

ELFO Analysis

FINAL PROJECT

Project number: H03D03

Final Report

Tuesday, 27 May 2003



Authors:

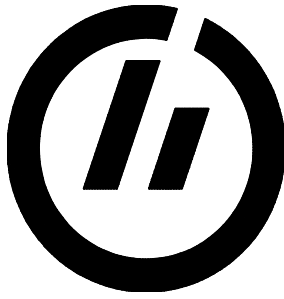
Jan Steinar Kvilesjø

Christian André Bråthe

Lars Martin Dobbe

In cooperation with the Istituto di Radioastronomia.





OSTFOLD COLLEGE

Ostfold College - Engineering education
Postboks 1192, Valaskjold, Visit: Tuneveien 20
1705 Sarpsborg
Telefon: 69 10 40 00 , Telefaks: 69 10 40 02
E-mail: post-ir@hiof.no URL: www.hiof.no

1 Tile page

Final Project	Free accessible	<input checked="" type="checkbox"/>
Number of ECTS: 15		
Engineering field: Computer Science	Accessible after agreement with the contractor	<input type="checkbox"/>

Project title: <i>E.L.F.O. Analysis</i>	Date: 2003-05-26
	Number of pages: 56
	Number of attachments: 7
Authors: Jan Steinar Kvilesjø, Lars Martin Dobbe and Christian André Bråthe	Councillor: Erling P. Strand
Department / line: IA - Informatics and Automatisasjon	Project code: H03D03

Produced in cooperation with: CNR, IRA, Italy	Contact person at the contractor: Stelio Montebugnoli
--------------------------------------------------	-------------------------------------------------------------

Extract: The E.L.F.O. system is a system developed by the IRA in Italy. Its purpose is to record low frequency electromagnetic signals which might be of interest for research on strange phenomena in Hessdalen. The system today consists of software and one E.L.F.O. correlation unit which can be connected to a personal computer by the USB port. The purpose of the project is debugging the existing software, create a user manual and to develop an automatic system to acquire low frequency signals in, for example, Hessdalen (Norway).

3 indexing terms:	<input type="text" value="Hessdalen Phenomena"/>
	<input type="text" value="E.L.F.O."/>
	<input type="text" value="Low frequent signals"/>

2 Preface

This report describes the E.L.F.O. analysis project, which is a senior student project at Ostfold Collage, and it's a part of the Computer Science study, at the engineering department. It's done in cooperation with Institute of Radio Astronomy (IRA) in Medicina, Italy.

Ostfold Collage has departments several places in Ostfold, Norway. In Halden the following departments are located: teacher education, informatics and automation and social studies and languages. In Sarpsborg, the engineering department is located. And last the academy of figurative theatre and health care is located in Fredrikstad. The collage has about 3725 students divided all over the different departments.

The Institute of Radio Astronomy is located Italy. And it's a part of the Italian National Research Council (CNR). It has a staff of about 100, astronomers, electronic engineers-physicists, software specialists, technical and administrative support. The Institute headquarters are in Bologna. Other sections of the institute are located in Firenze, Matera, Medicina Noto and Cagliari. The Medicina station has about 24 employees.

At this station we have got much help and support from the employees. We thank them for everything. We will also like to thank Per-Olav Rusås, Martin Kermit and Åge J. Eide for good help and information during this project.

