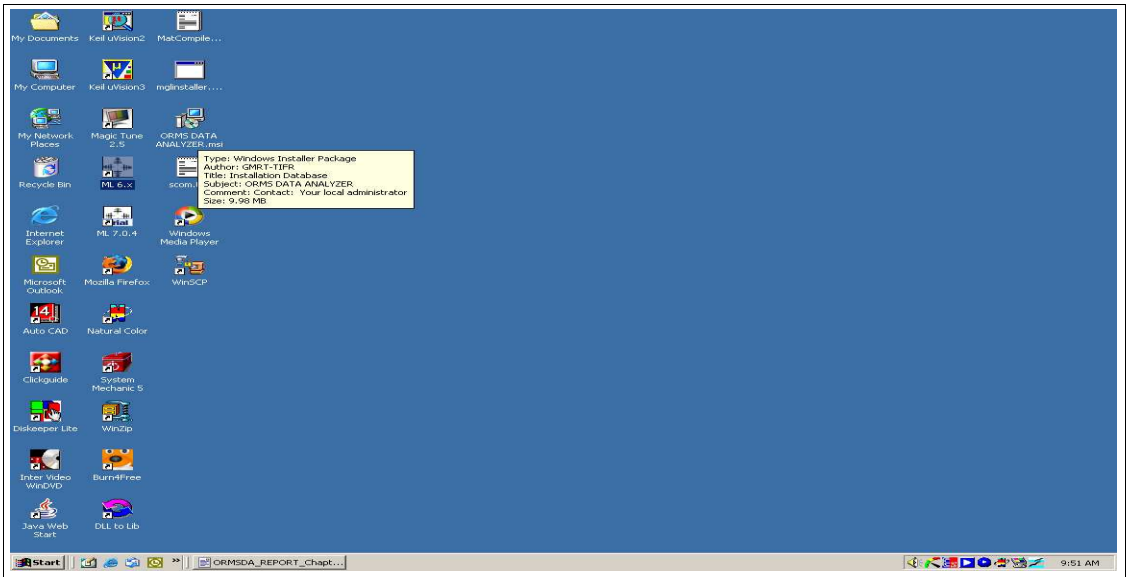


Fig.40 : Screen shot of running the ***ORMS DATA ANALYZER.msi***

e) If you do not want to install, press the ***Cancel*** button.

f) To install the program ***ORMSDA*** in your computer, press the button named 'Next'.

g) Here one thing should be notified that ***ORMSDA is specially build for the platform 'Microsoft's Windows 2000', 'Microsoft's Windows XP', 'Microsoft's Windows 2003 Server' and upwards. If you install it for 'Microsoft's Windows 98' or 'Microsoft's Windows 95' etc, it will not run correctly or give a dozens of error. Hence we recommends 'Microsoft's Windows 2000' and upwards.***

h) Install shield wizard will let you to ***modify, repair or remove*** the application ***ORMSDA***.

i) To continue with the wizard, press ***Next***. See Fig.41.

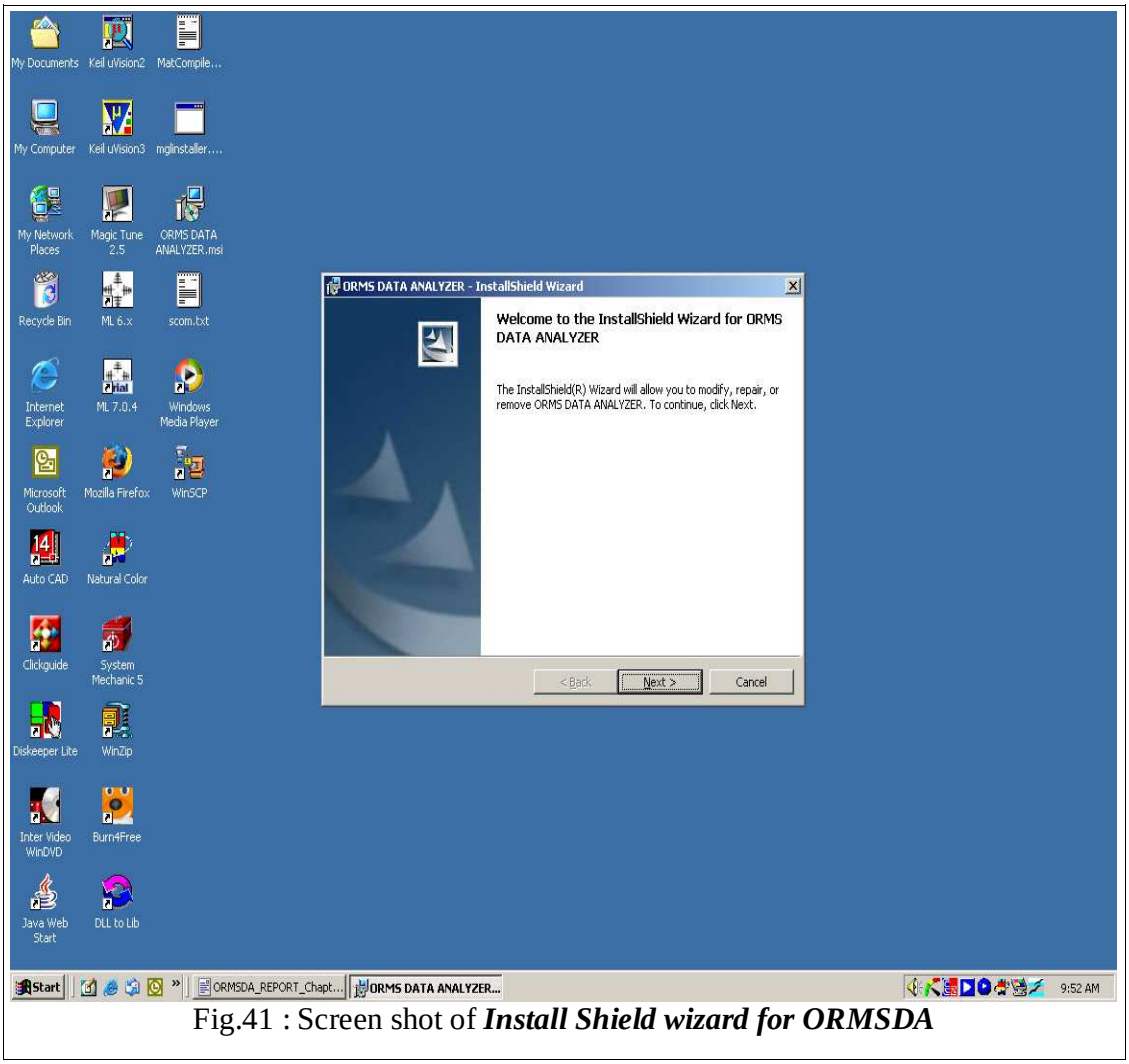



Fig.41 : Screen shot of ***Install Shield wizard for ORMSDA***

j) After pressing the ***Next*** button a new window will appear.

k) It gives you the option of changing your installation directory.

l) If you want to go back for something, press the button named ***<Back***.

m) If you don't want to install and go further, press the button named ***Cancel***.

n) If you want to change the installation directory, press Change...

o) The default directory for installation of ***ORMSDA*** is ***C:\Program Files\ORMSDA***. If you wish to continue with the default settings, click ***Next***. See Fig.42.

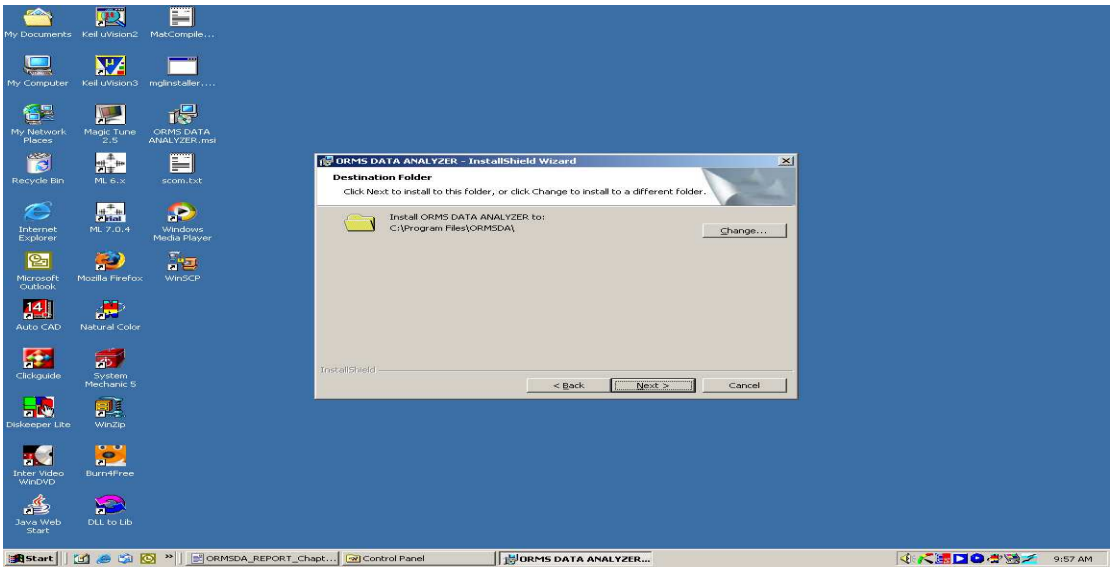



Fig.42 : Screen shot of ***Specifying the installation directory***

p) After clicking the ***Change*** button a new window will appear like in Fig.43.
You can create your own destination directory for installation or specify the installation path on the ***Folder name*** field below. We here choose ***D:\ORMSDA*** as destination directory for installation.

q) If you do not want to install the program now, quit the installation program by pressing ***Cancel***. To continue further, press ***OK***, then the previously described window will appear again with your newly given path, specified for installation directory. See Fig.43. Then click the button named ***Next***.

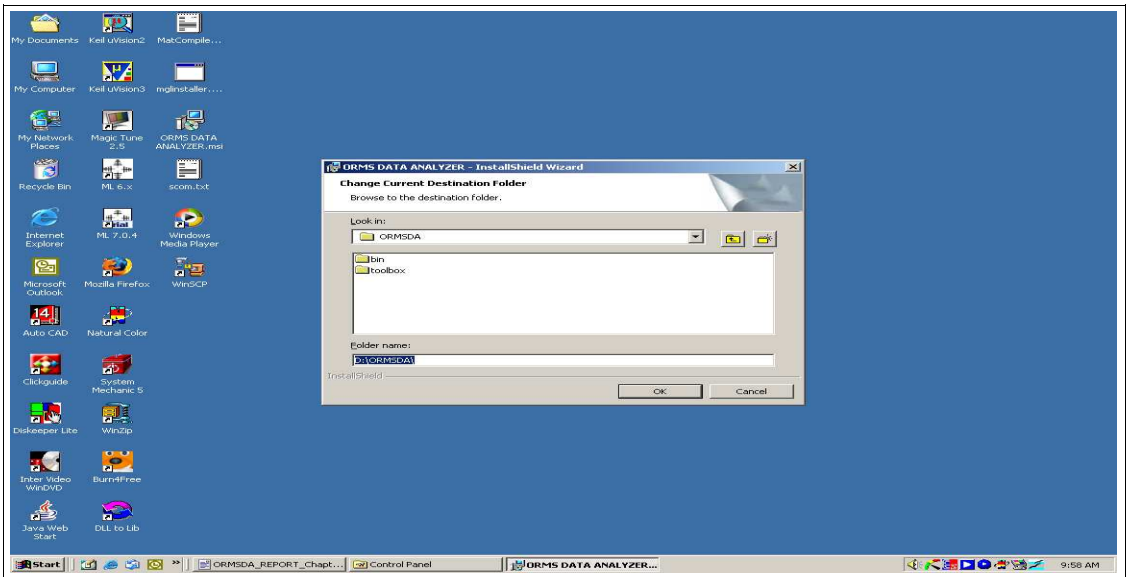



Fig.43 : Screen shot of ***Providing the path of installation***

r) After clicking the button named ***Next***, a new window will appear as shown below.

s) It shows ready to install dialog. If you want to review or change the installation settings you may click ***<Back*** and do whatever you want. It is the last step for installation settings.

t) If you do not want to install the application, quit it by pressing the ***Cancel*** button.

u) To install the application, press the Install button. See Fig.44.

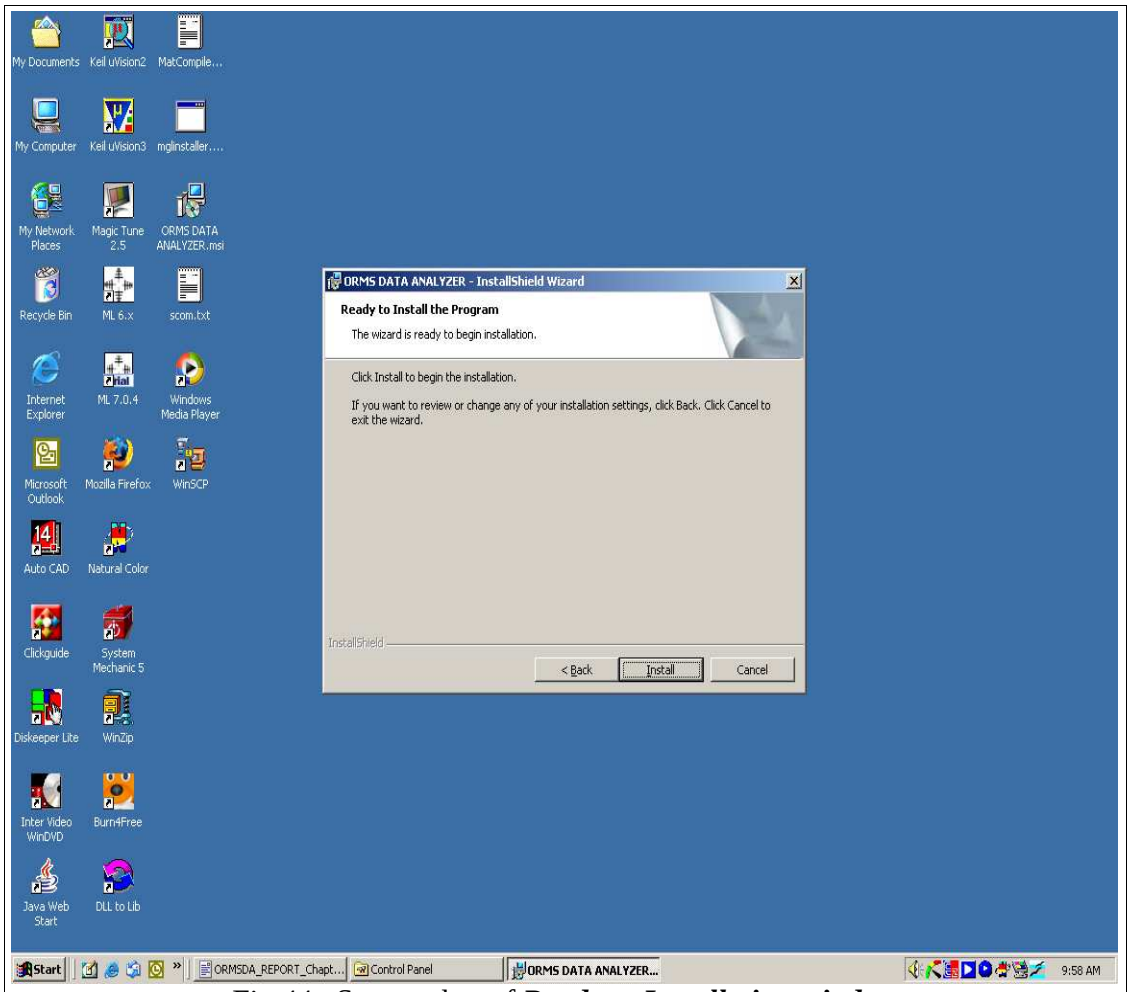



Fig.44 : Screen shot of ***Ready to Installation window***

v) After clicking the ***Install*** button a new window will appear like in Fig.45.

w) If you want to abort installation, press the ***Cancel*** button. If you want to install, do not disturb the setup until it is completed.

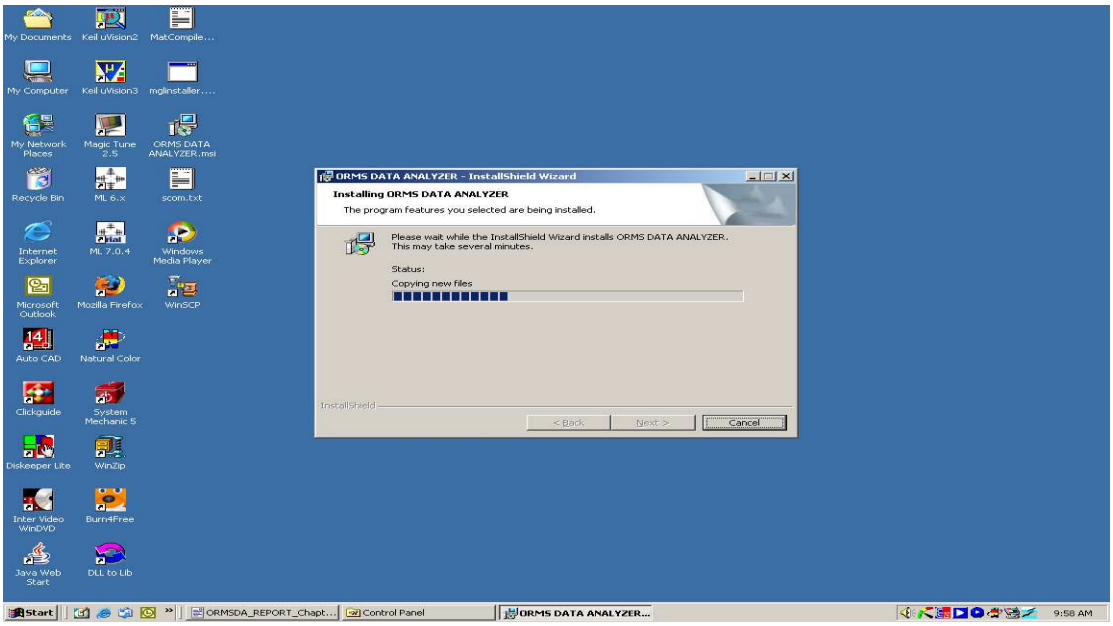



Fig.45 : Screen shot of ***Installation progress***

x) At the end of installation another window will appear showing that the install shield wizard is completed.

y) To finish the installation click ***Finish*** and ***ORMSDA*** is installed in your computer. See Fig.46.

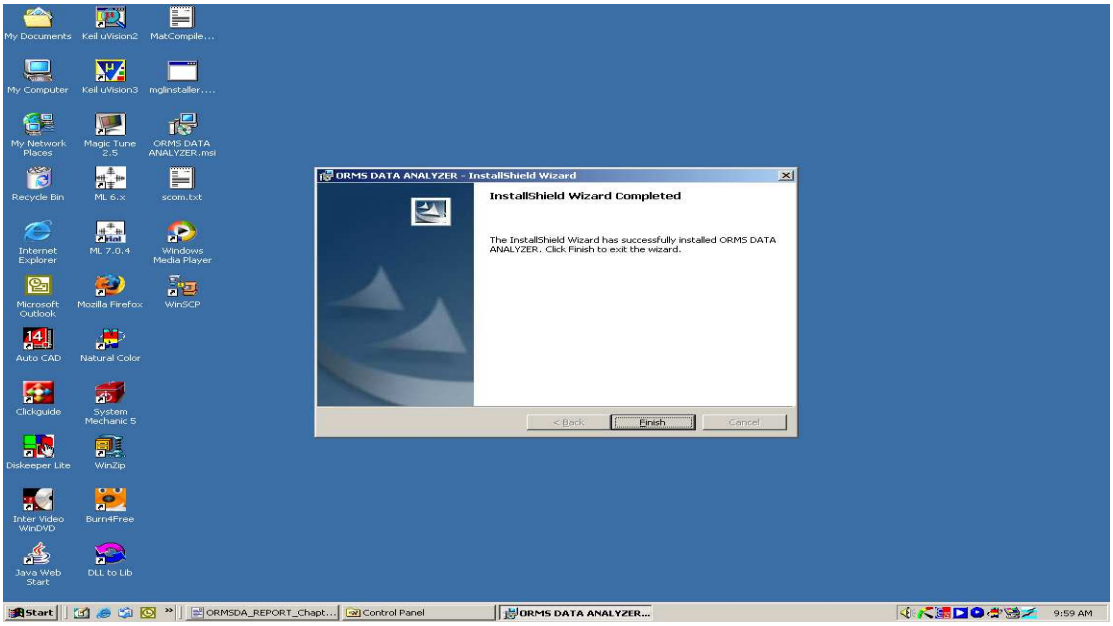



Fig.46 : Screen shot of ***Finishing Installation***

# 2) Running the application:

To run the application, follow the steps below:-

a) The ORMSDA will be installed in the start menu's programs menu. Go to the ***Start*** menu: - ***Programs: - GMRT-TIFR: - ORMS DATA ANALYZER*** as shown below in the figure. You will find a ***Launch ormsda.exe*** shortcut. To run the application, click on the ***Launch ormsda.exe*** shortcut. See Fig.47.

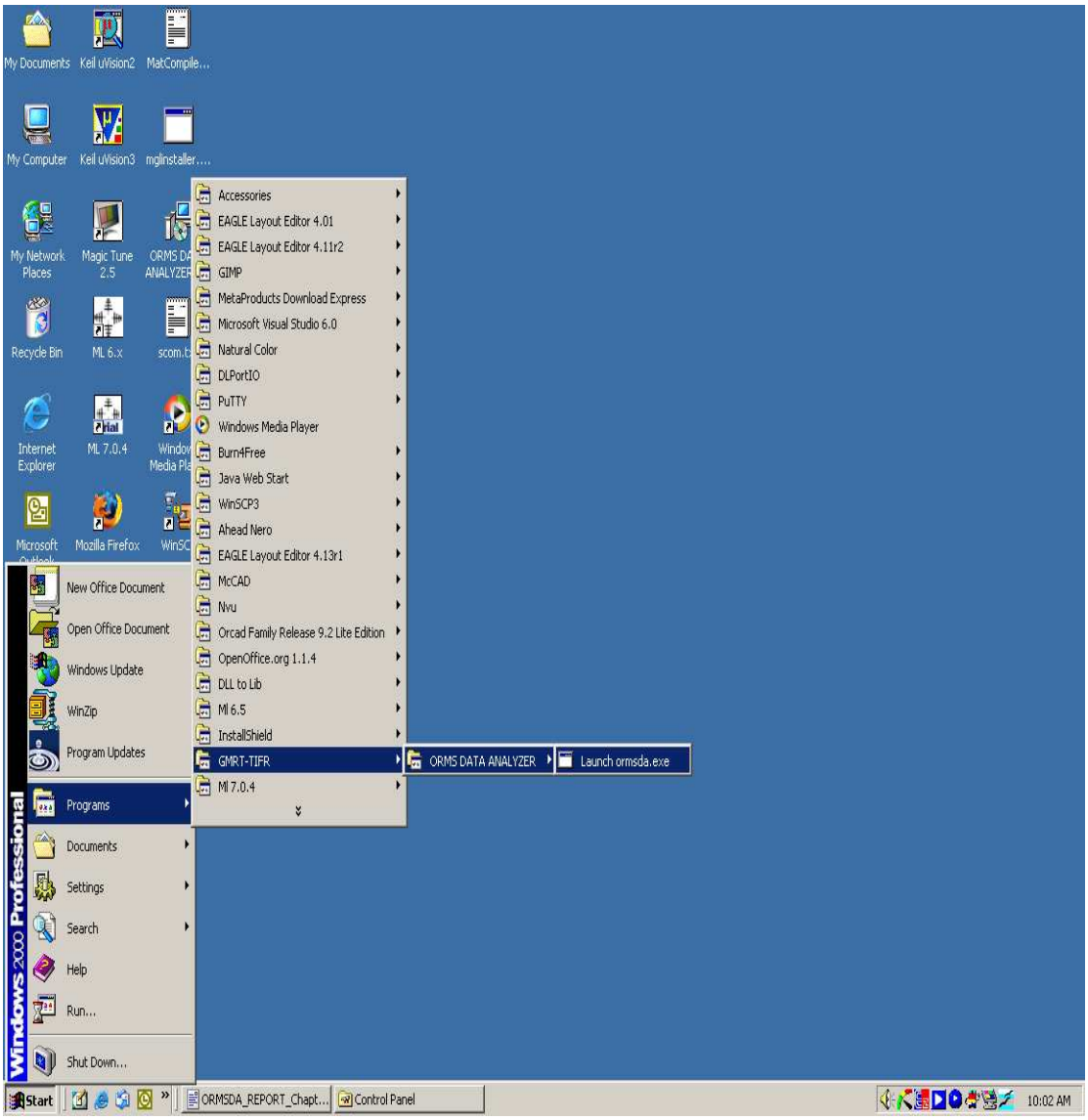



Fig.47 : Screen shot of ***Launching the application***

b) After clicking the shortcut a new command window will appear like below in Fig.48. Wait a while.

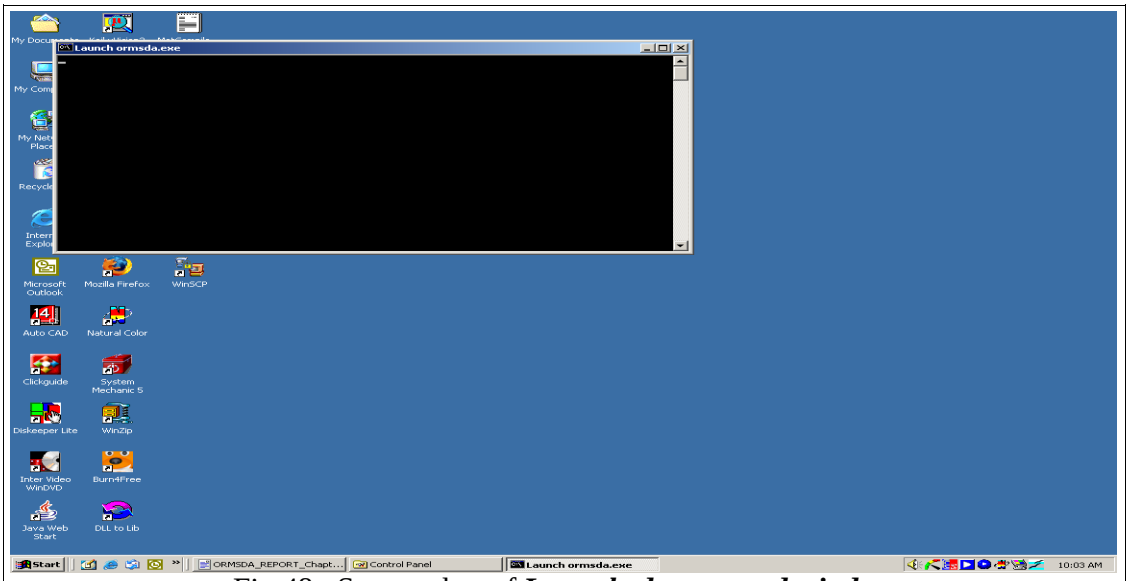



Fig.48 : Screen shot of ***Launched command window***

c) After waiting a while a new ***GUI*** window will appear like below in Fig.49. This is our main window of ***ORMSDA*** application. Now you can go further in processing the ****AVG.rfi*** files.

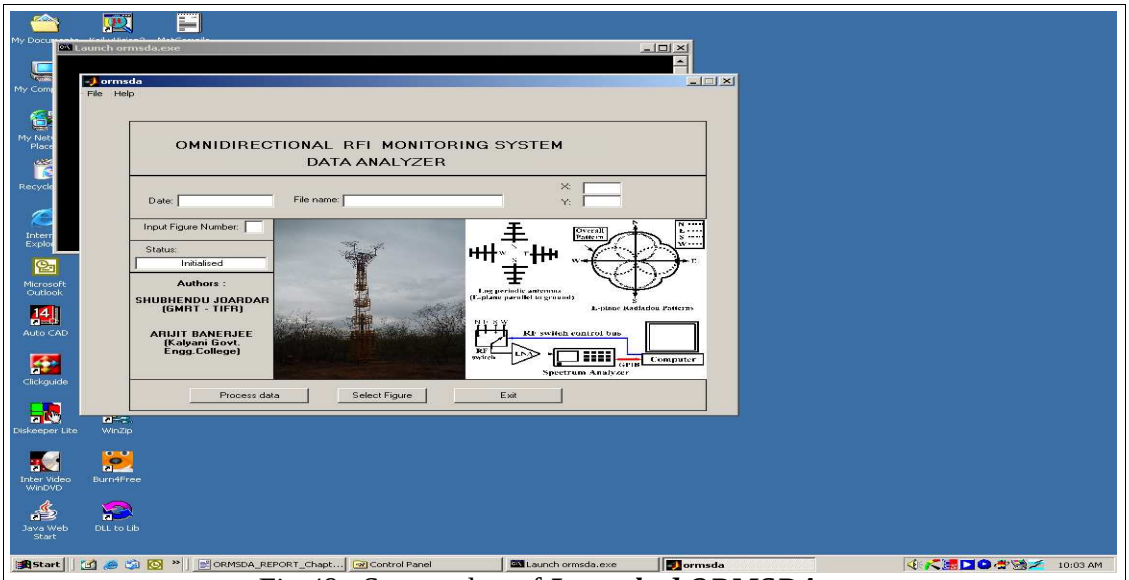



Fig.49 : Screen shot of ***Launched ORMSDA.exe***

# 3) Main screen overview:

a) The screen shot in Fig.50 shows most of the components of the main window. Here in ***magnifier*** window we see the file menu. File menu has two sub menus. One is open sub menu to open ****AVG.rfi*** files. The other one is ***Exit*** sub menu to quit the application.

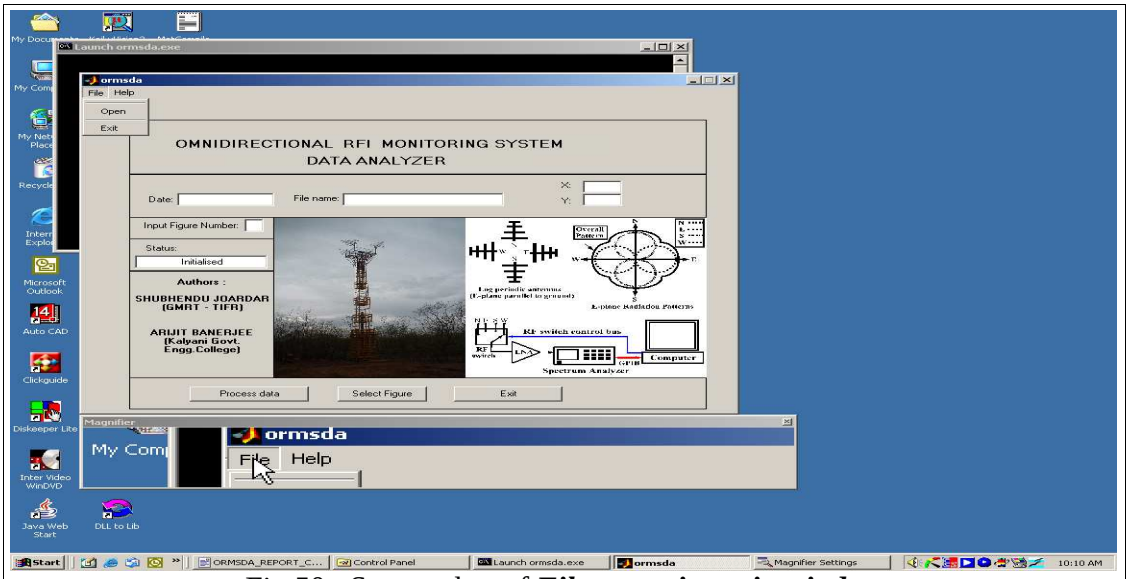



Fig.50 : Screen shot of ***File menu in main window***

b) In the screen shot in Fig.51 help menu is highlighted in the ***magnifier*** window. The help menu has two sub menus one is ***Documentation*** sub menu by which user can read the ***ORMSDA*** documentation file. The other sub menu is ***About Authors*** in which the authors' name is provided.

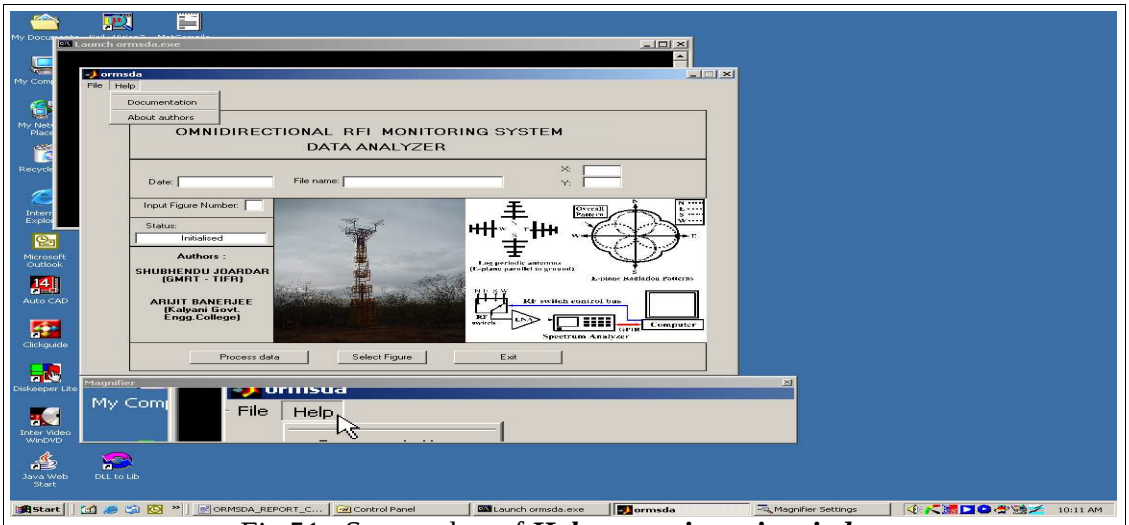



Fig.51 : Screen shot of ***Help menu in main window***

c) In the Fig.52 below, screen shot of the highlighted thing named ***Date:*** is an output text object which displays, after processing, the date of the input ****AVG.rfi*** file.

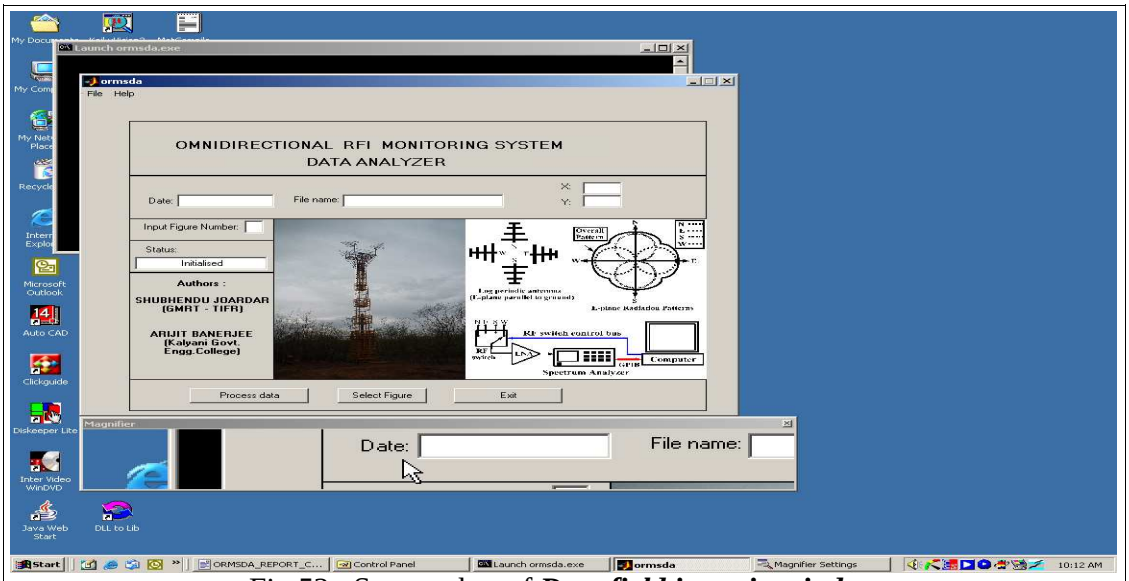



Fig.52 : Screen shot of ***Date field in main window***

d) The below picture in Fig.53 shows the highlighted object named ***File name*** in the ***magnifier*** window. The object is an output text, it shows the name of the ****AVG.rfi*** file after processing.

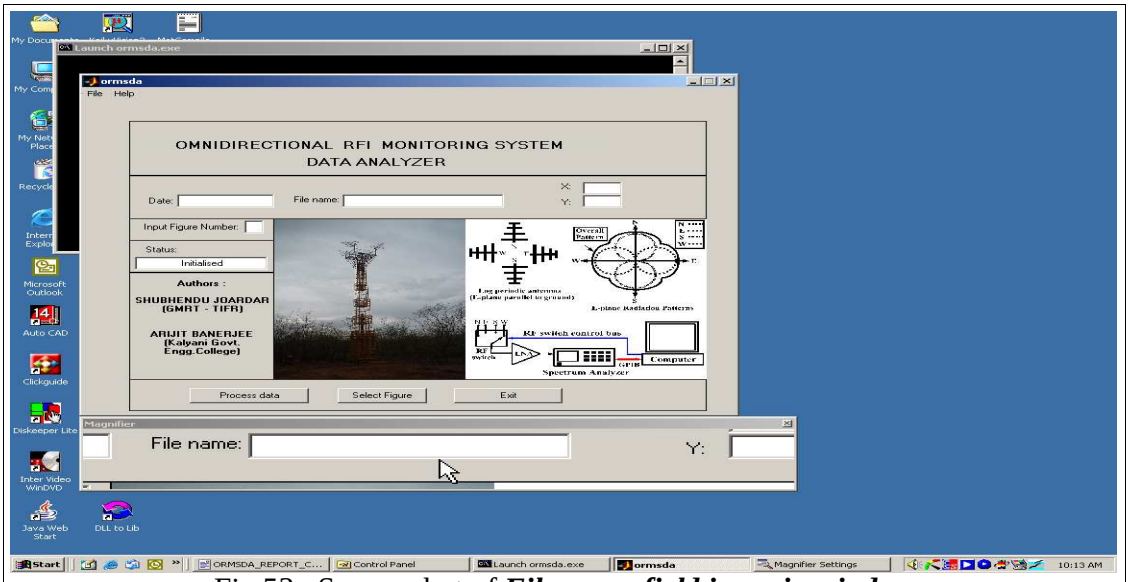



Fig.53 : Screen shot of ***File name field in main window***

e) The screen shot shows the highlighted component ***X*** & ***Y***. These are the x & y coordinate position of the mouse pointer after processing and selecting one of the figures. See Fig.54.

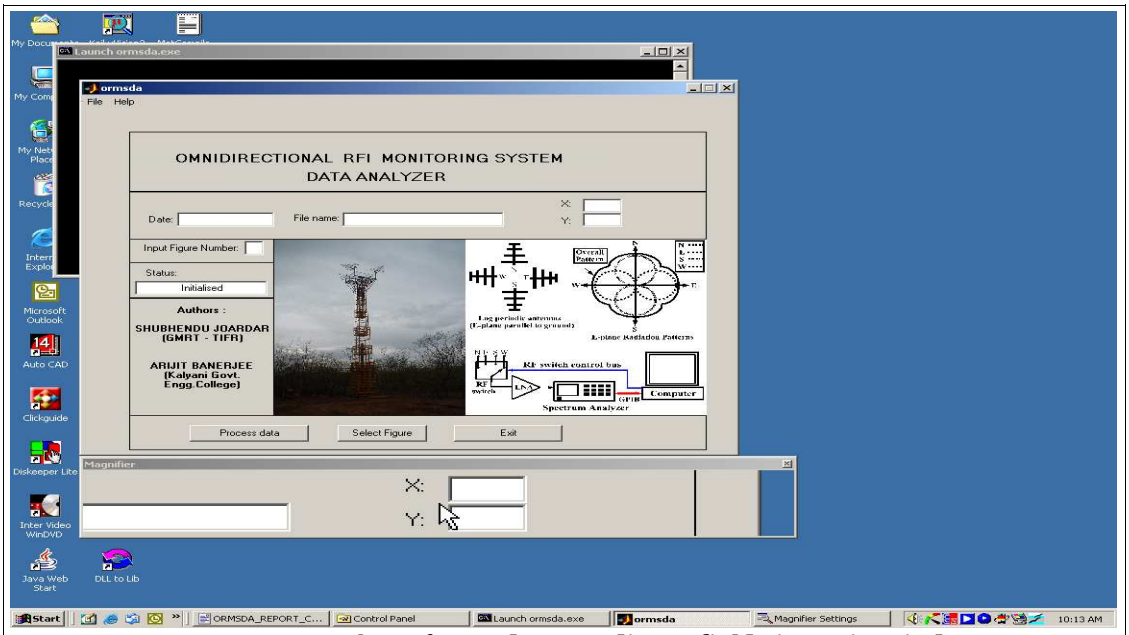



Fig.54 : Screen shot of ***X and Y coordinate fields in main window***

f) The highlighted component shown in the ***magnifier*** window is the ***Status*** field of the application. It shows the initialized, processing and completed states. See Fig.55.

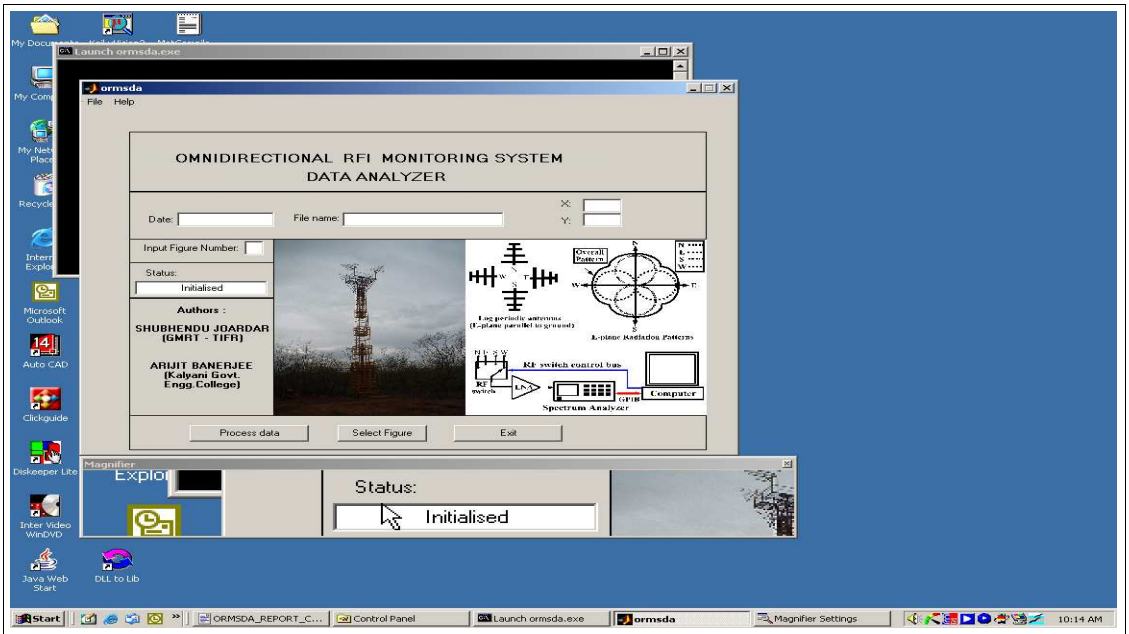



Fig.55 : Screen shot of ***Status field in main window***

g) The highlighted object in the ***magnifier*** window is the push button ***Process Data***. It is an alternative to file menu's open option. See Fig.56.

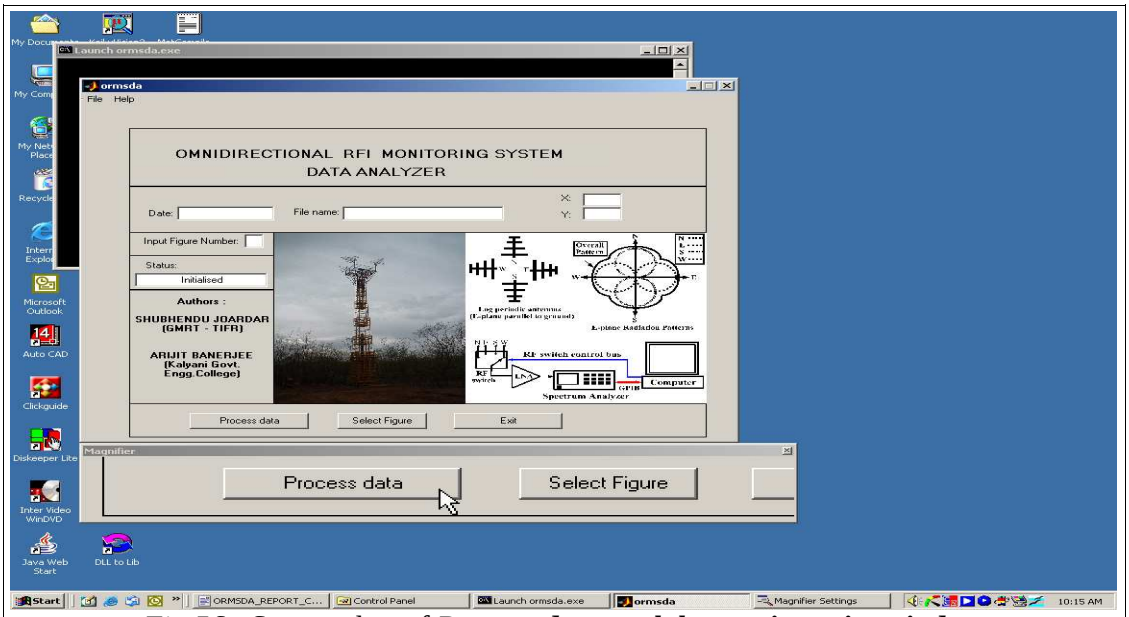



Fig.56 : Screen shot of ***Process data push button in main window***

h) The ***magnifier*** window shows the highlighted object named ***Input Figure Number***, which is for inputting a digit between 1 to 4. See Fig.57.

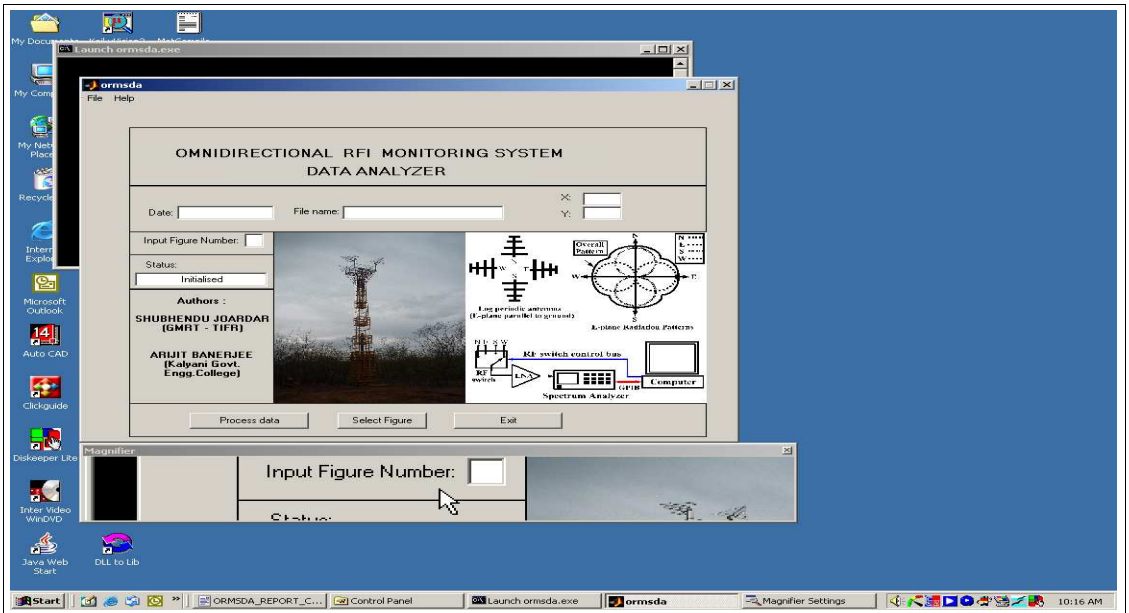



Fig.57 : Screen shot of ***Input Figure Number field in main window***

i) The ***magnifier*** window shows the push button named ***Select Figure*** in Fig.58. After inputting a valid figure in the input figure number object, clicking the select figure button selects the valid figure.