E.L.F.O. analysis homepage: <http://syfest.hiof.no/~d200303/>

CNR homepage: <http://www.ira.cnr.it/>

Sarpsborg, 27 May 2003

Christian André Bråthe

Lars Martin Dobbe

Jan Steinar Kvilesjø

3 Contents

1	TILE PAGE	2
2	PREFACE	3
3	CONTENTS	4
4	SUMMARY	6
5	INTRODUCTION	7
6	ORIGINAL SITUATION	8
6.1	OVERVIEW	8
6.2	SYSTEM OVERVIEW	9
6.2.1	E.L.F.O. CORRELATION UNIT	10
6.2.2	E.L.F.O. SOFTWARE	12
7	USER REQUIREMENTS	15
7.1	USER DEMANDS FOR THE EXISTING E.L.F.O. SOFTWARE.....	15
7.1.1	GENERAL EXISTING E.L.F.O. SOFTWARE DEMANDS	15
7.1.2	DOCUMENTATION REQUIREMENTS	15
7.2	USER DEMANDS FOR THE E.L.F.O. ANALYZE SYSTEM.....	16
7.2.1	ILLUSTRATE THE WAV FILES AS JPG IMAGE.....	16
7.2.2	SIGNAL FILTERING / NEURAL NETWORK	16
7.2.3	UPLOADING FILES TO A REMOTE LOCATION.....	16
7.2.4	FILE COMPRESSION	17
7.2.5	REMOTE ADMINISTRATION	17
7.2.6	DOCUMENTATION REQUIREMENTS	17
8	SYSTEM SPESIFICATION	18
8.1	EXISTING E.L.F.O. SYSTEM IMPROVEMENT.....	18
8.1.1	<i>Software debugging</i>	18
8.1.2	<i>User manual</i>	19
8.1.3	<i>Running the software on Unix/Linux platform</i>	19
8.2	TOTAL E.L.F.O. SYSTEM DESCRIPTION.....	20
8.2.1	<i>Development of the system</i>	20
8.2.2	<i>Software</i>	22
8.2.2.1	Wav2Jpg converter.....	23
8.2.2.1.1	Automatic Mode	23
8.2.2.1.2	Manual Mode.....	24
8.2.2.1.3	3D feature	24
8.2.2.1.4	Technical solution.....	25
8.2.2.1.4.1	Retrieving signal using the javax.sound.sampled API.....	26
8.2.2.1.4.2	The FFT Algorithm.....	28
8.2.2.1.4.3	Integer to RGB conversion.....	30
8.2.2.1.4.4	Creating the JPG image	30
8.2.2.1.4.5	Complete conversion example	31
8.2.2.1.4.6	Displaying an image in 3D.....	33
8.2.2.1.4.7	Known errors and possible improvements	33
8.2.2.2	Neural network filtering software.....	34
8.2.2.2.1	Tested algorithms	34
8.2.2.2.2	The O-Algorithm.....	35
8.2.2.2.2.1	Pre-processing the signal with Haar wavelets.....	35
8.2.2.2.2.2	O-algorithm (Technical description)	36
8.2.2.2.2.3	Training data	37
8.2.2.2.2.4	Advantages with O-algorithm	37
8.2.2.2.2.5	Known problems with O-algorithm	37
8.2.2.2.3	The Feed-Forward Back-Propagation algorithm	38
8.2.2.2.3.1	Pre-processing the signal with FFT transformation	38
8.2.2.2.3.2	Feed-Forward Back-Propagation algorithm (Technical description)	38
8.2.2.2.3.3	Training data	40

8.2.2.2.3.4	Advantages with Feed-Forward Back-Propagation algorithm.....	40
8.2.2.2.3.5	Problems with feed-forward back-propagation network.....	40
8.2.2.2.4	Algorithm conclusion	41
8.2.2.2.5	Software structure and use of the chosen algorithm	41
8.2.2.3	Scheduler	42
8.2.2.3.1	Program structure.....	42
8.2.2.3.2	Program usage.....	45
8.2.2.3.3	Uploading files to remote location	45
8.2.2.3.4	Compressing files	46
8.2.3	<i>Webpage</i>	47
8.2.3.1	Web-Page structure.....	47
8.2.3	<i>Remote administration</i>	48
8.3	CONCLUSION	49
9	PROJECT SCHEME	50
9.1	MILESTONE CHART	50
9.2	ACTIVITY AND RESPONSIBILITY CHART FOR THE PROJECT	50
9.2.1	<i>Activity chart</i>	50
9.2.2	<i>The responsibility distribution</i>	51
9.3	PROJECT BUDGET (SKETCH)	52
10	ABBREVIASJONS SURVEY.....	53
11	SOURCES.....	54
12	PICTURE REFERENCES	55
13	ENCLOSURE INDEX.....	56

4 Summary

The E.L.F.O. system today consists of software and one correlation unit which can be connected to a personal computer through the USB port. The system is developed by the IRA in Italy, and its purpose is to record low frequency electromagnetic signals which might be of interest for research on strange phenomena in Hessdalen.

This project has two purposes. The first one is to check the existing E.L.F.O. software for bugs and report those to the creator for correction. A number of bugs have been found, and are reported to the creator. It's also necessary to write a user manual for the E.L.F.O. software so other peoples who don't know the program can more easily use it. This user manual can be found as enclosure 3.

Latter, a complete E.L.F.O. analyze system has been designed. This system should be able to filter out uninteresting signals, and then to send interesting signals to a remote server. There, the data can be presented on web. To do this we need to make the system more intelligent in order to decide "good" data from the uninteresting one for example using a neural network system. When the neural network has filtered out interesting data, our wav2jpg converter will create images. The jpg output file will be sent to www.hessdalen.org. After filtering has been done, the original wav files are compressed and stored on a large disk for backup.

In Hessdalen there is an automatic station called "Blue Box": The system is intended to be used there, to record any possible low frequent signals coming from the strange phenomenon.

You can read more about Hessdalen at: www.hessdalen.org

5 Introduction

Project Hessdalen started in 1983 with a large expedition. The result of this was 53 visual observations. Some years later, an automatic measurement station was put up. This station (Figure 5-1) automatically records and presents the result on web.



Figure 5-1: Hessdalen station

The cooperation between the Hessdalen project and IRA started in the beginning of 1990 by Massimo Teodorani, who made contact with Erling P. Strand.

1996 a cooperation between IRA and Østfold Collage started in order to use some of the instruments designed for radio astronomy for acquiring data from Hessdalen. This project was called Embla. During a mission in 2000, Italian researchers designed a special VLF (Very Low Frequency) receiver that records electromagnetic signals. This receiver was called E.L.F.O.

6 Original situation

6.1 Overview

The E.L.F.O. system is consisting of: Two VLF (Very low frequency) antennas, two VLF receivers, one correlation unit and one PC with software. With the correlation unit you can receive electromagnetic signals from two antennas, and send the captured signals to a computer using the USB interface. The software has the ability to capture and store the signals coming from the receiver. Results from this system can be used to analyze for example the Hessdalen Phenomena.

With the E.L.F.O. we can analyze signals between 1 to 16 kHz. The E.L.F.O. software you can also specify a smaller range. It can, for example, only capture signals between 1 to 5 kHz if that's more appropriate.

Every natural signal on earth (for example: lightning and other electromagnetic interference in the atmosphere) are VLF signals. There for the E.L.F.O. system is created to capture those signals. If an unknown phenomena occurs, the signal can be analyzed to see if it's only a natural signal or if it's something strange occurs.

In Figure 6-1 a test setup of the E.L.F.O. system is shown.