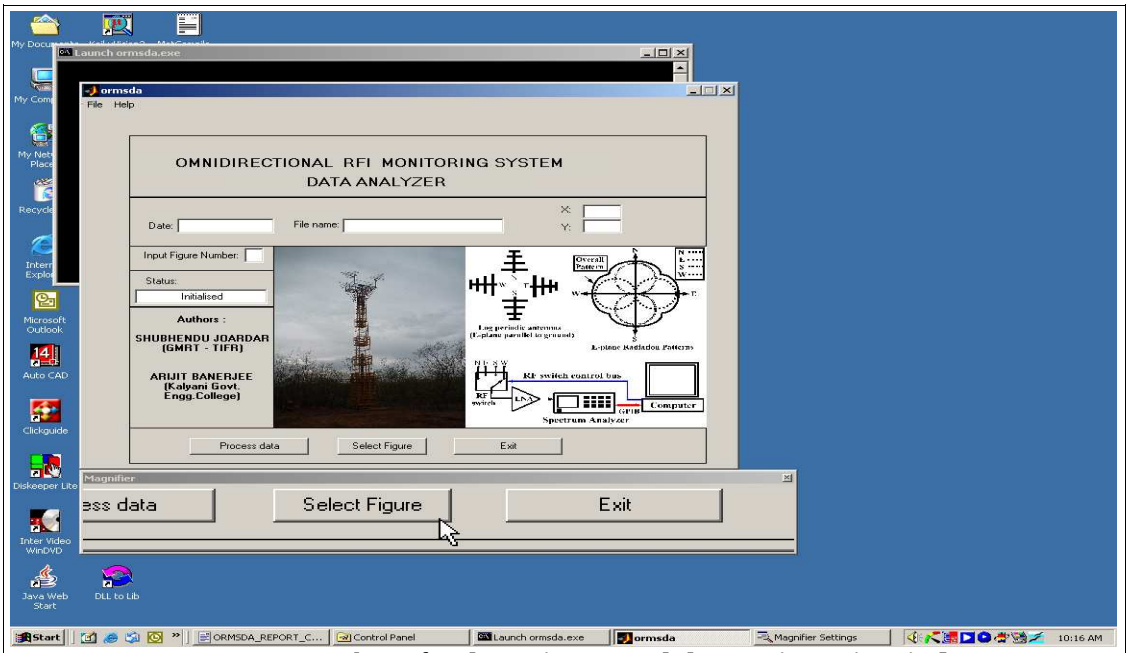



Fig.58 : Screen shot of ***Select Figure push button in main window***

j) The ***magnifier*** window highlights the push button named ***Exit***. To quit this application, press it and follow the instructions. See Fig.59.

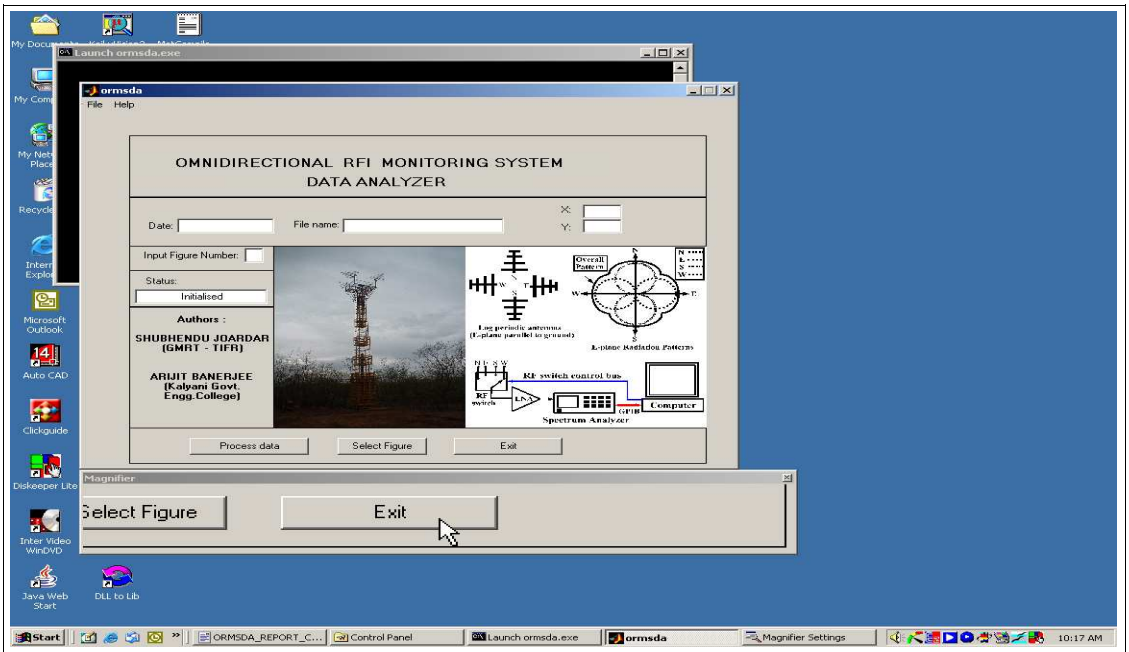



Fig.59 : Screen shot of ***Exit push button in main window***

# 4) Opening a **AVG.rfi* file:

To open a file for processing, just follow the instructions below:-

a) Go to the file menu. Select the sub menu named ***Open*** and click it as shown in the screen shot in Fig.60 below.

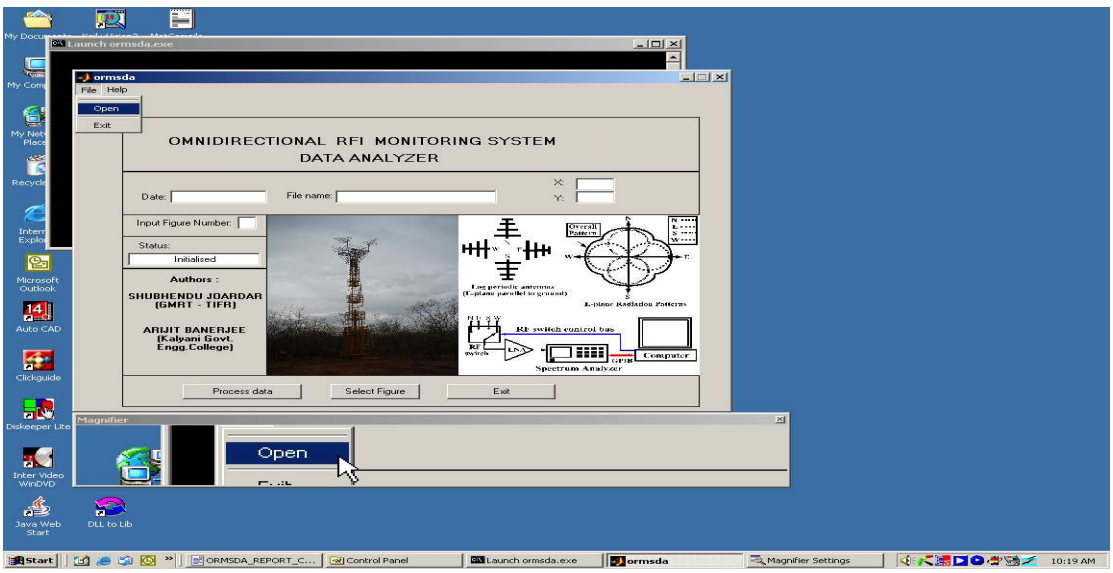



Fig.60 : Screen shot of ***Opening a file through Open sub menu***

b) After clicking the ***Open*** sub menu a new window will appear as shown in the screen shot. It has an embedded filter of ****AVG.rfi***. See Fig.61.

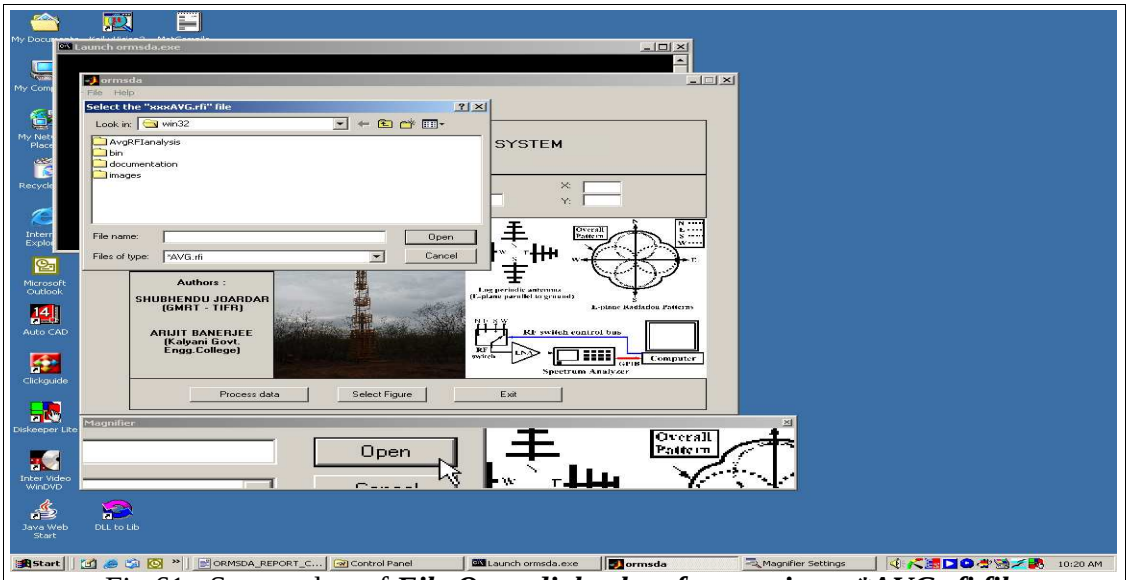



Fig.61 : Screen shot of ***File Open dialog box for opening a *AVG.rfi file***

c) Now specify the path of the input file and click the ***Open*** push button. If you don't wish to continue, click ***Cancel***. See Fig.62.

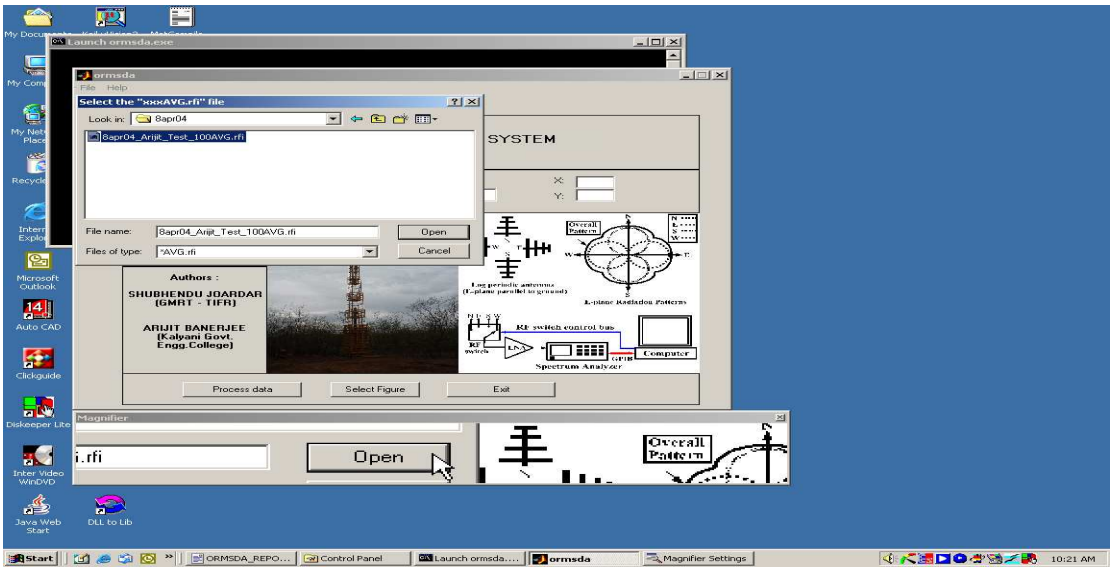



Fig.62 : Screen shot of ***Specifying the input file***

# 5) Settings of input parameters:

a) After opening a file, an input system parameters' dialog box appears. Here the ***magnifier*** window shows the ***ST4W Switch Insertion Loss (dB)***. It is seen that the default value is 2.0dB. See Fig.63. To change it, just click the mouse and edit the setting.

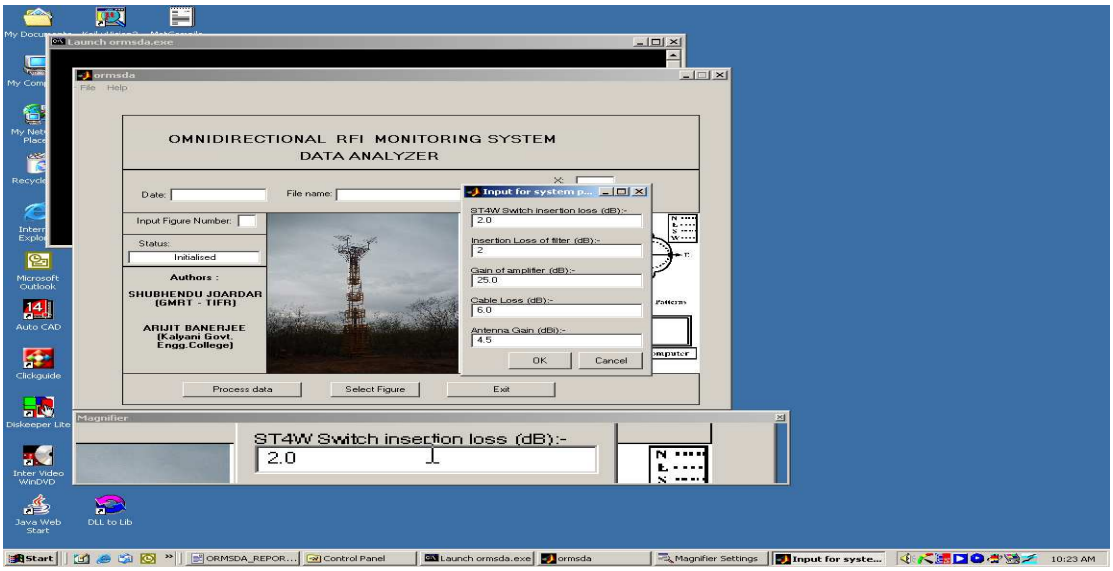



Fig.63 : Screen shot of ***ST4W Switch insertion loss (dB) input field***

b) The ***magnifier*** window shows the highlighted object input text named ***Insertion Loss of filter (dB)***. See Fig.64. It has a default value of 2dB. If required change the setting.

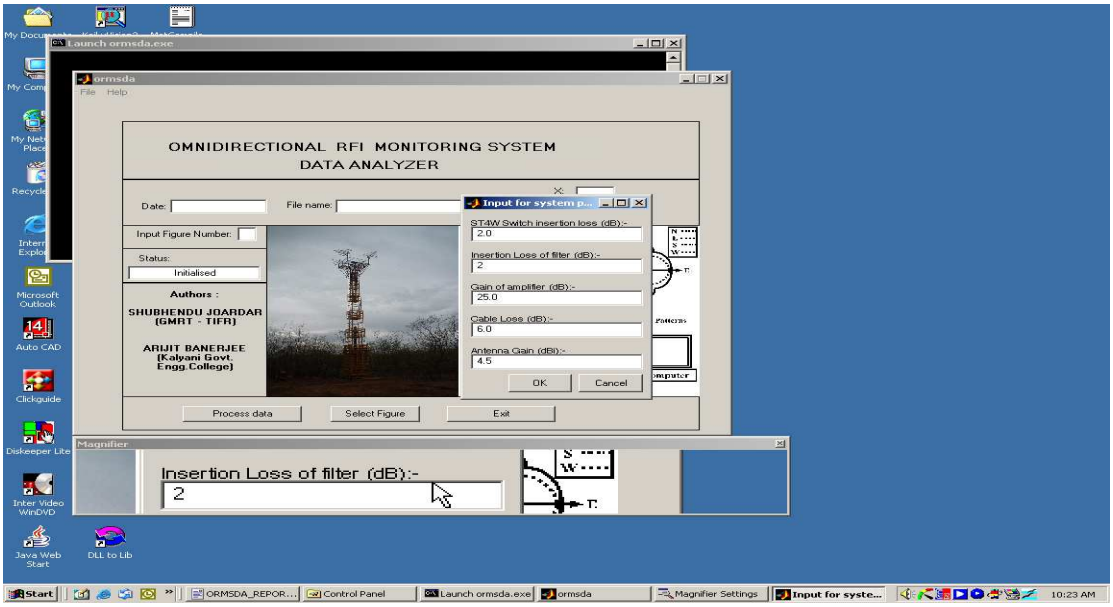



Fig.64 : Screen shot of ***Insertion Loss of filter (dB) input field***

c) The highlighted object is ***Gain of amplifier (dB)*** in the ***magnifier*** window. It has a default value of 25.0dB. Change it as per requirements. See Fig.65.

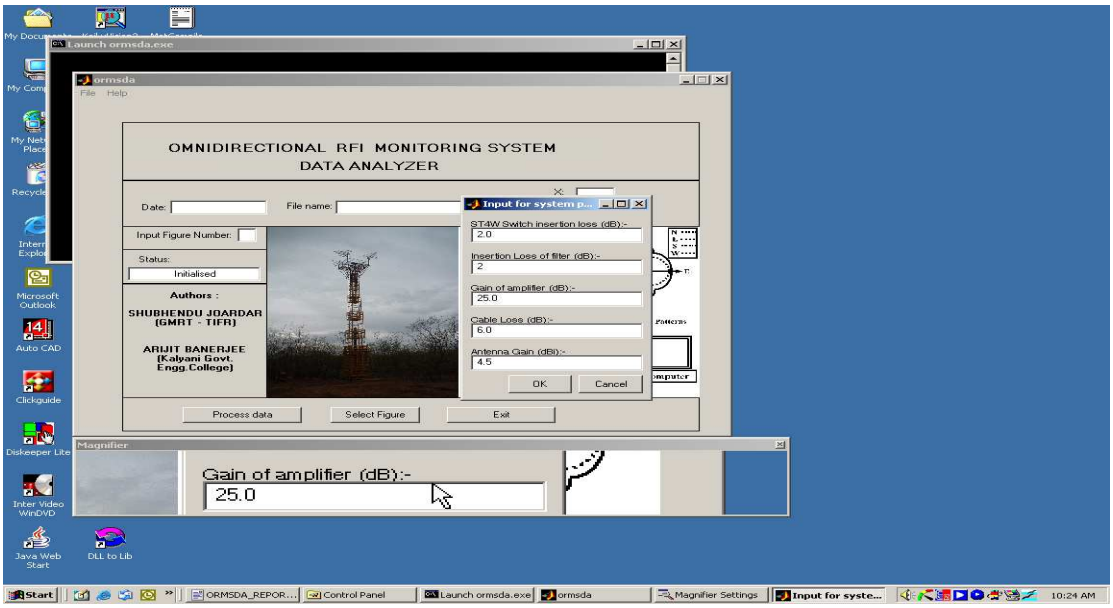



Fig.65 : Screen shot of ***Gain of amplifier (dB) input field***

d) The screen shot shows the output text object ***Cable Loss (dB)***. It has a default value of

6.0dB. You can change it as required. See Fig.66.

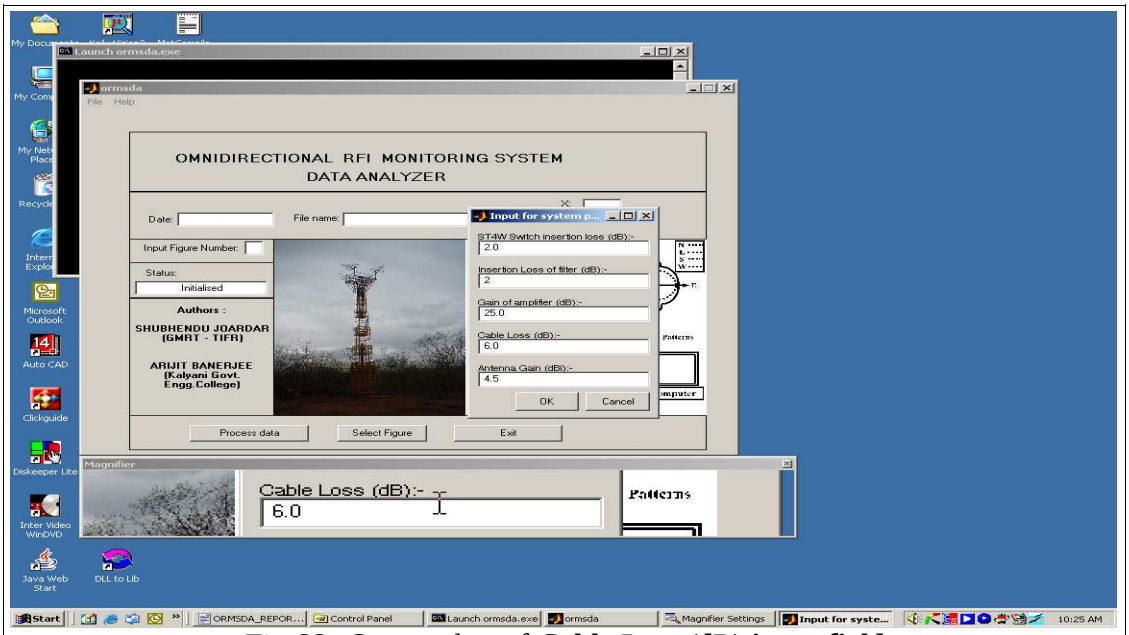



Fig.66 : Screen shot of ***Cable Loss (dB) input field***

e) The last input setting is highlighted in the ***magnifier*** window as ***Antenna Gain (dBi)***. It has a default value of 4.5dBi. User can modify it as per requirements. See Fig.67.

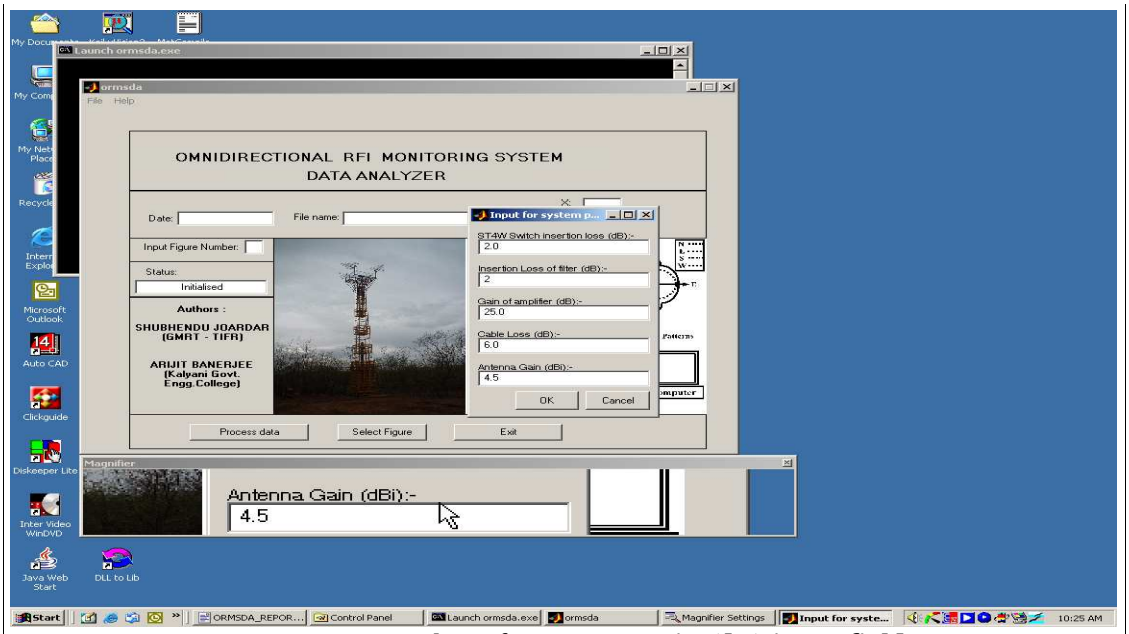



Fig.67 : Screen shot of ***Antenna Gain (dB) input field***

f) After setting the all apt values of the input dialog box, click ***OK*** to continue or ***Cancel*** to abort the processing. See Fig.68.