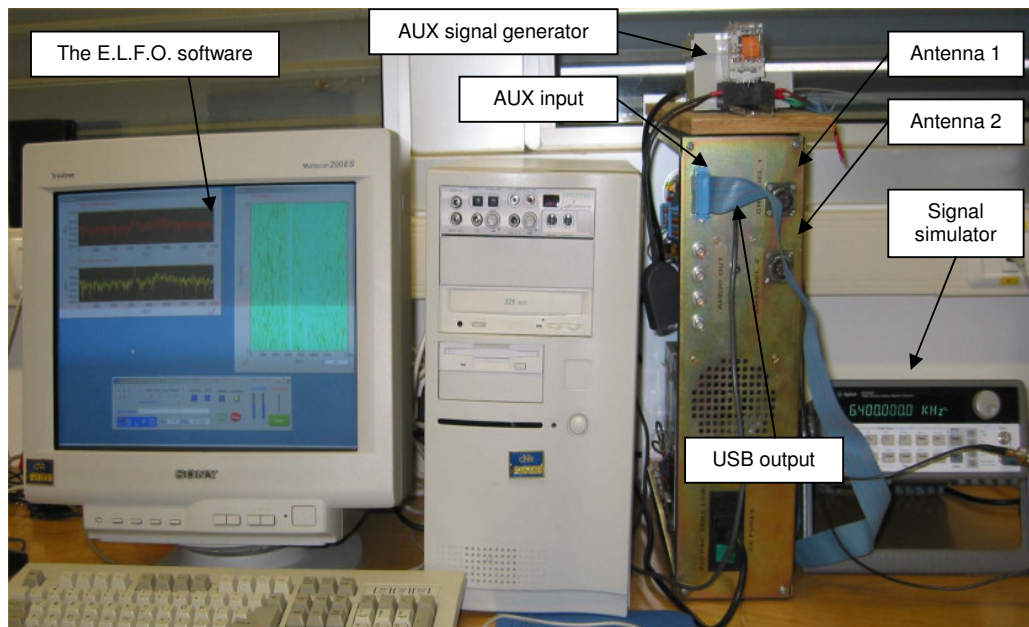


Figure 6-1: Test setup overview

6.2 System overview

The E.L.F.O. system consists of the two VLF receivers that get the signals from the antennas. The VLF receivers are connected to a correlation unit. This unit is controlled by a host computer which has the E.L.F.O. software installed. On the host computer the E.L.F.O. software reads the digitalized signal through the USB port and then decide if the signal should be saved or not. The wav files then can be sent to a remote location. The system was tested in 2000 in the “Blue Box” in Hessedalen, Norway.

See Figure 6-2 for a system overview.