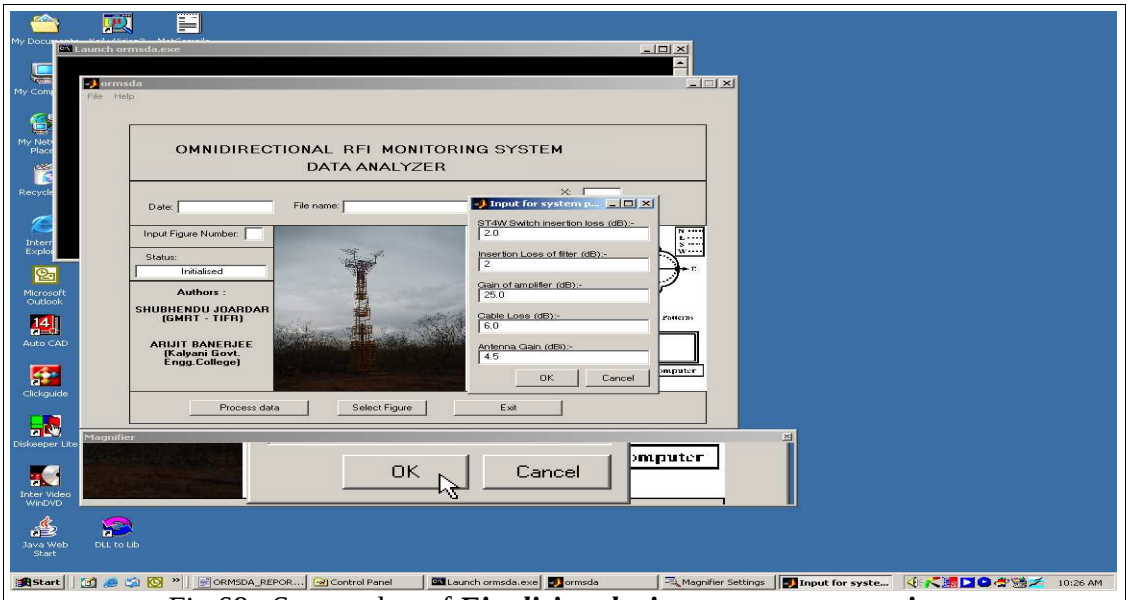



Fig.68 : Screen shot of ***Finalizing the input parameter settings***

# 6) Analyzing the data:

a) After clicking the input setting dialog box to ok, a new message dialog window will come to give you a waiting message. Mean while the message window will be gone and a wait bar will appear messaging ***Processing data! Please wait...***. Please wait until the processing go completed and the status field shows ***Processing Completed***. See Fig.69.

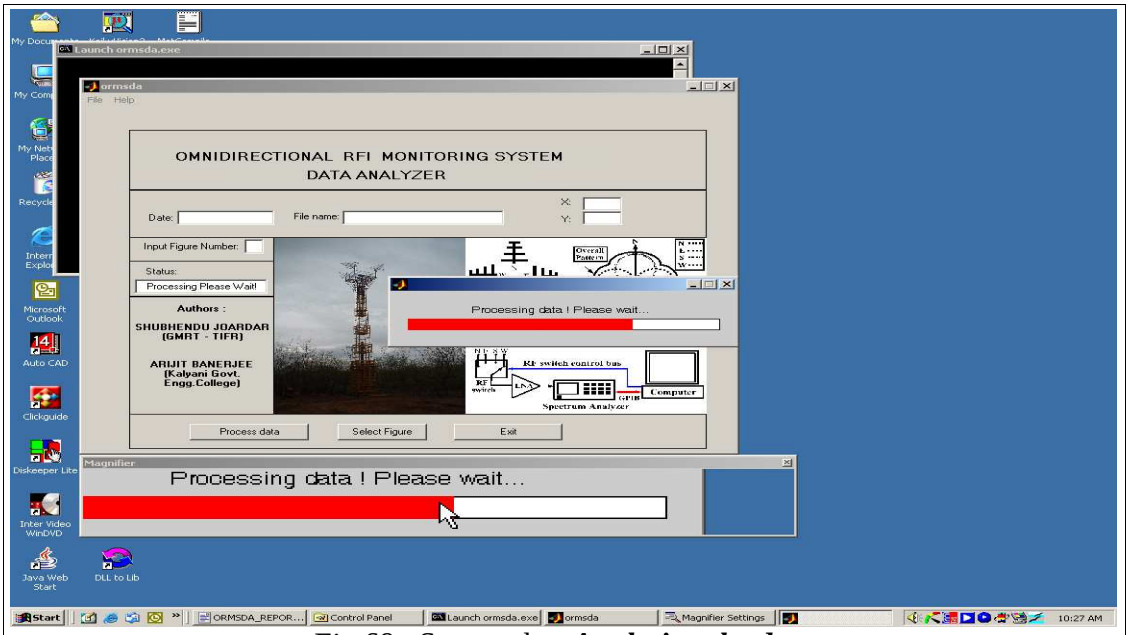



Fig.69 : Screen shot ***Analyzing the data***

# 7) Generating the figures:

a) The application generates four figures named ***figure 1, figure 2, figure 3, and figure 4*** automatically. ***Figure 1*** is a ***Statistics plot***, ***figure 2*** is a ***Contour plot***, ***figure 3*** is a ***three dimensional plot*** and ***figure 4*** is an ***Average power plot***. See Fig.70.

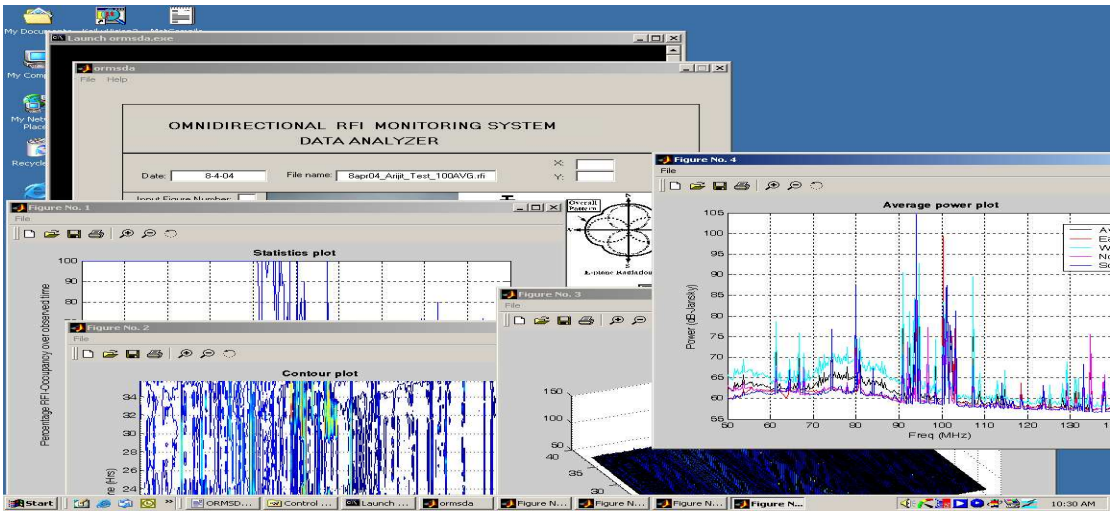



Fig.70 : Screen shot of ***Generating figures***

# 8) Automated saving of figures:

a) The application saves the figures automatically. For verification open ***[install directory]\bin\win32***. You will find a folder named ***AvgRFIanalysis***. Just double click it to open. See Fig.71.

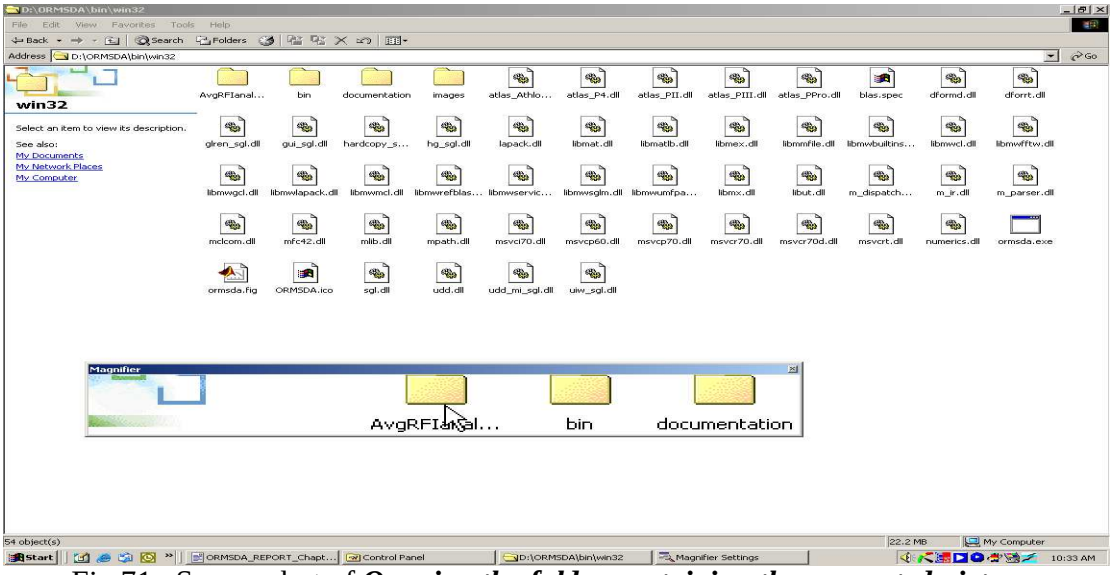



Fig.71 : Screen shot of ***Opening the folder containing the generated pictures***

b) You will find that there are two folders; - one is named ***images*** and another ***xyzdata***. See Fig.72.

c) Double click the images folder.

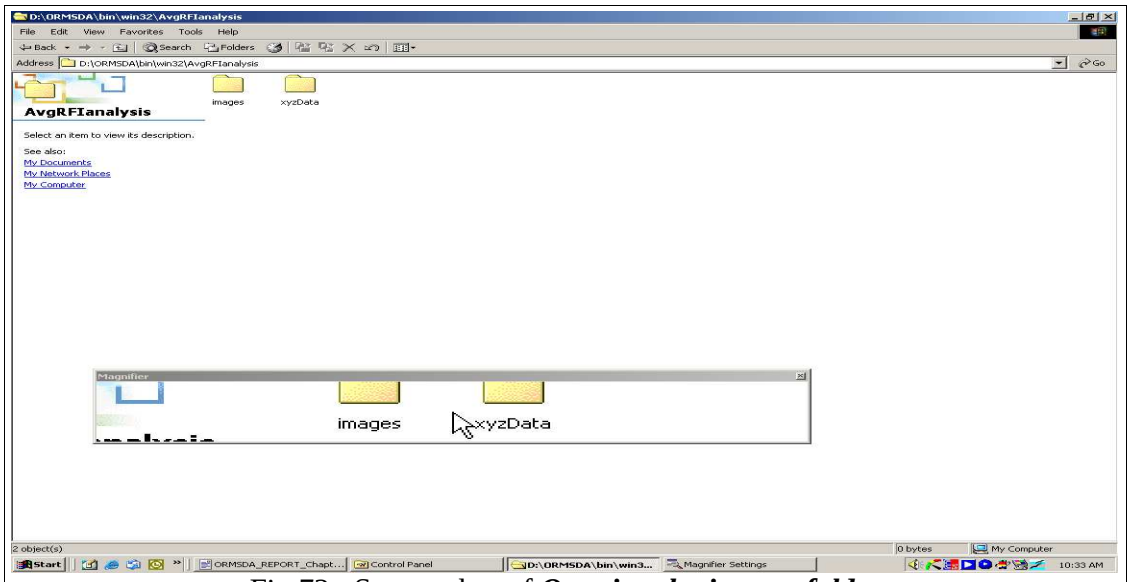



Fig.72 : Screen shot of ***Opening the images folder***

d) You will find that in the images folder there exists another folder with the same name of the input file name except extension name.

e) Double click it to open. See Fig.73.

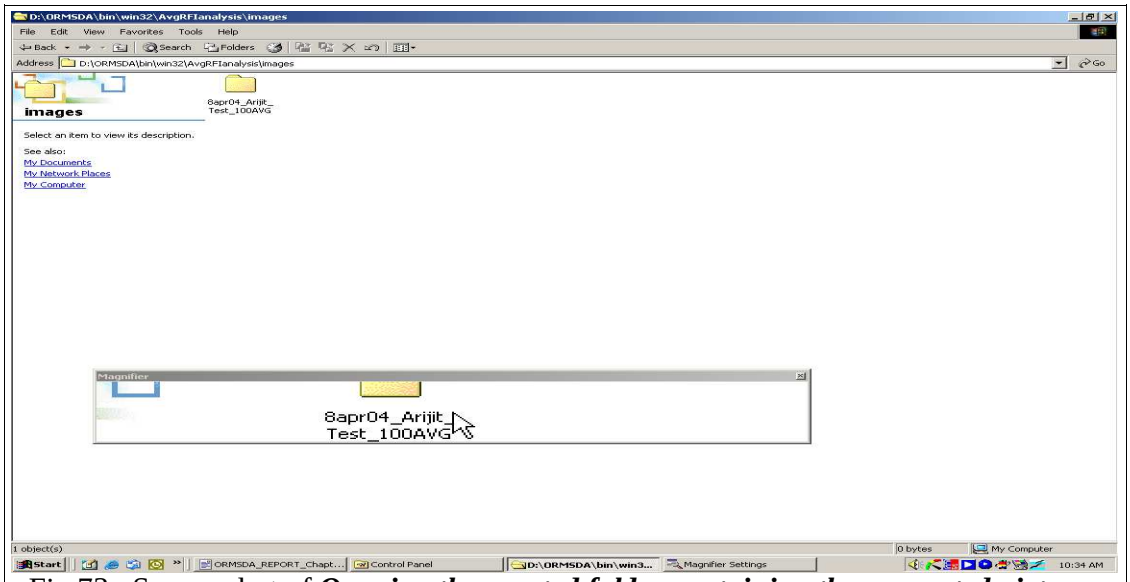



Fig.73 : Screen shot of ***Opening the created folder containing the generated pictures***

f) You will find eight picture files, four of them are of the jpg or jpeg format and other fours are of esp. format. See Fig.74. These are the same files which are generated in the four figures.

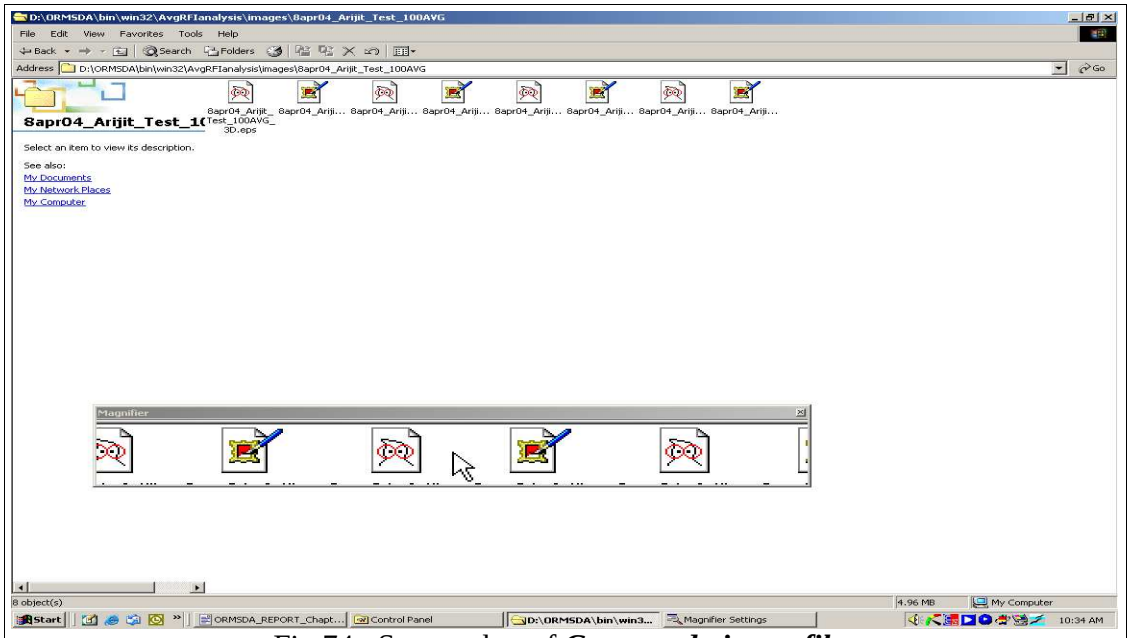



Fig.74 : Screen shot of ***Generated picture files***

g) If you open the ***xyzdata*** folder, you will find the files in processed data format like shown below in the screen shot in Fig.75.

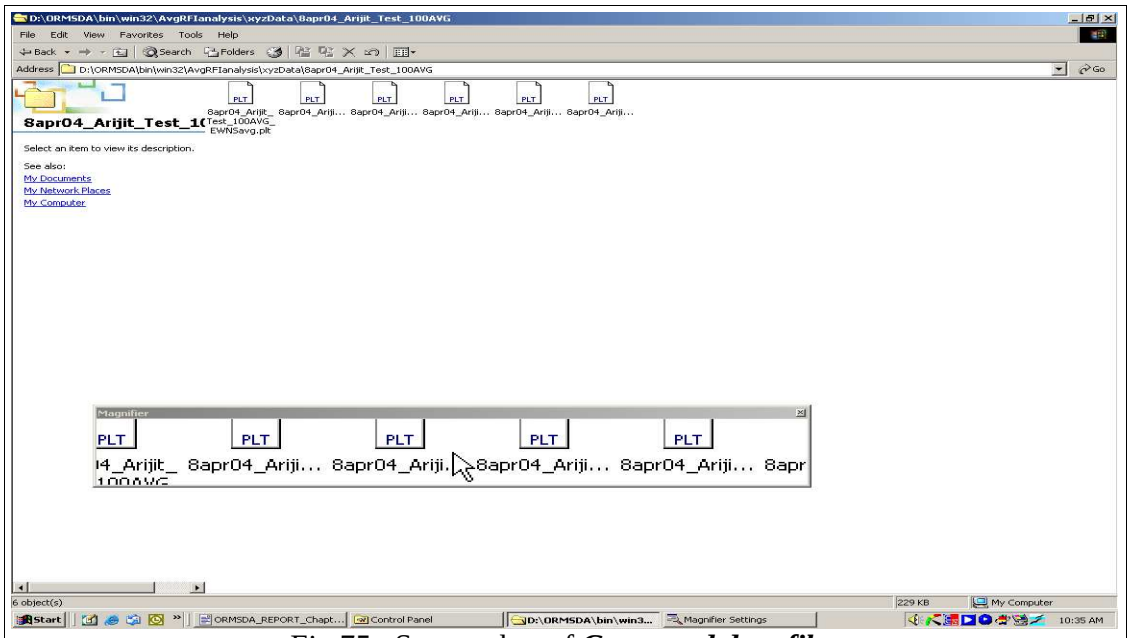



Fig.75 : Screen shot of ***Generated data files***

# 9) Analyzing the figures:

a) To analyze the figures, just input the figure number you want to select which is from 1 to 4. You must specify the figure number in the ***Input Figure Number*** text object. See Fig.76.

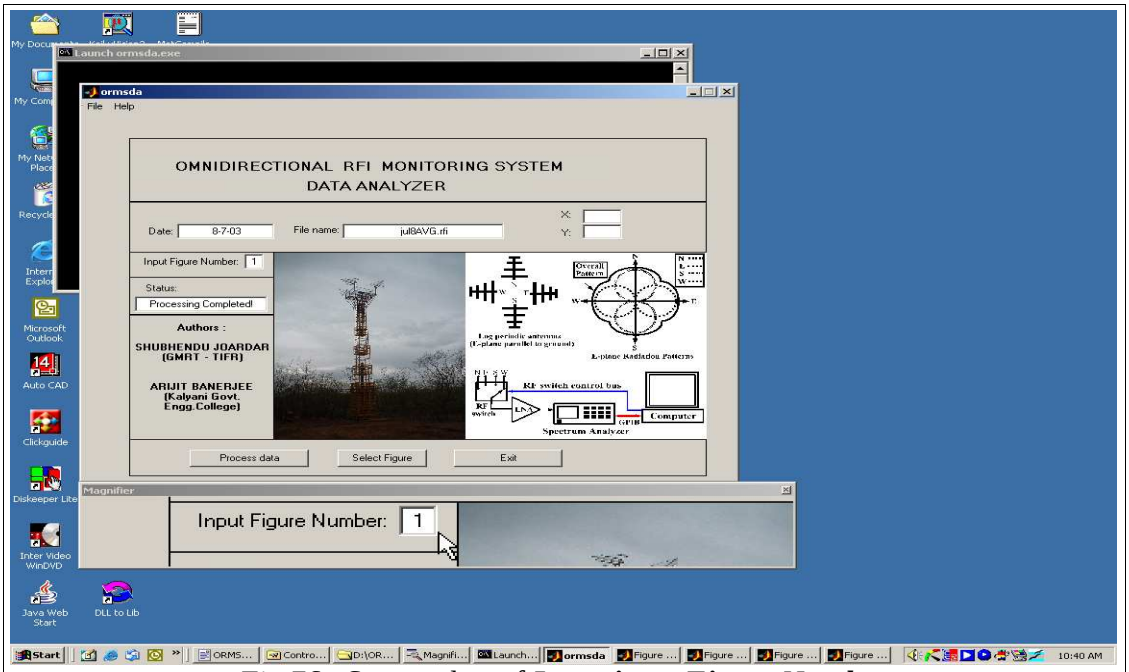



Fig.76 : Screen shot of ***Inputting a Figure Number***

b) Now click the push button named ***Select Figure***. See Fig.77.

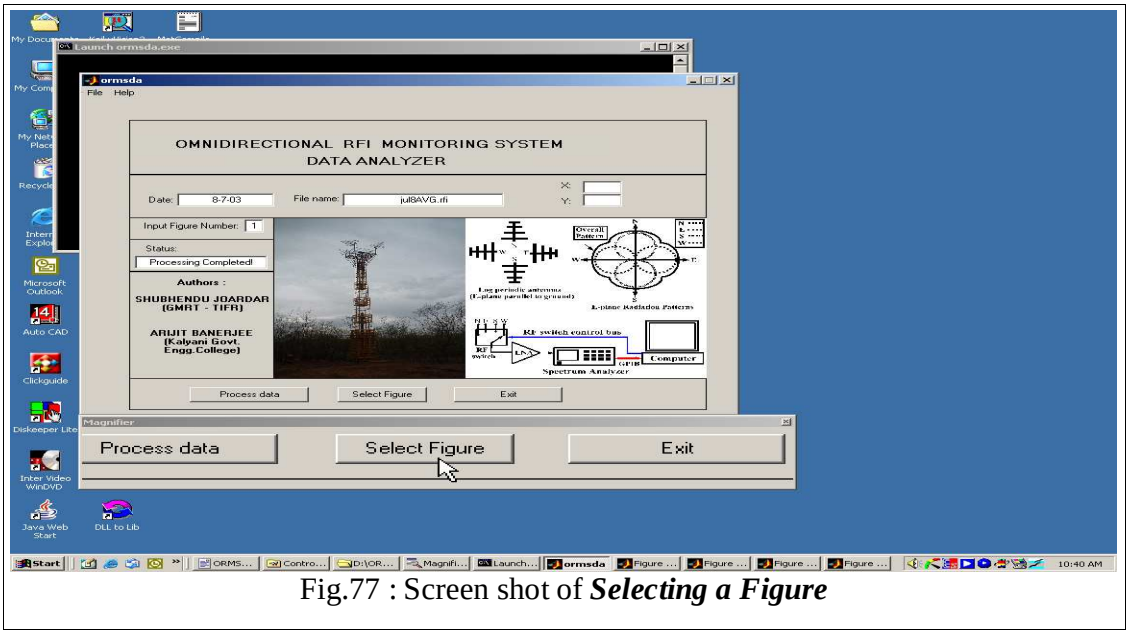



Fig.77 : Screen shot of ***Selecting a Figure***

c) The specified figure window will be selected and displayed along with a big cross hair over it. The cross hair is highlighted in the ***magnifier*** window as shown in Fig.78.

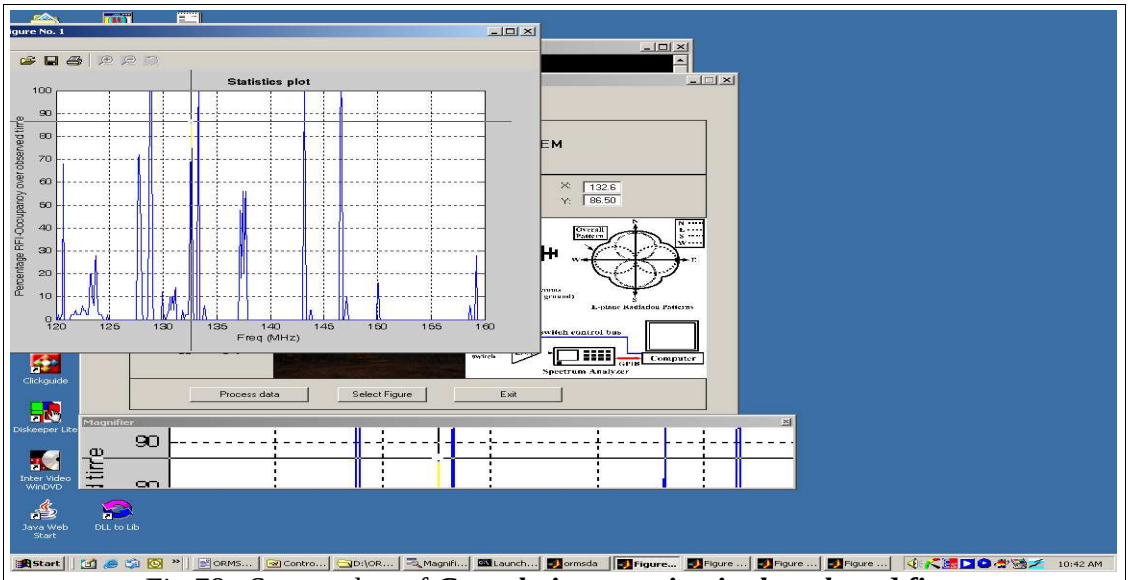



Fig.78 : Screen shot of ***Cross hair appearing in the selected figure***

d) Now you can view the x and y coordinates of the selected figure.

e) Just left click on the point of which you want to know the coordinate. The x and y coordinates will be displayed on the outputs ***X:*** and ***Y:*** fields. See Fig.79.

f) For deselecting the figure just right click the mouse on the same figure. Another way to deselect the figure is to press the ***Esc*** key of the key board.

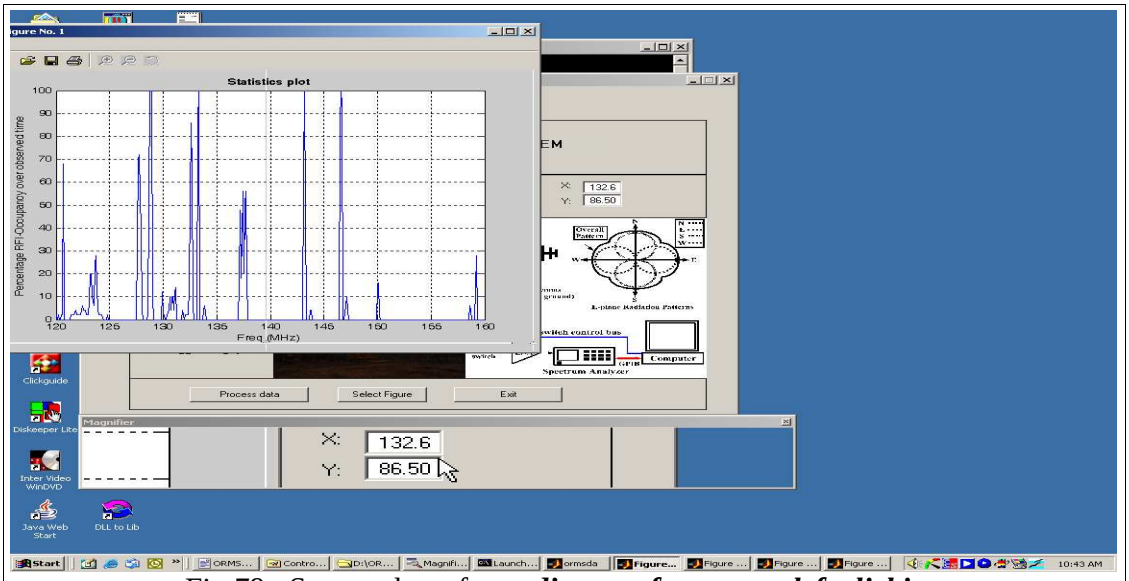



Fig.79 : Screen shot of ***coordinates after mouse left clicking***

g) You can again select another figure or the previous one. See Fig.80.

h) To select another figure input a different figure number and press the select figure button. It will select a new figure and the so called big cross hair will appear on it.

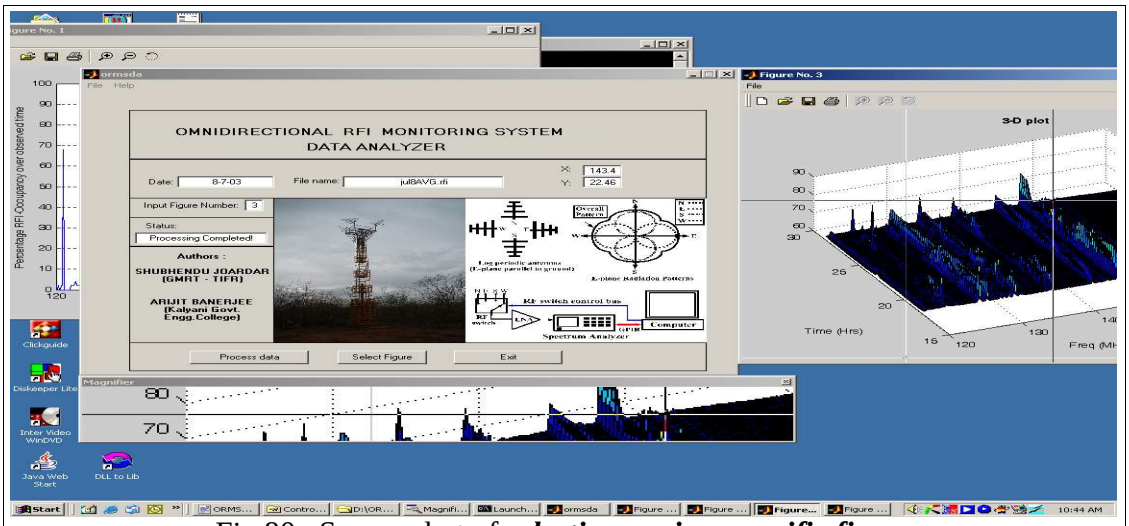


Fig.80 : Screen shot of ***selecting again a specific figure***

# 10) Printing the figures:

a) To print a figure just click the print button on the figure and the figure will be printed on the network printer or local printer associated with the computer. But make sure that the ***Print*** button is active. See Fig.81.

b) If you are analyzing the figure first disable the cross hair by right clicking or pressing the escape button.

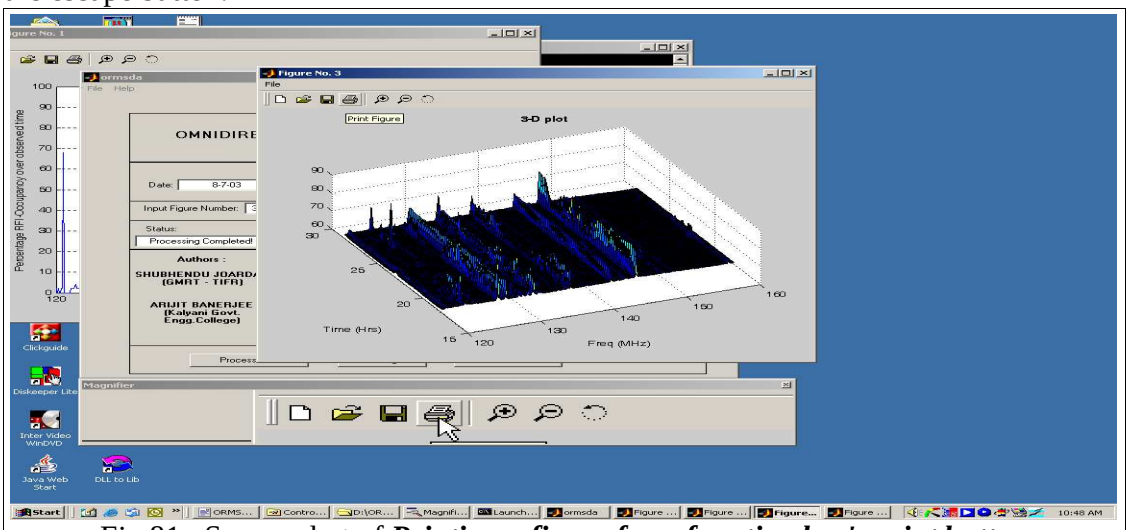
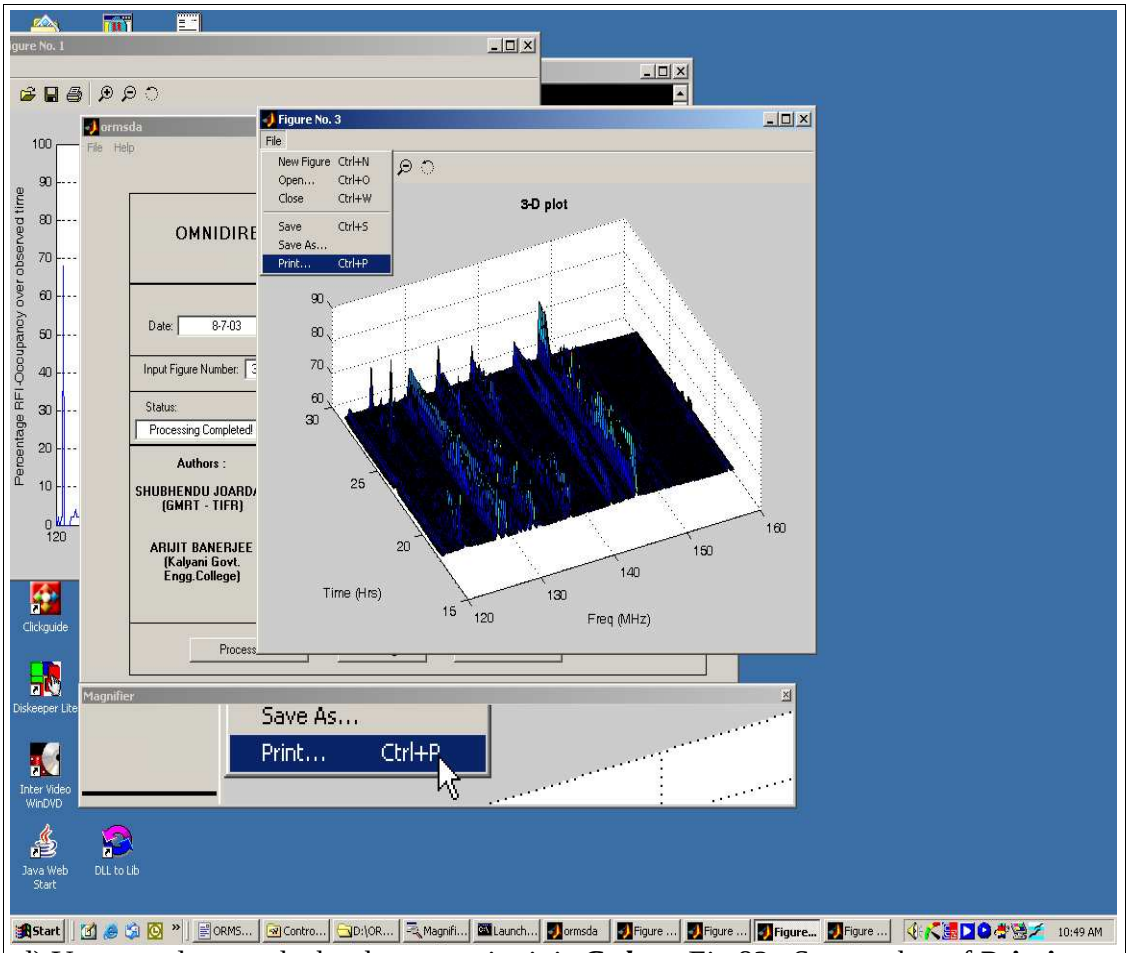


Fig.81 : Screen shot of ***Printing a figure from function bar's print button***

c) You can alternatively print your analyzed figures by going to the print sub menu option of the file menu. See Fig.82.




d) You can also use the hot keys to print it is ***Ctrl + p***.Fig.82 : Screen shot of ***Printing a figure from print sub menu or hot key***

# 11) The command window:

a) The command window, in Fig.83, is the launcher program for ***ormsda.exe*** as well as it creates a link with the system console.

b) Whenever an undefined or defined error occurs, it reacts on it and gives the message output to the command window.

c) If you get a problem with processing and it seems to be that the application hangs, do not try to close the ***ormsda.exe***, but better close the command window that is ***launch ormsd.exe***. It will instantly abort the application.

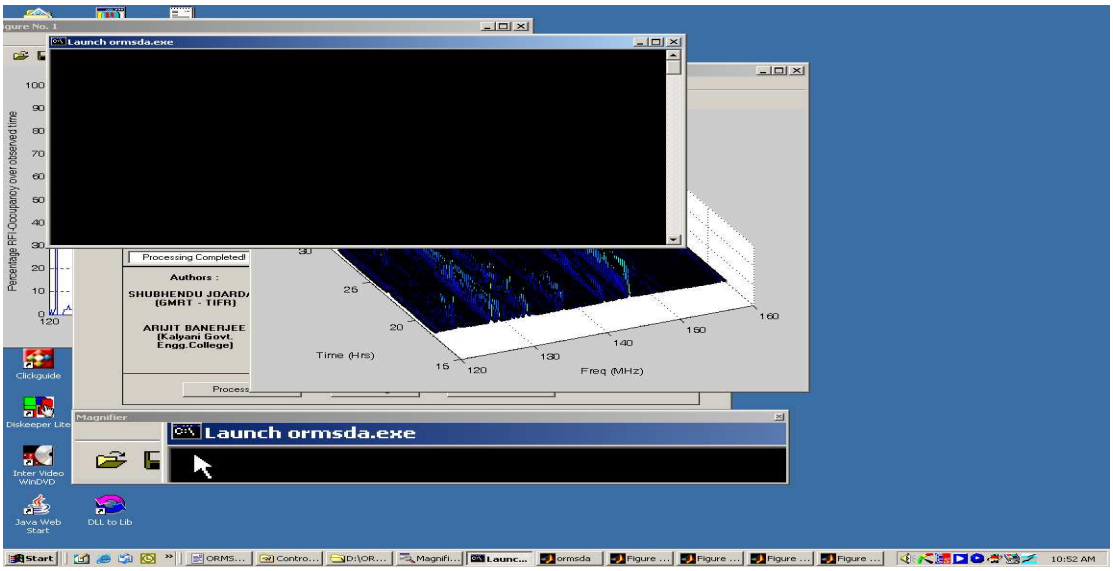



Fig.83 : Screen shot of ***Command window***

# 12) General user errors:

a) Error due to selecting a figure when Status is ***initialized*** and Input Figure Number is null:

Here in the screen shot shown in Fig.84, we see that the status field is ***initialized***, that is no processing is done and the ***input figure number*** is also blank.

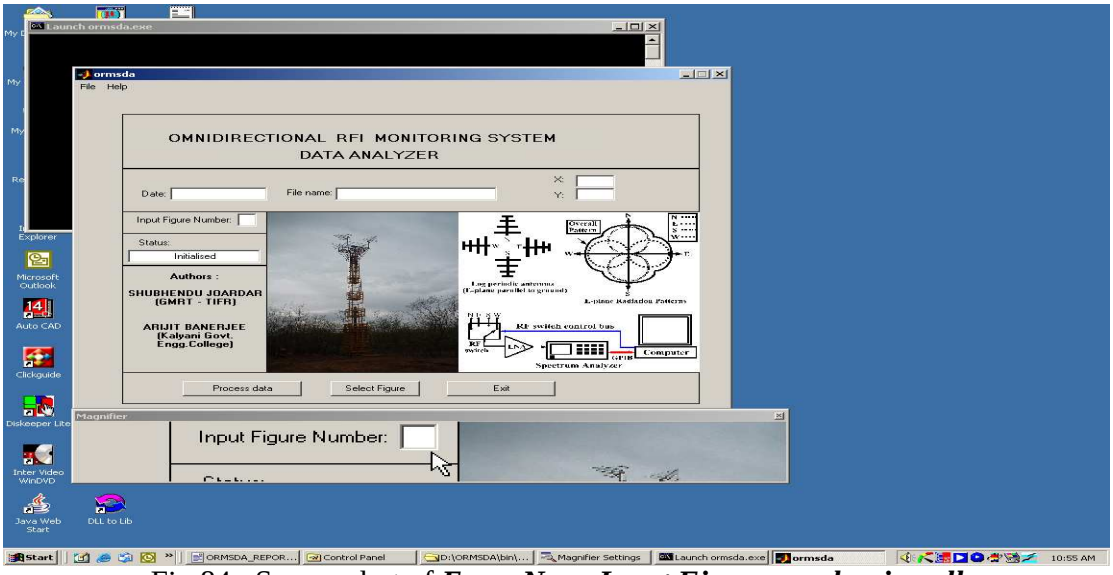



Fig.84 : Screen shot of ***Error No a, Input Figure number is null***

In these circumstances, if the user clicks the ***Select Figure*** button in Fig.85, it will give

an error message.

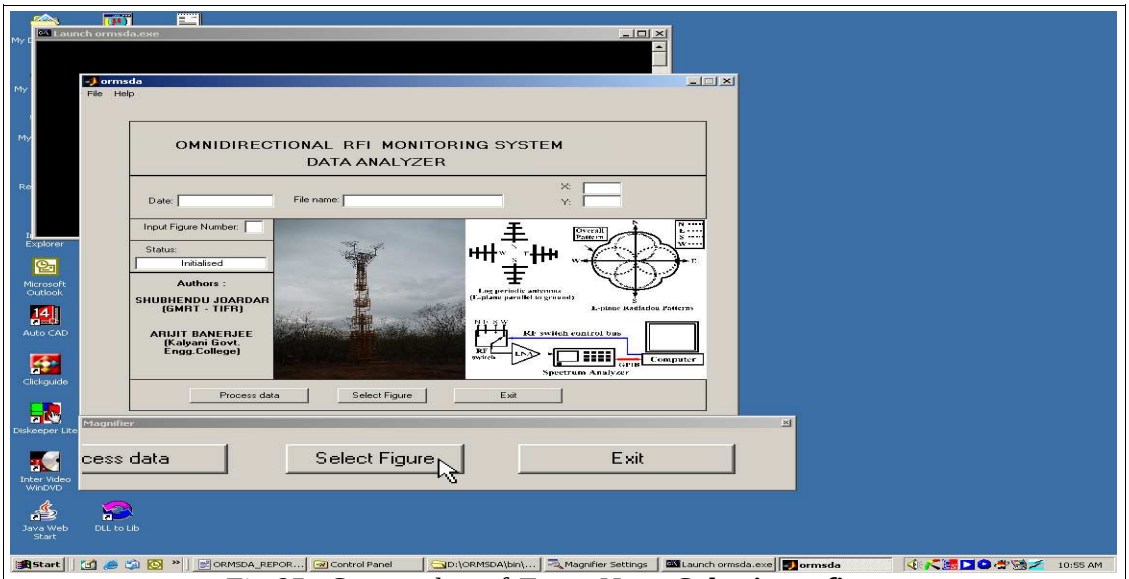



Fig.85 : Screen shot of ***Error No a, Selecting a figure***

The error message dialog is shown in the screen shot in Fig.86 as well as highlighted in the ***magnifier*** window.

**<u>Hence do not select a figure when Status is *initialized* and Input Figure Number is null.</u>**

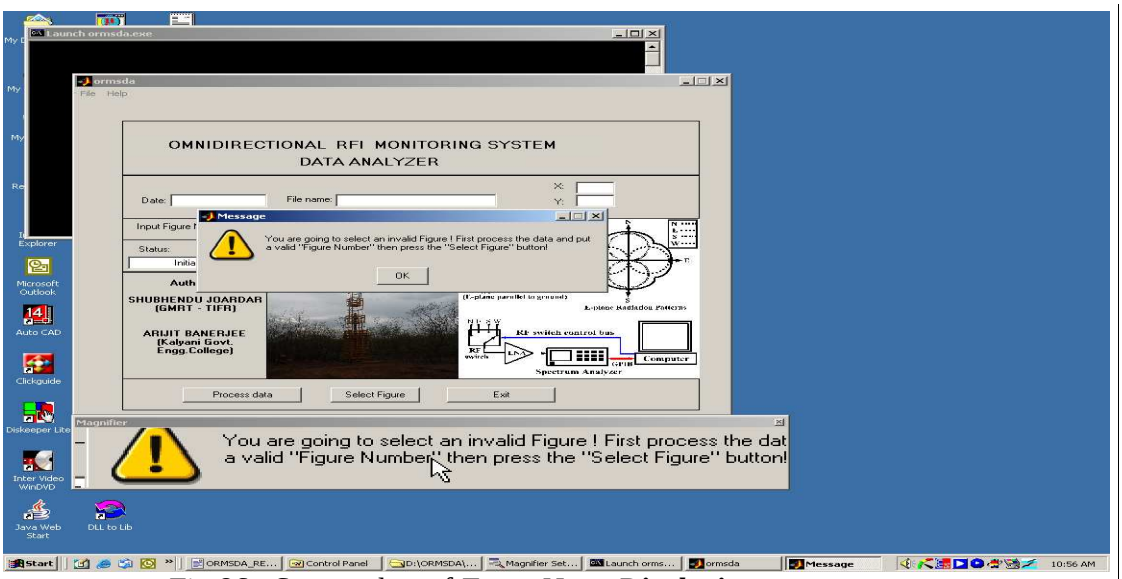



Fig.86 : Screen shot of ***Error No a, Displaying error message***

b) Error due to selecting a figure when Status is ***initialized*** and Input Figure Number is specified:

Here in the screen shot shown in Fig.87, we see that the Status field is ***initialized*** that is the processing is not done yet and the input figure number is specified.

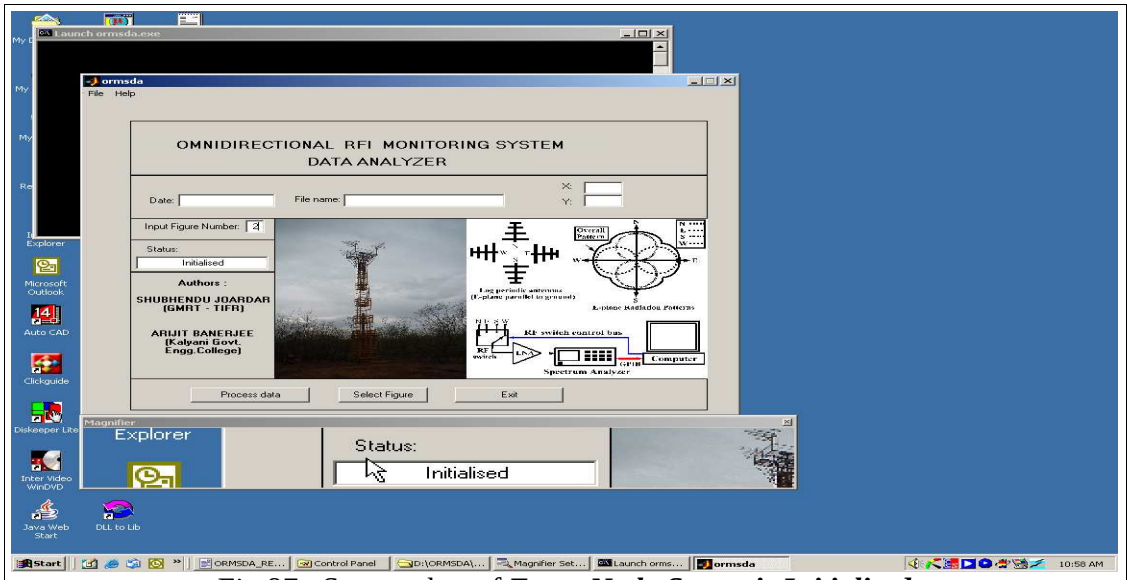



Fig.87 : Screen shot of ***Error No b, Status is Initialized***

In this situation if we click the ***Select Figure*** button as shown in Fig.88, it will cause an error.

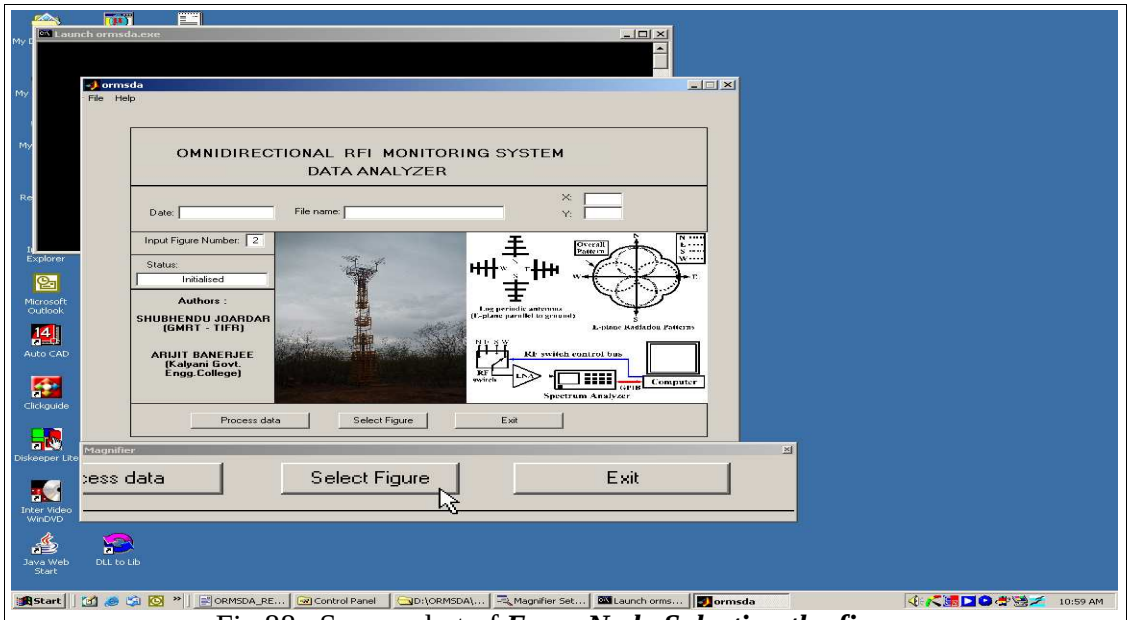



Fig.88 : Screen shot of ***Error No b, Selecting the figure***

The error message dialog box is shown in the screen shot in Fig.89 below.

**Hence do not select a figure when Status is *initialized* and Input Figure Number is specified.**

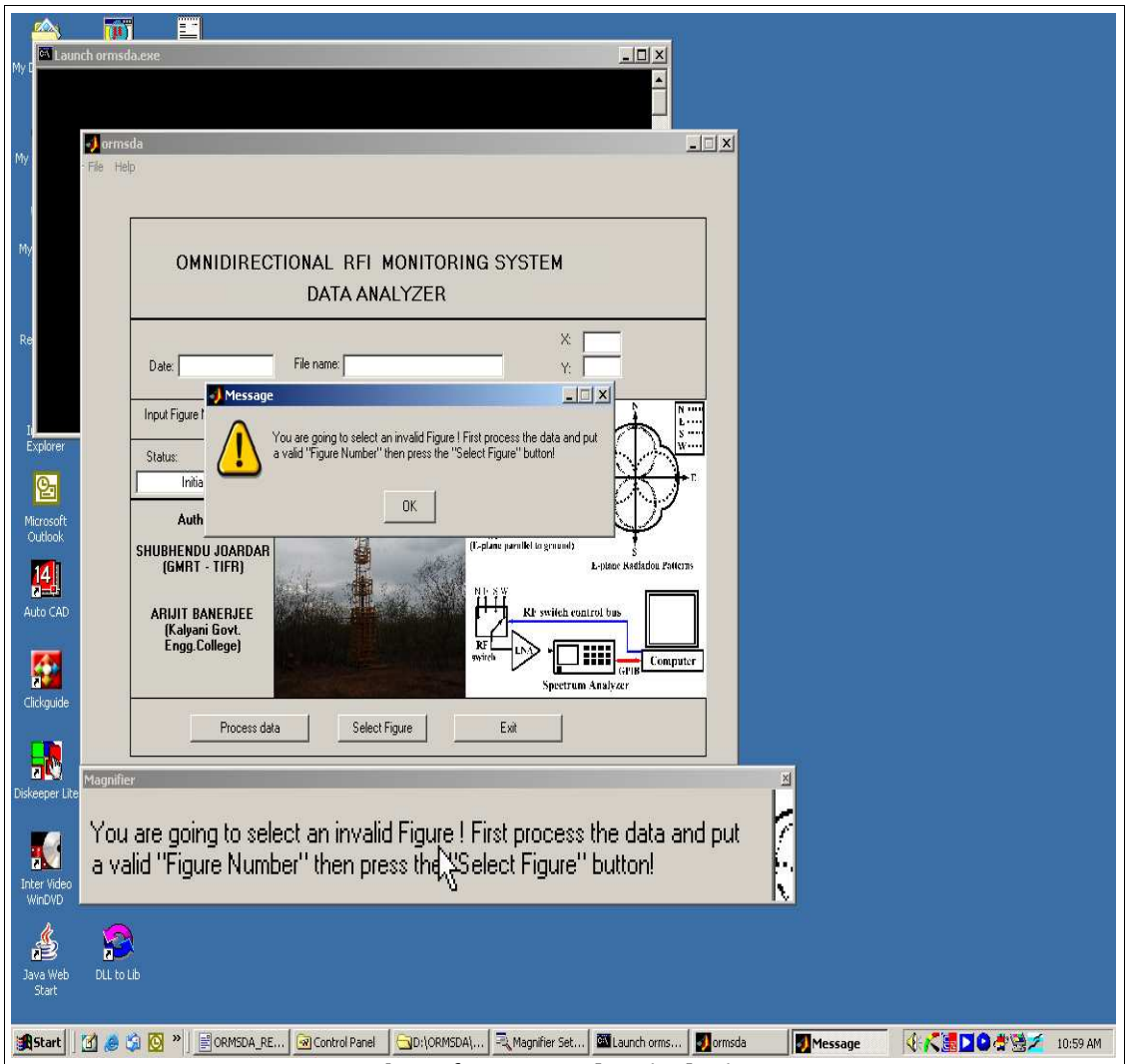


Fig.89 : Screen shot of ***Error No b, Displaying error message***

## c) Error due to selecting a figure when the Status is ***Processing Completed*** and ***Input Figure Number*** is invalid:

In the screen shot shown in Fig.90, we see that the Status field is ***Processing Completed***, that is the processing is done, yet the input figure number is invalid.

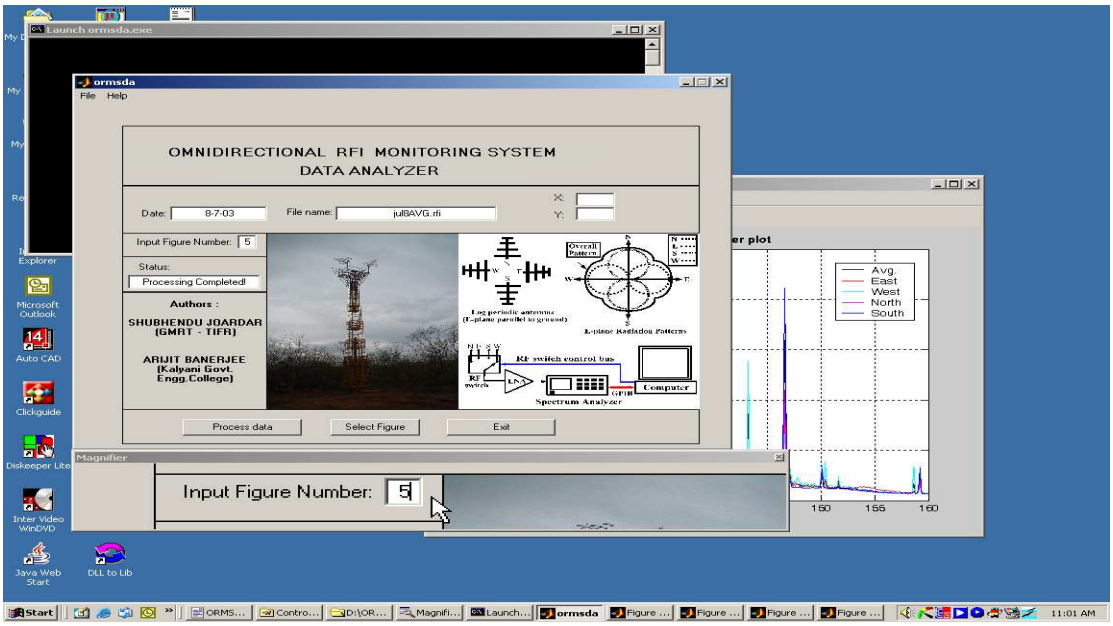



Fig.90 : Screen shot of ***Error No c, Input figure is invalid***

In this situation if we click the ***Select Figure*** button as in Fig.91, an error will occur.

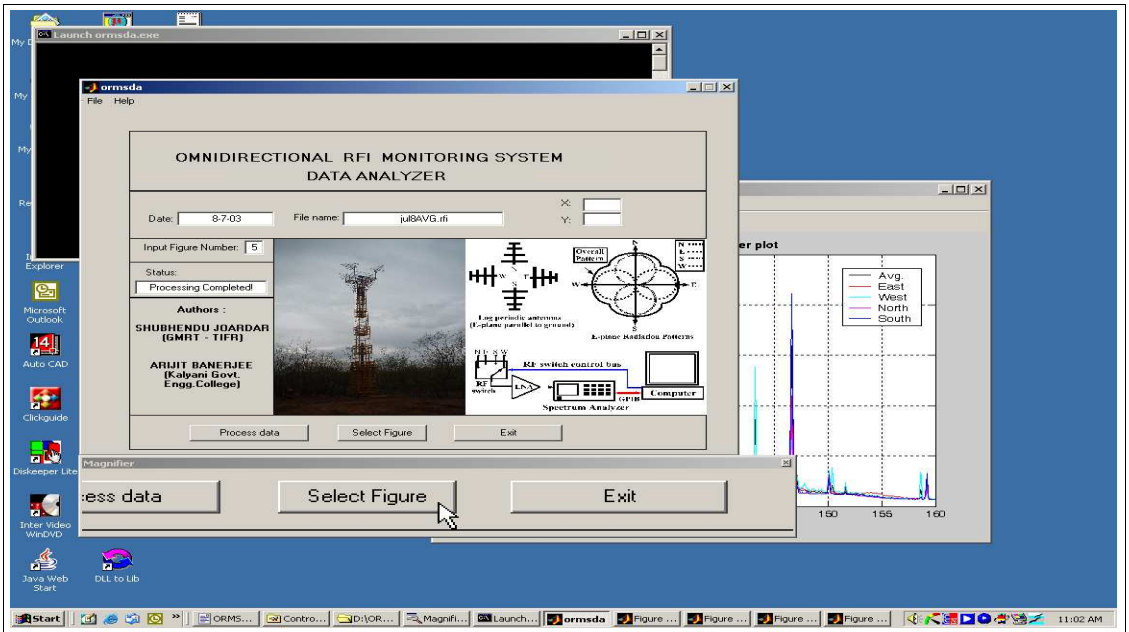



Fig.91 : Screen shot of ***Error No c, Selecting the invalid figure***

The error dialog box will show a warning as shown in the screen shot in Fig.92.

**<u>Hence do not select a figure while the status is *Processing Completed* and the input figure number is invalid.</u>**

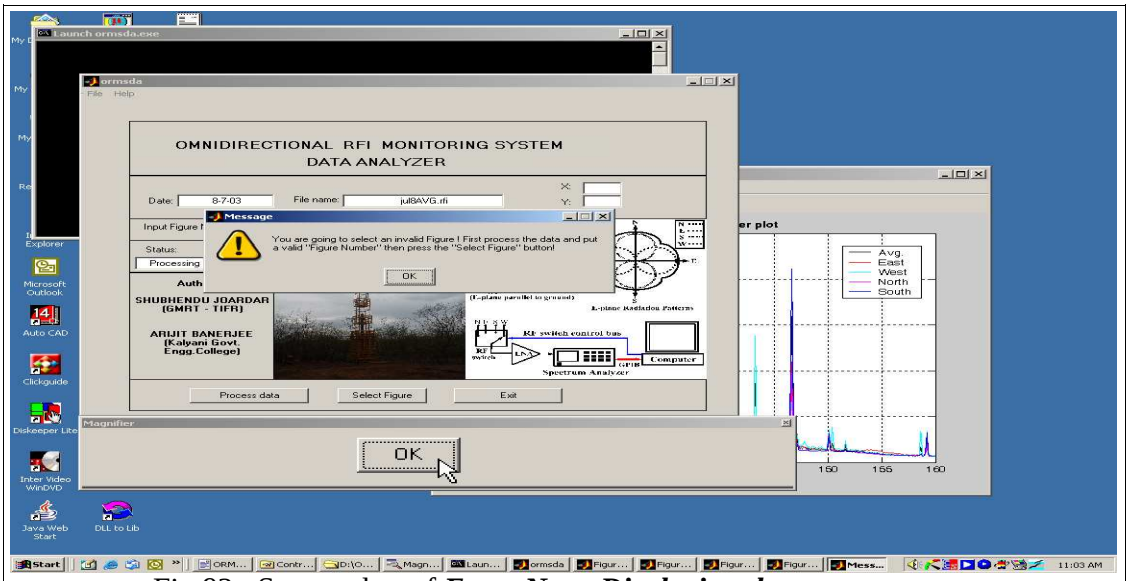



Fig.92 : Screen shot of ***Error No c, Displaying the error message***

## d) Error due to selecting a figure when the Status is ***Processing Completed*** and ***Input Figure Number*** is a character:

Here in the screen shot shown in Fig.93, the Status field is ***Processing Completed***, that is the processing is done, yet the input figure number is a character.

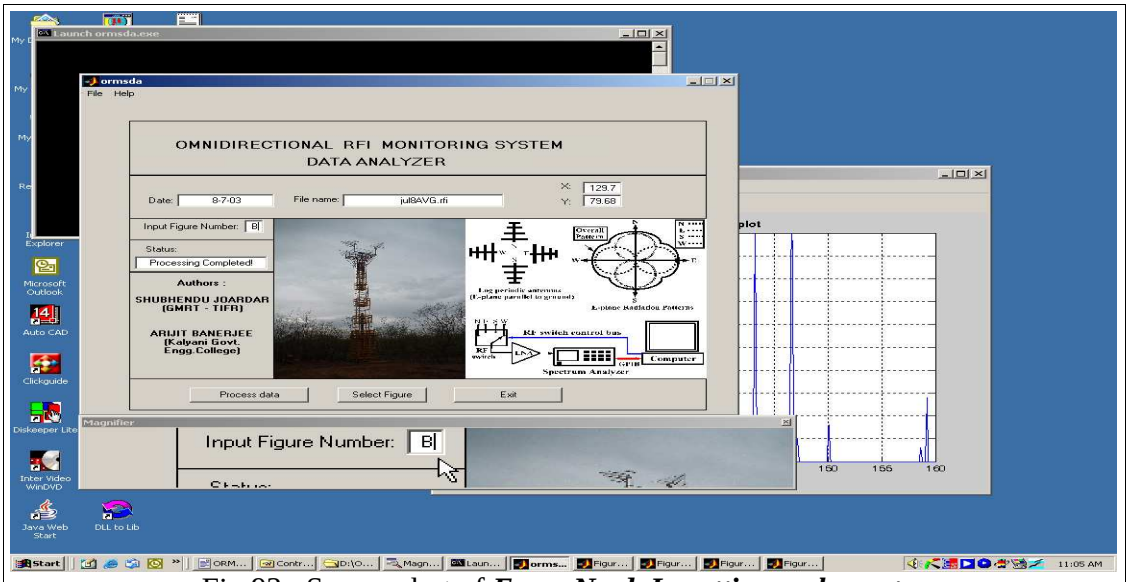



Fig.93 : Screen shot of ***Error No d, Inputting a character***

In this situation, if we click the ***Select Figure*** button, an error will occur. The error dialog box will show a warning as shown in the screen shot in Fig.94.

**Hence do not select a figure while the status is *Processing Completed* and the input figure number is a character.**

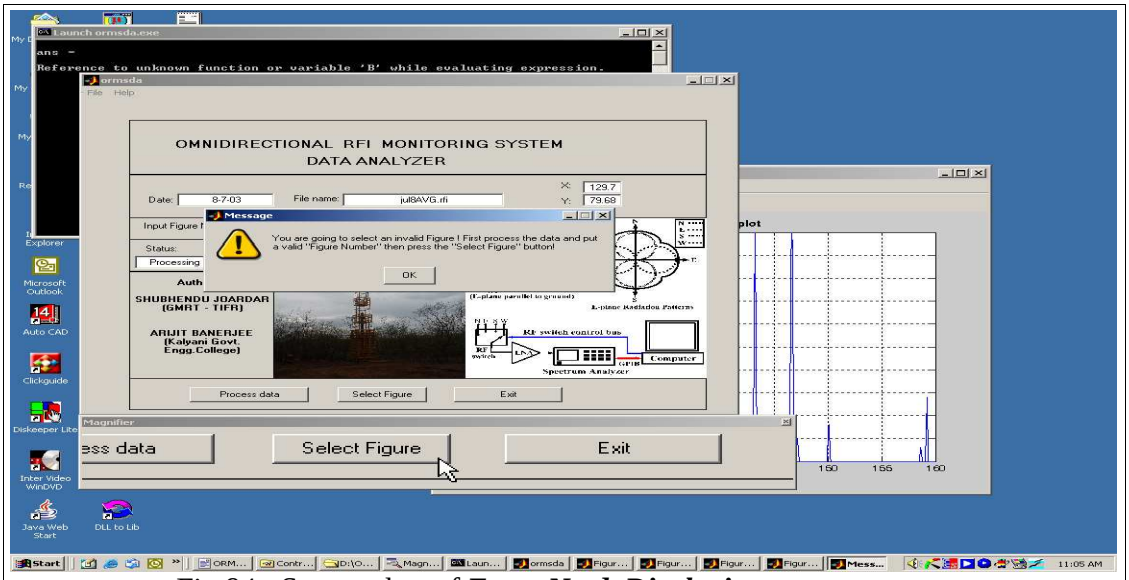


Fig.94 : Screen shot of ***Error No d, Displaying error message***

The character error is not defined in the application, but we can see that the command window gives an error message. See Fig.95.

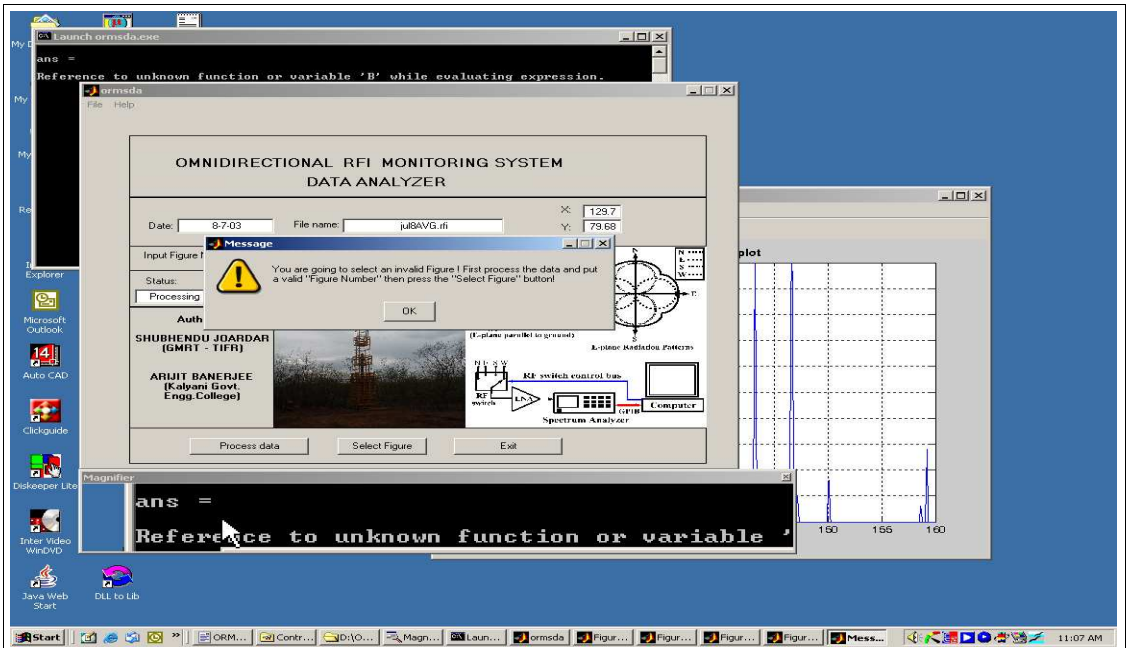


Fig.95 : Screen shot of ***Error No d, Displaying error in command window***

e) Error due to selecting a figure when the Status is ***Processing Completed*** and figures are manually deleted and ***Input Figure Number*** is specified:

Here in the screen shot shown in Fig.96, we see that the Status field is ***Processing Completed*** and figures are manually deleted.

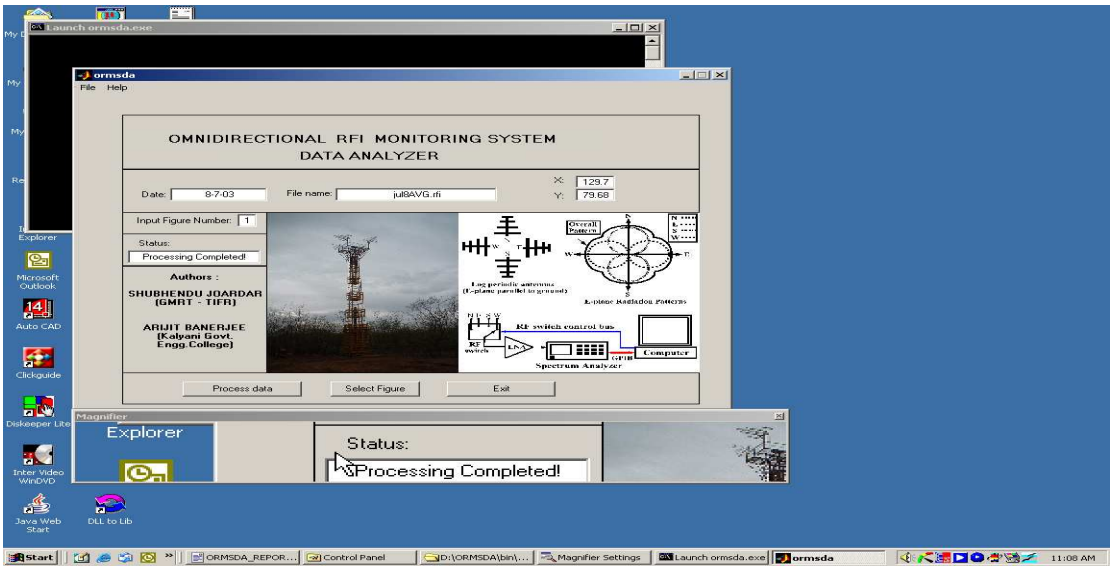



Fig.96 : Screen shot of ***Error No e, Status is Processing Completed***

Here the ***Input Figure Number*** is specified in the input object text field in Fig.97.

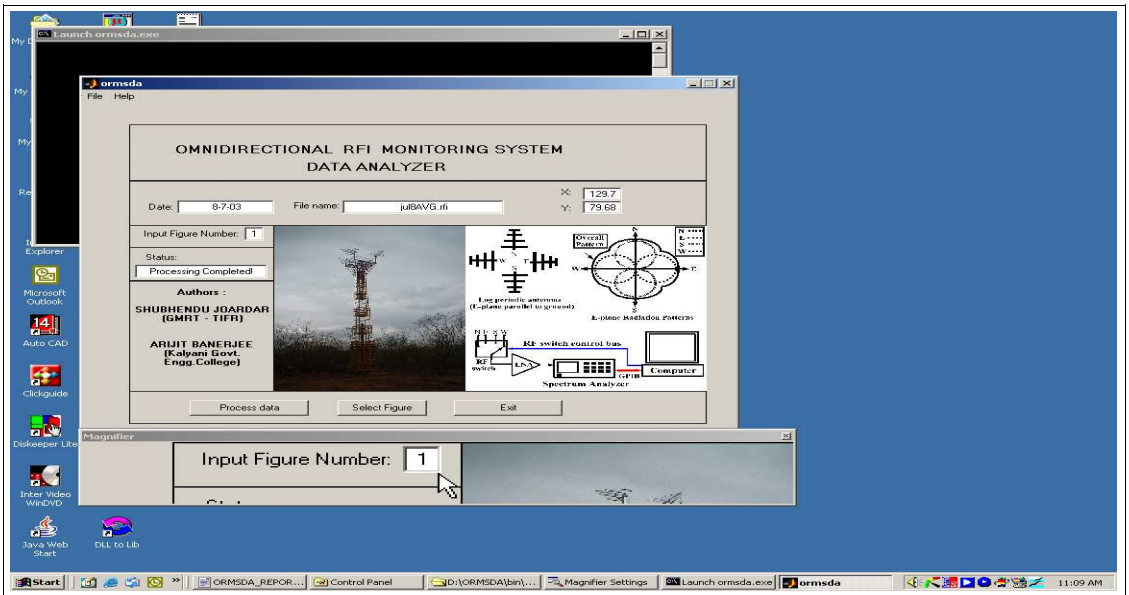



Fig.97 : Screen shot of ***Error No e, Inputting a figure number***

Under these circumstances, if ***Select Figure*** push button is pressed then a blank figure window will be opened which will be of no use. This is not an error in the truest sense of the term, but algorithmically it is an error. The cross hair will also be there to get the coordinate information's.

**<u>Hence do not select a figure while the status is *Processing Completed* and figures are manually deleted and *Input Figure Number* is specified.</u>**

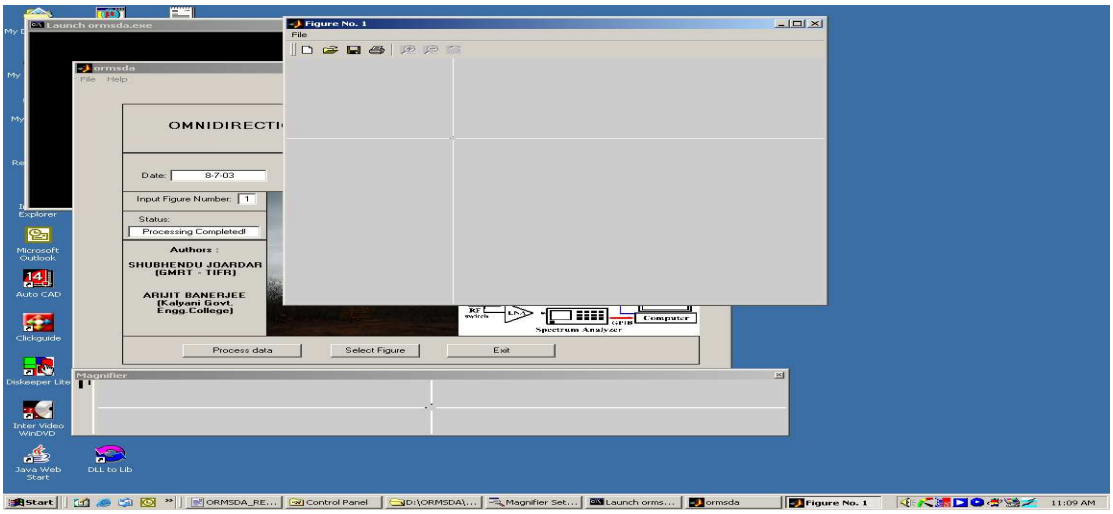


Fig.97 : Screen shot of ***Error No e, Invalid and irrelevant figure***

## f) Error due to deleting an irrelevant figure when the Status is ***Processing Completed*** and input figure is specified:

Here in the screen shot shown in Fig.98, we see that the Status field is ***Processing Completed*** and irrelevant figures are manually deleted.

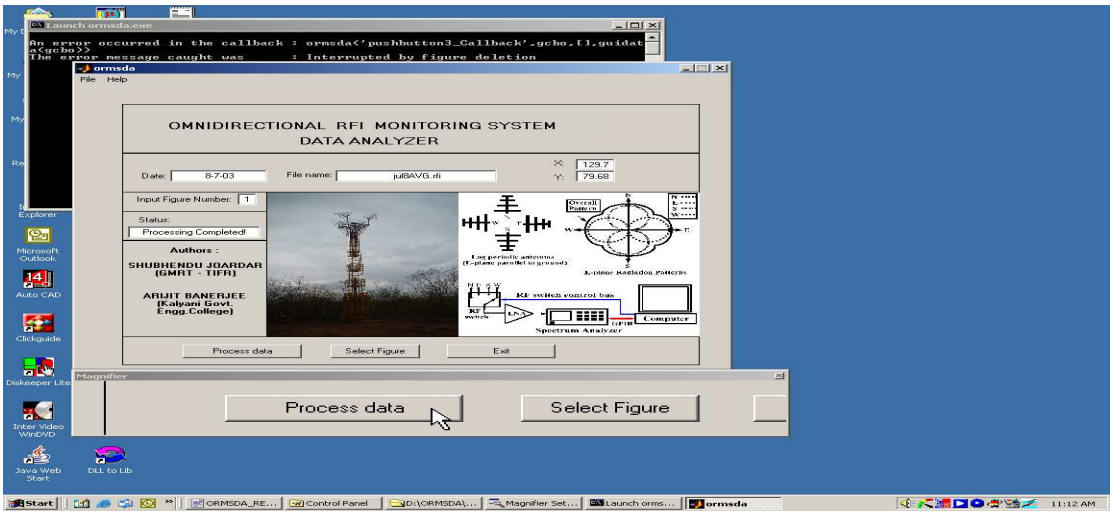


Fig.98 : Screen shot of ***Error No f, pressing Process data button***

Under these circumstances, the command window will show an error message. The screen shot of the error message in the command window is highlighted in the ***magnifier*** windows as shown in Fig.99.

**Hence do not delete an irrelevant figure when Status is *Processing Completed* and input figure is specified. Better do not try to create an invalid figure.**

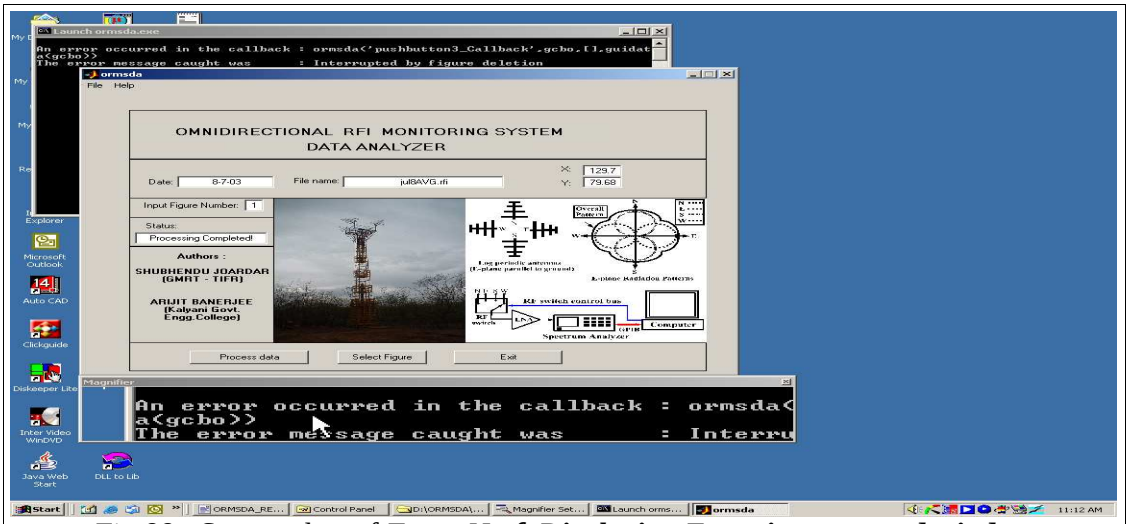



Fig.99 : Screen shot of ***Error No f, Displaying Error in command window***

## g) Error due to opening and analyzing a corrupted **AVG.rfi*** file:

Here in the screen shot in Fig.100, we see that the user is going to open a ****AVG.rfi*** file. Now we see that the user wishes to select a 111KB file, named ***abcAVG.rfi,*** which is a corrupted file (**insufficient data**).

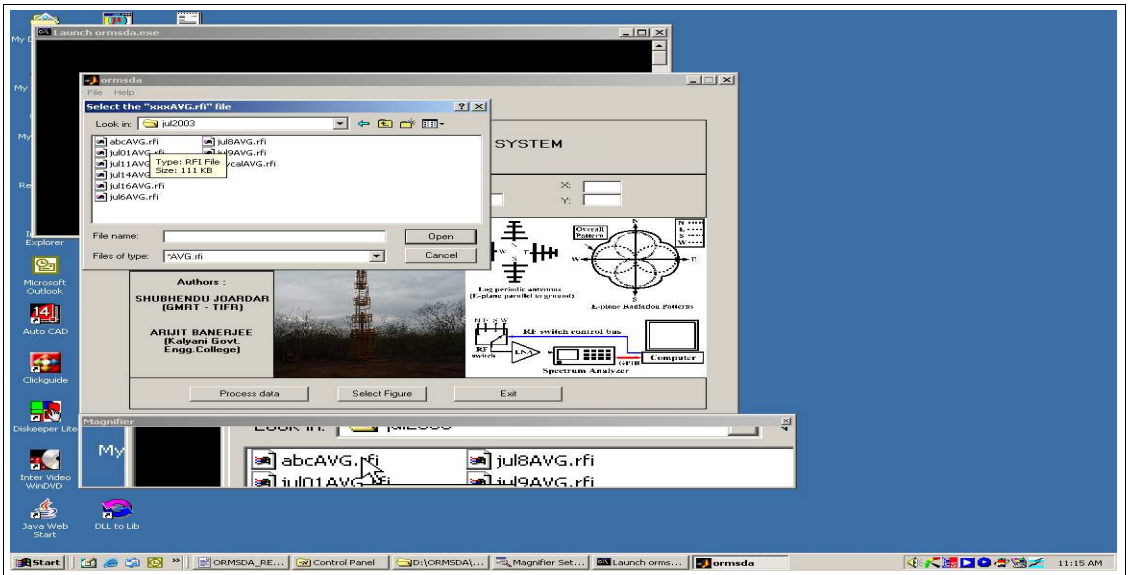



Fig.100 : Screen shot of ***Error No g, Selecting an corrupted data file***

After opening the corrupted file and tending to process, it results on an error in the command window. The command window shows an ***insufficient data*** message along with a ***divide by zero warning*** and some internal object error messages. See Fig.101. The wait bar stands still with a little bit of processing indicator level. Just close the wait bar manually.

**Hence do not try to process corrupted/insufficient data files and also not try to load and process invalid data files.**

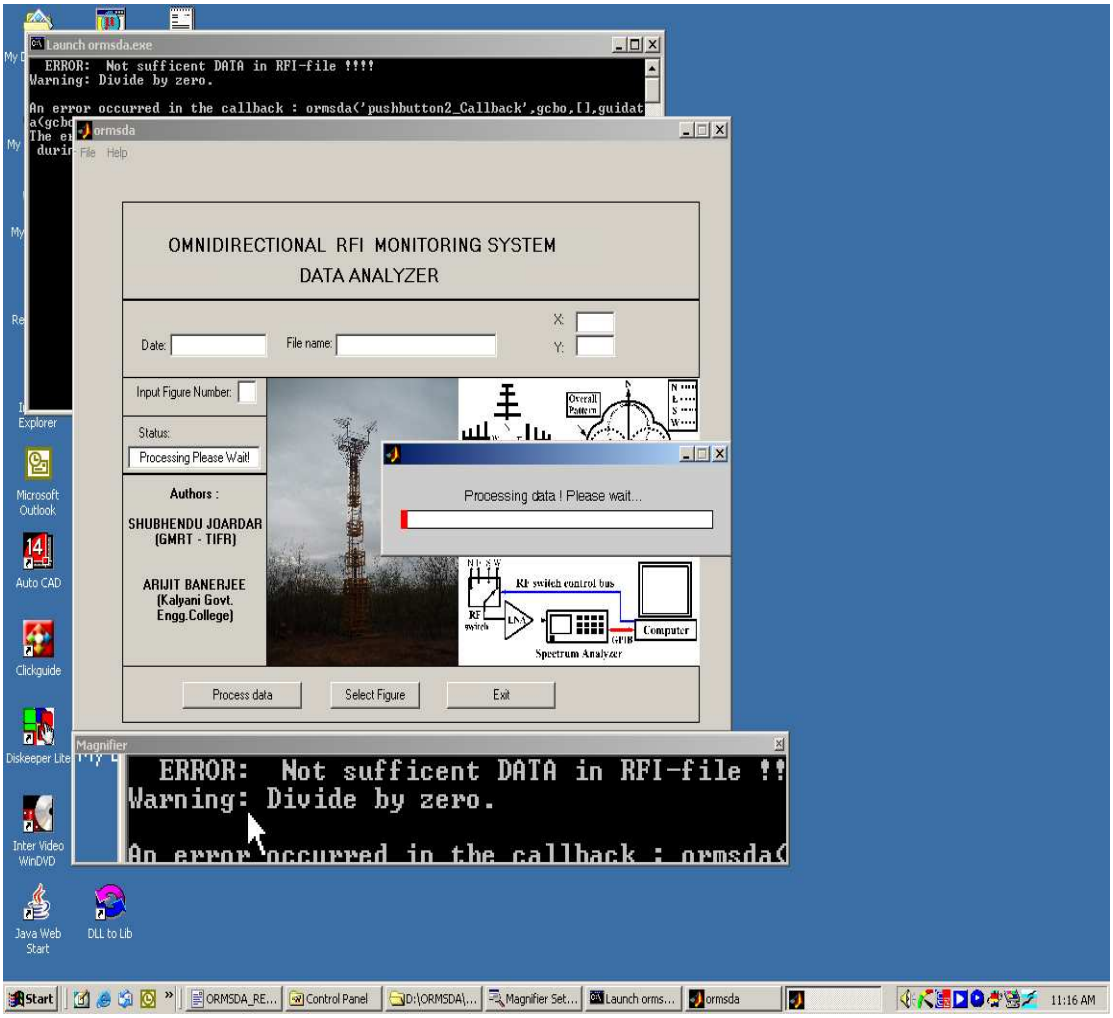



Fig.101 : Screen shot of ***Error No g, Displaying error in command window***

## h) Error due to selecting the documentation sub menu while the documentation file named ***ORMSDADOC.PDF*** is manually deleted:

Here in the screen shot shown in Fig.102, we see that the user is going to open the documentation file named ***ORMSDADOC.PDF*** which is manually deleted before by

some one.

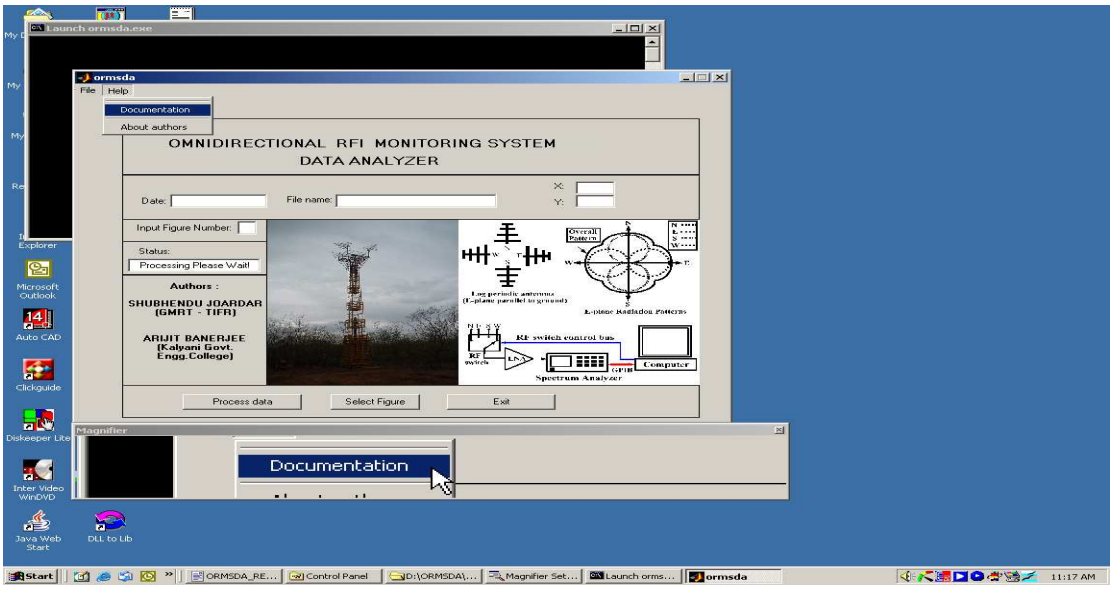



Fig.102 : Screen shot of ***Error No h, Clicking the Documentation sub menu***

In this situation, clicking the documentation sub menu results on an error. The error is something like ***ORMSDADOC.PDF is not recognized as an internal or external command, operable program or batch file.*** The magnifier window shows the error message clearly. See Fig.103 below.

**<u>Hence do not delete the documentation file manually and click the documentation sub menu thereafter.</u>**

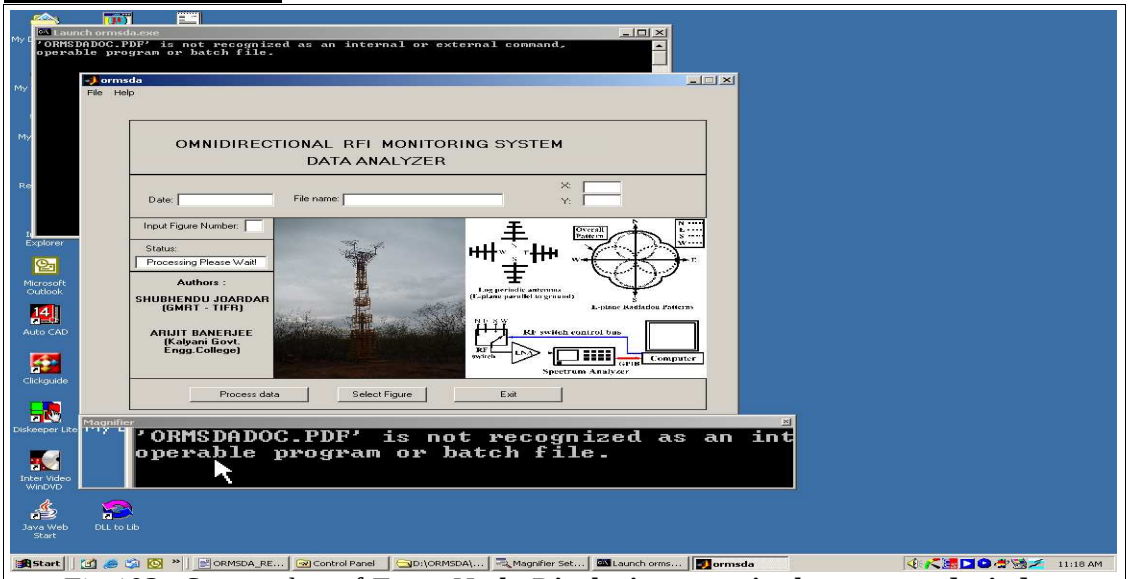



Fig.103 : Screen shot of ***Error No h, Displaying error in the command window***

# 13) Uninstalling the application:

To uninstall the application just follow the steps below:-

a) Open the ***Start*** menu. Go to ***Settings***. Click the ***Control Panel***. The operation is displayed on the screen shot as shown in the magnifier window in Fig.104 below.

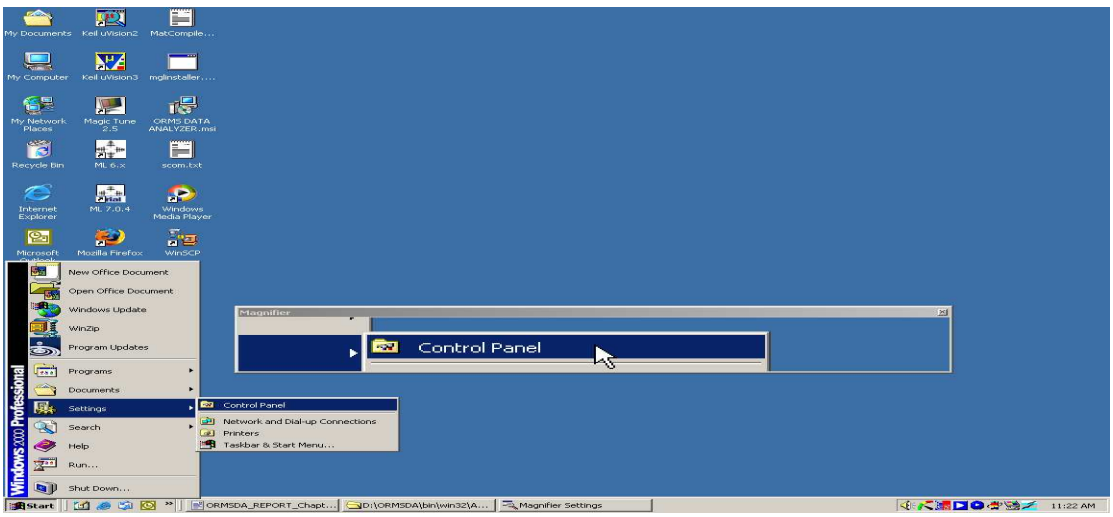



Fig.104 : Screen shot of ***Opening Control Panel***

b) After entering the control panel window double click the ***Add/Remove Program*** icon. The screen shot is showing the process in the ***magnifier*** window. See Fig.105 below.

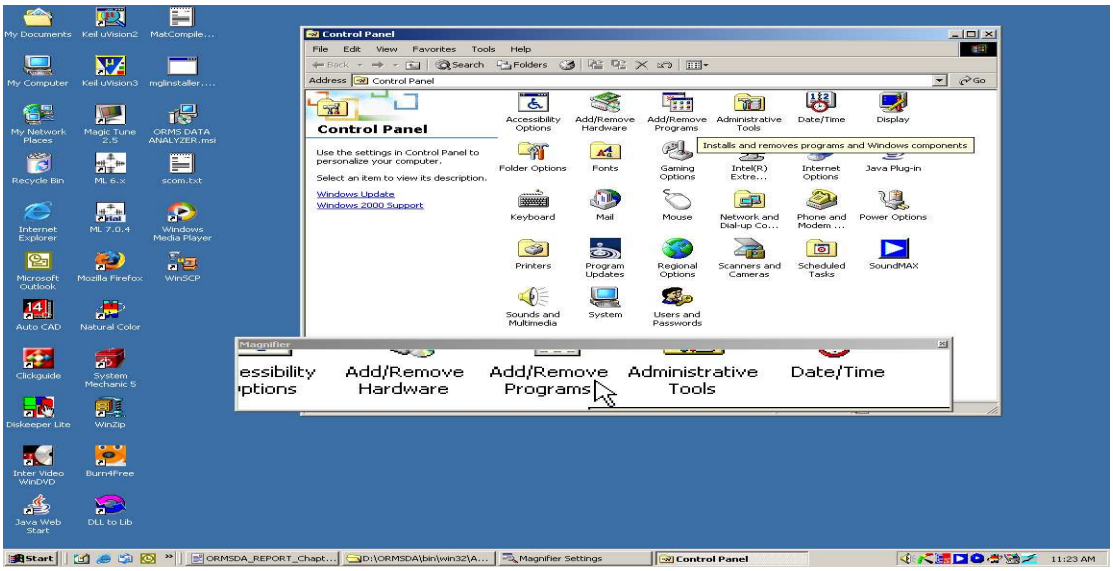



Fig.105 : Screen shot of ***Double clicking Add/Remove Programs***

c) A new window opens up. It shows the list of installed program in your machine. Select the program named ***ORMS DATA ANALYZER***.
d) If you want to change the settings, click the ***Change*** button and follow the instructions.

e) If you want to remove the application completely from your machine, click ***Remove***. The screen shot displays the process graphically as shown below in Fig.106.

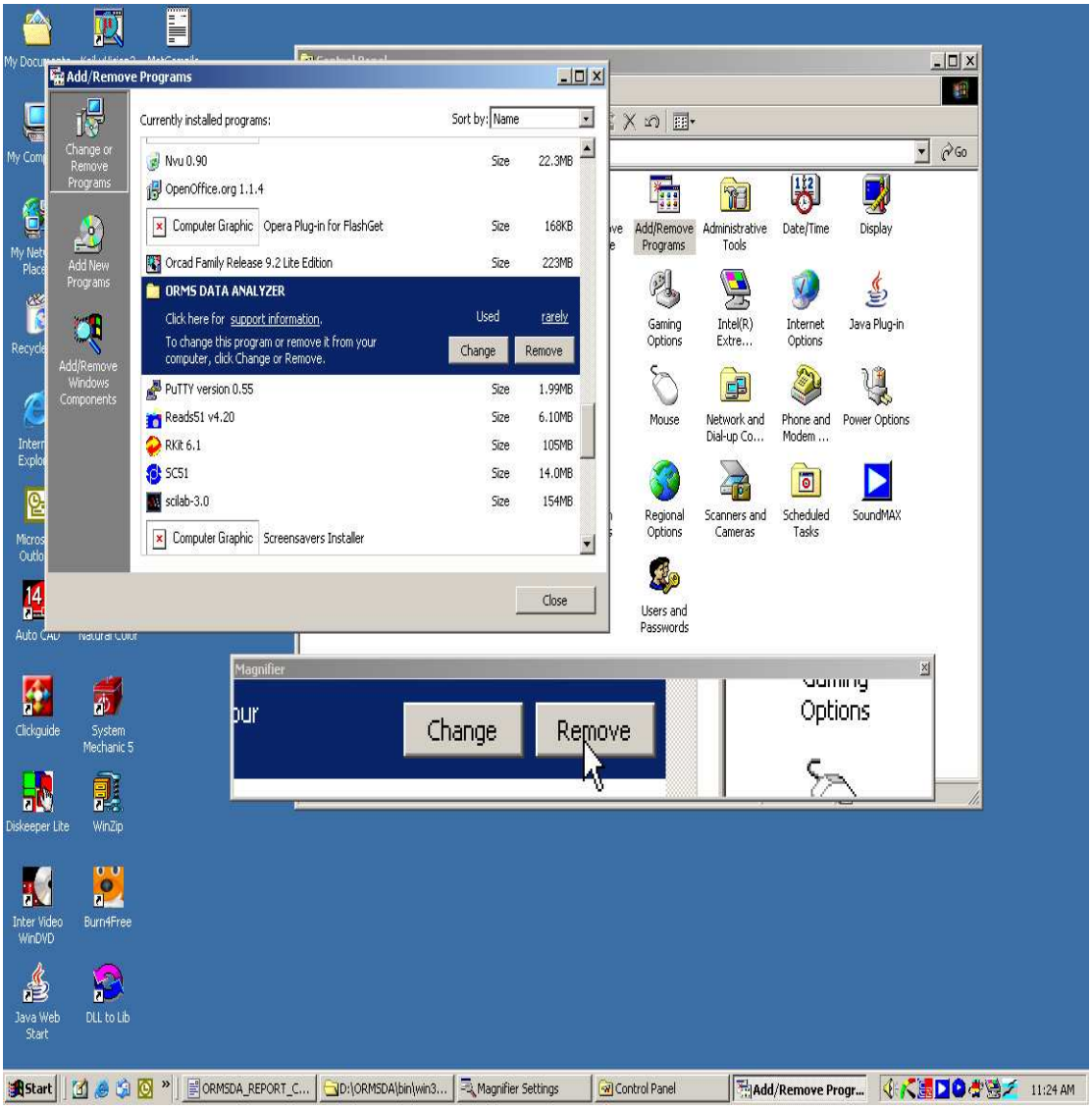


Fig.106 : Screen shot of ***Selecting ORMSDA to uninstall***

f) After clicking the remove button a new dialog window appears.

g) If you do not want to remove the software, click ***No***.

h) If you wish to uninstall the application from your machine, click ***Yes***. The screen shot shown in Fig.107 shows the process.

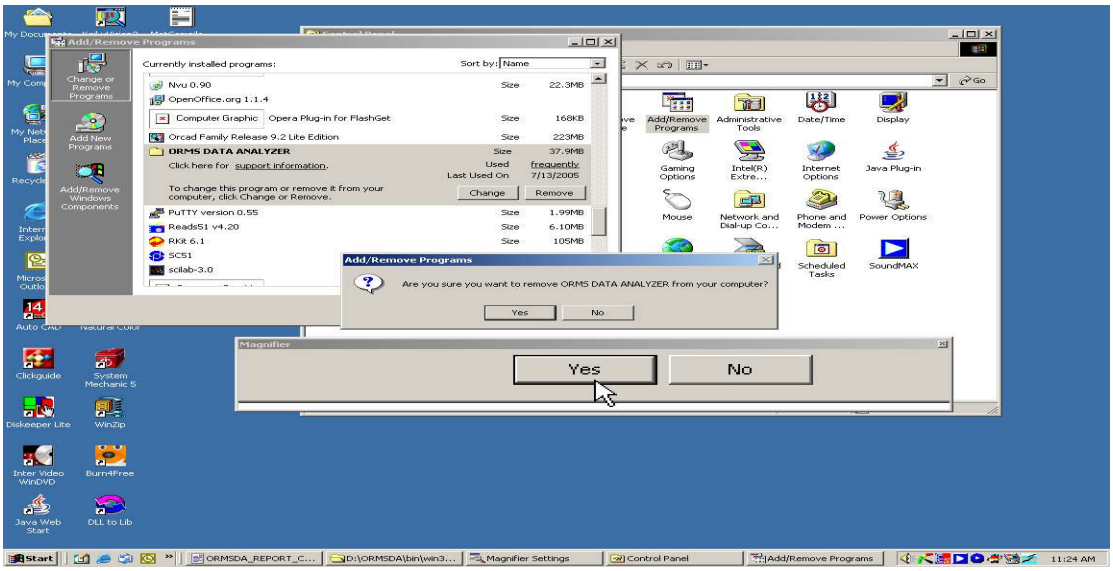



Fig.107 : Screen shot of ***Clicking the Yes button to uninstall the application***

i) The uninstallation process continues and ***ORMSDA*** application will be uninstalled completely from your computer.

j) Please manually delete the installation directory named ***ORMSDA***.

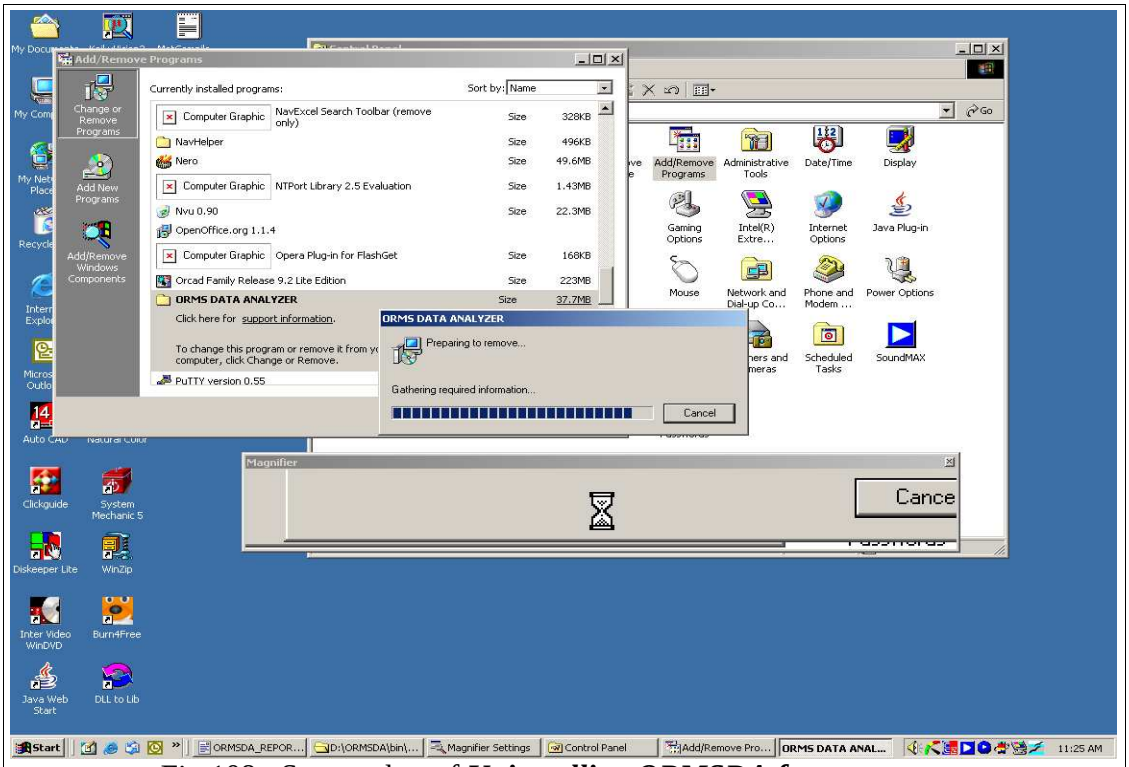



Fig.108 : Screen shot of ***Uninstalling ORMSDA from computer***

# Chapter 4

# RESULTS OF THE DATA ANALYSIS DONE BY ORMSDA WITH FEW **AVG.rfi* FILES

In this chapter we will show the results of some analyzed data files for the frequency ranges as mentioned below:-

1) 135 MHz – 175 MHz.

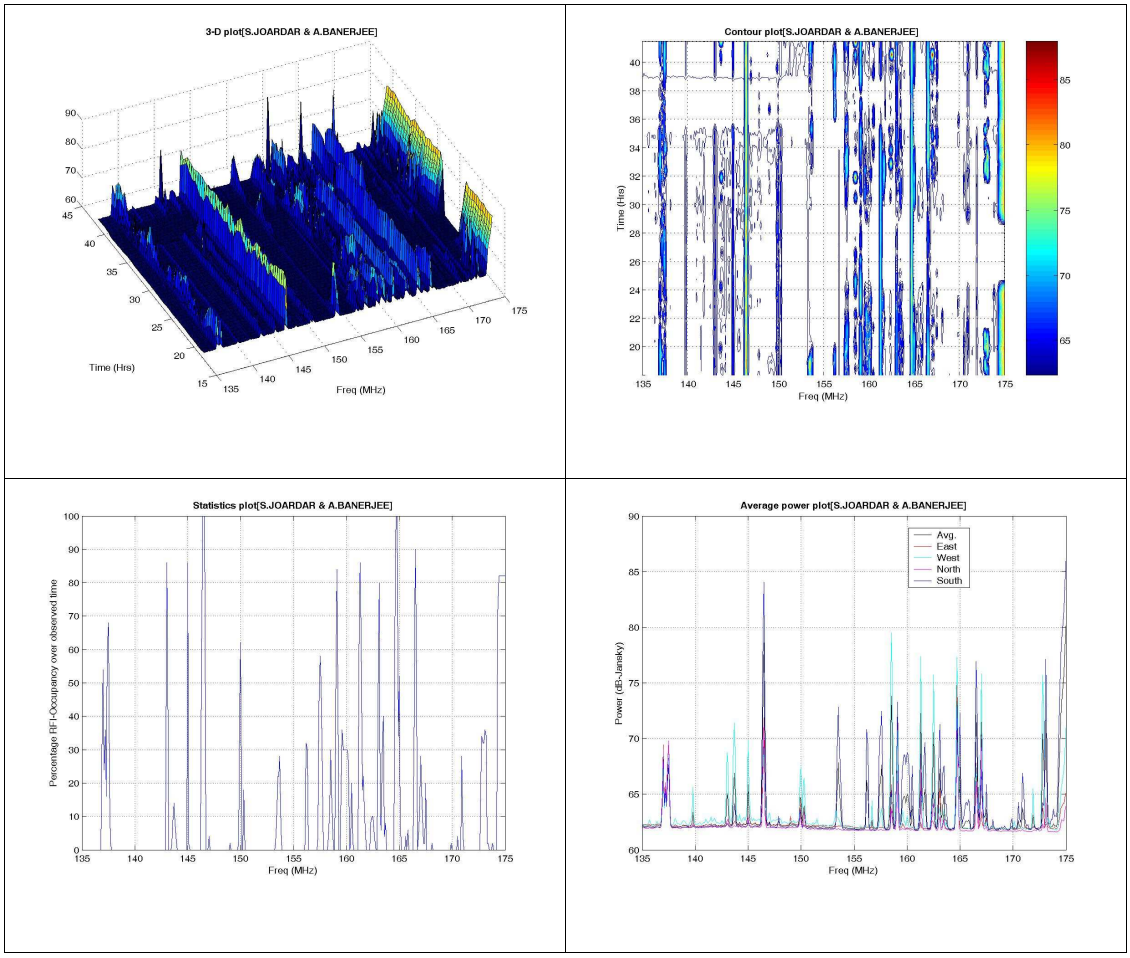
2) 110 MHz – 155 MHz.

3) 110 MHz – 155 MHz.

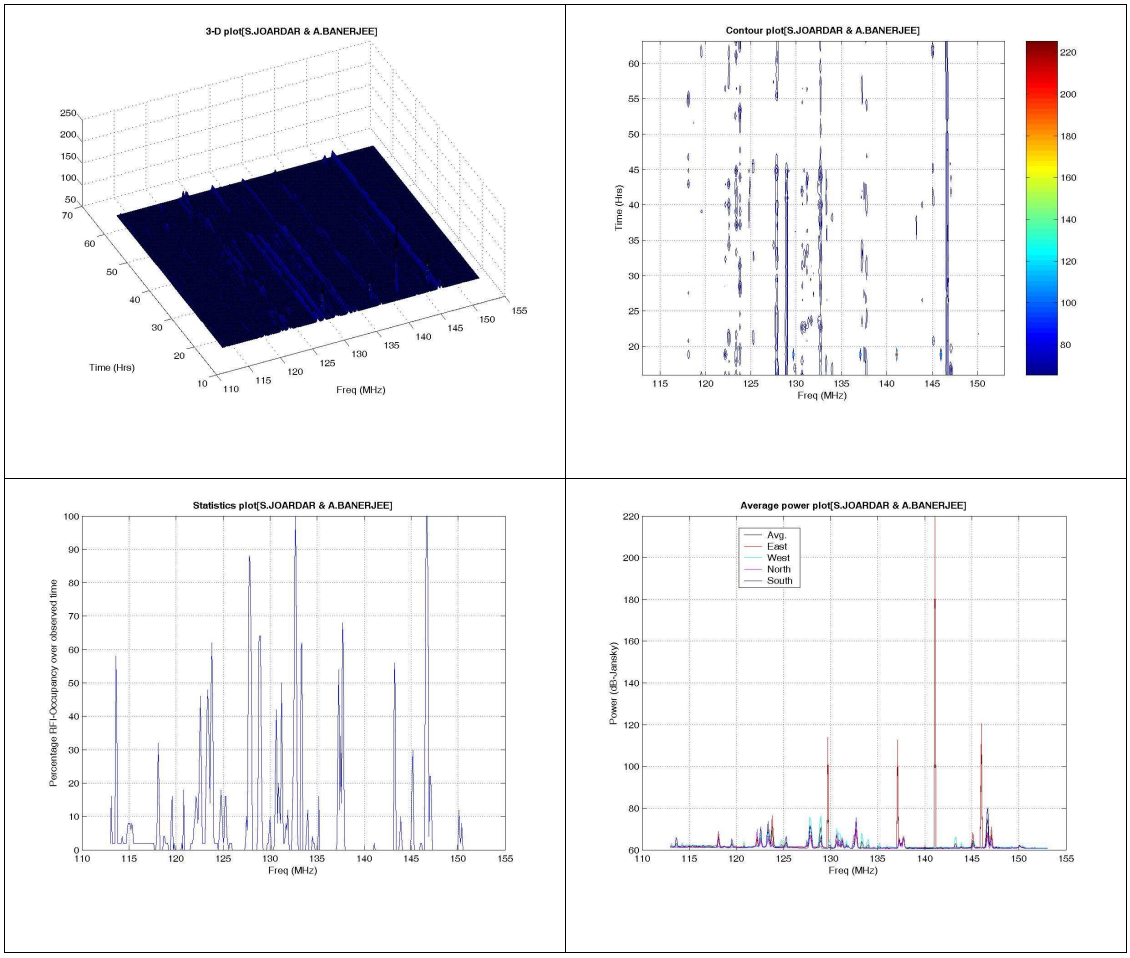
4) 50 MHz – 150 MHz.

Four plots are made from each data file, viz., three dimensional plot, contours, percentage RFI occupancy plot and time averaged plots for all the four directions.

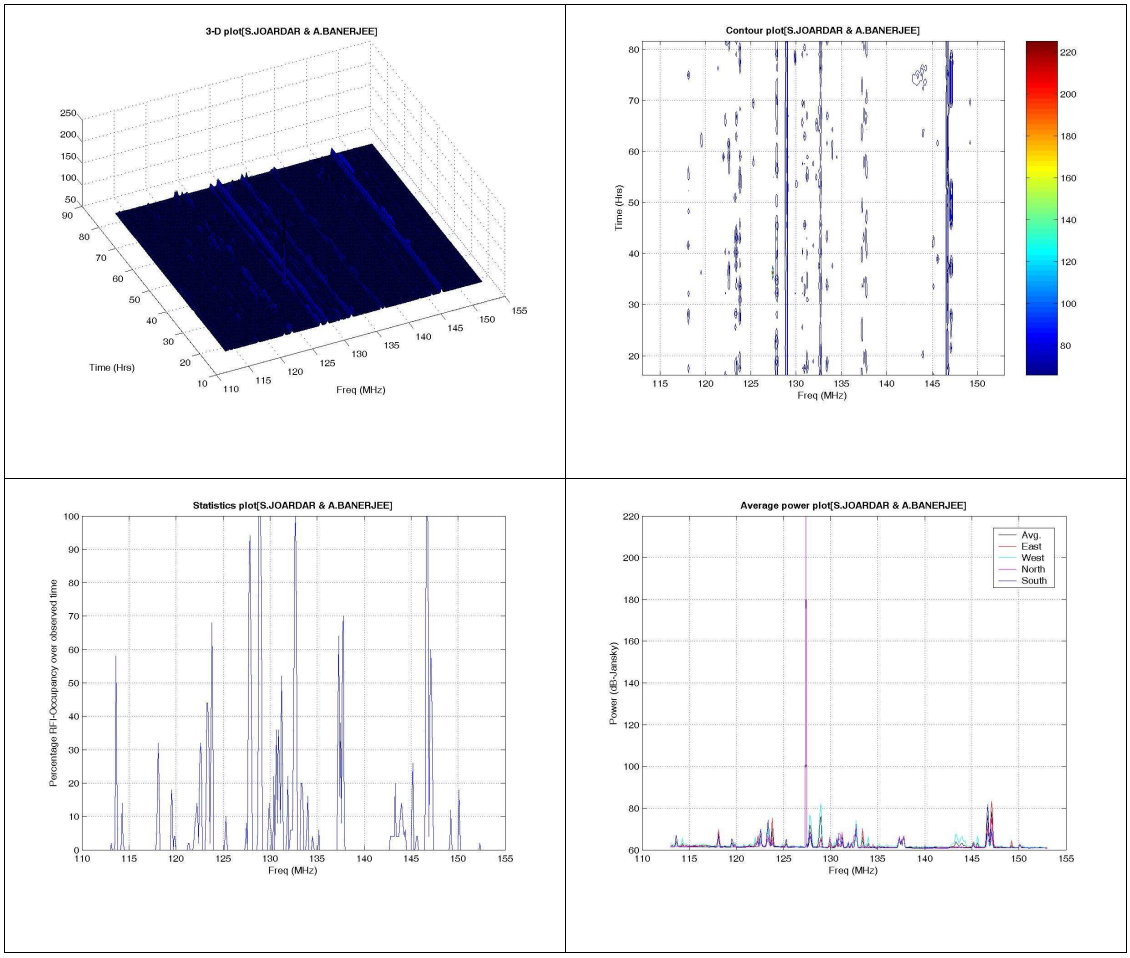
# 1) **135 MHz – 175 MHz**



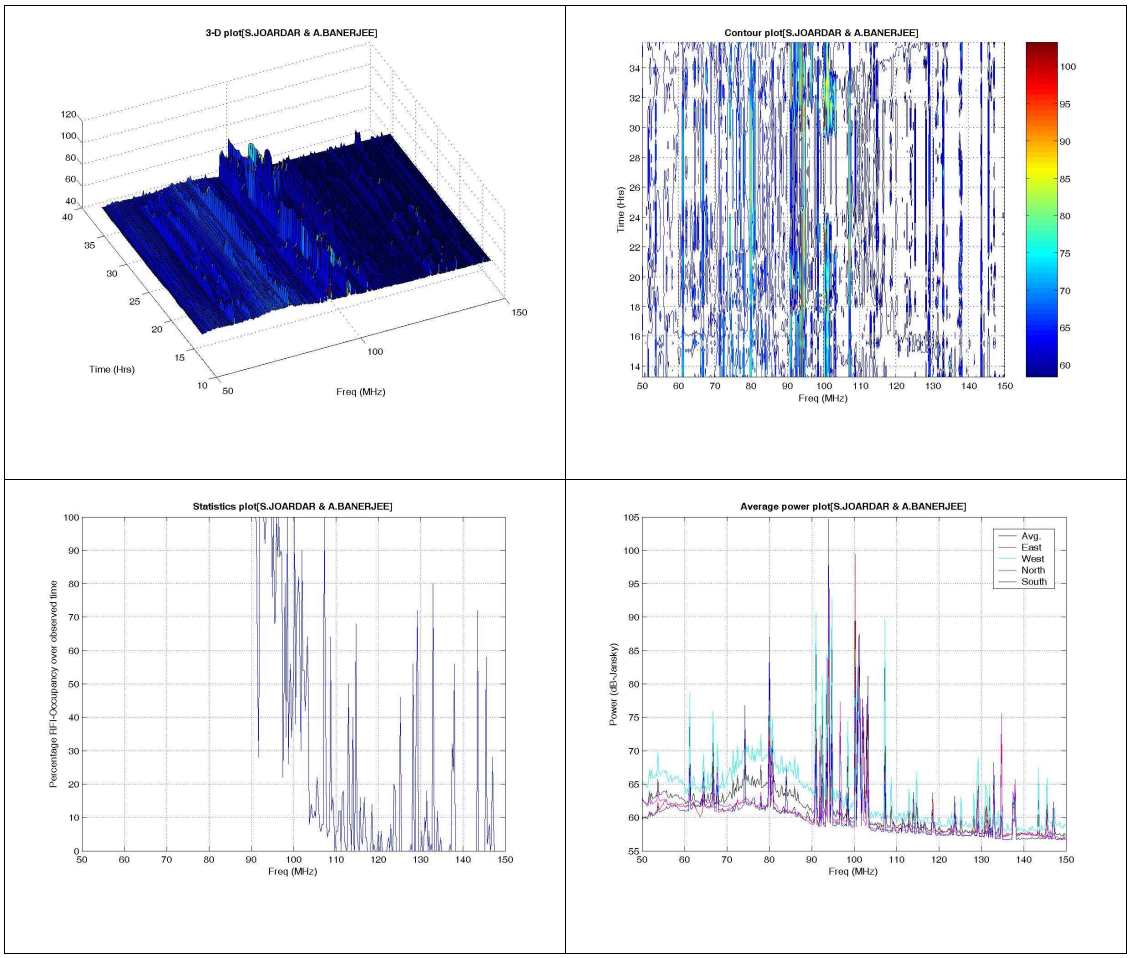
# 2) **110 MHz – 155 MHz**



# 3) **110 MHz – 155 MHz**



# 4) **50 MHz – 150 MHz**




---