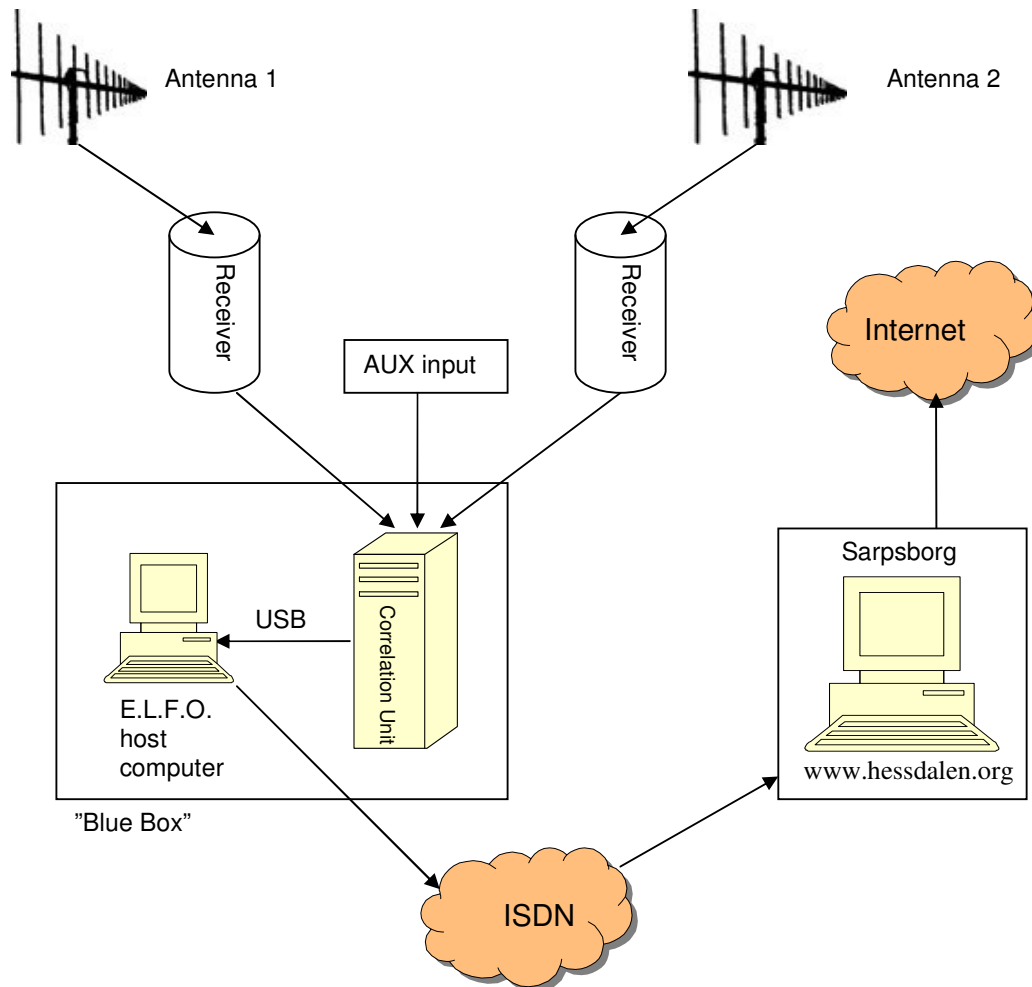


Figure 6-2: E.L.F.O. system overview

6.2.1 E.L.F.O. correlation unit

The correlation unit (Figure 6-4) correlates the two received signals from the antennas. The signals are sent to the correlation unit and the ADC (Analog to digital converter). Special output on the correlation unit allows you to listen to the received signals through a loudspeaker and take them out on an audio interface. Another possibility is to take the digital signals from the ADC out through the USB port to the host computer that controls the whole system.

Inside the E.L.F.O. correlation unit, there is a Ni-DAQ 6020, USB interface unit, used to capture real-time data. It's this DAQ (Digital Acquisition) unit which controls the communication with the computer through the USB port. This part is essential to make the correlation unit work with a host computer. You can read more about the Ni-DAQ 6020 in the official specification PDF found here [10].

You can see Figure 6-3. There you will see the data float inside the correlation unit.

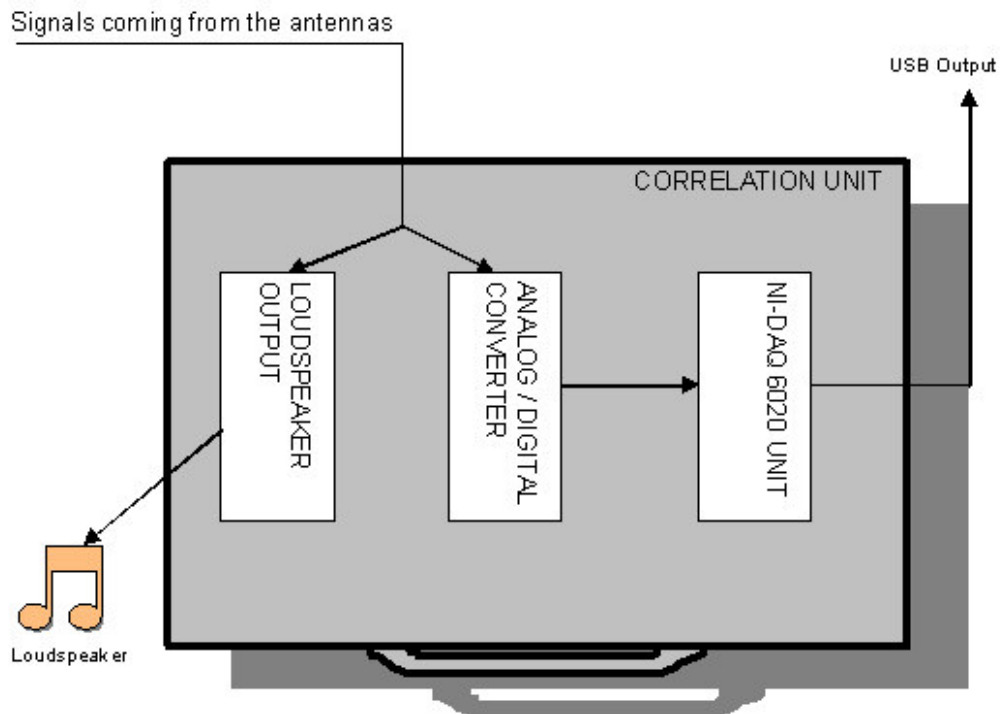


Figure 6-3: Correlation unit overview

The E.L.F.O. correlation unit also has an AUX input. Here you can, for example, connect a signal from a camera which can be used to trigger the recording mechanism. In this way, if for example a camera captures a strange phenomenon, the data capture can start.

Inside there are also connected two batteries; they can be used to make the unit mobile. If you are going to use the E.L.F.O. system in a place without power, it can operate for several hours.

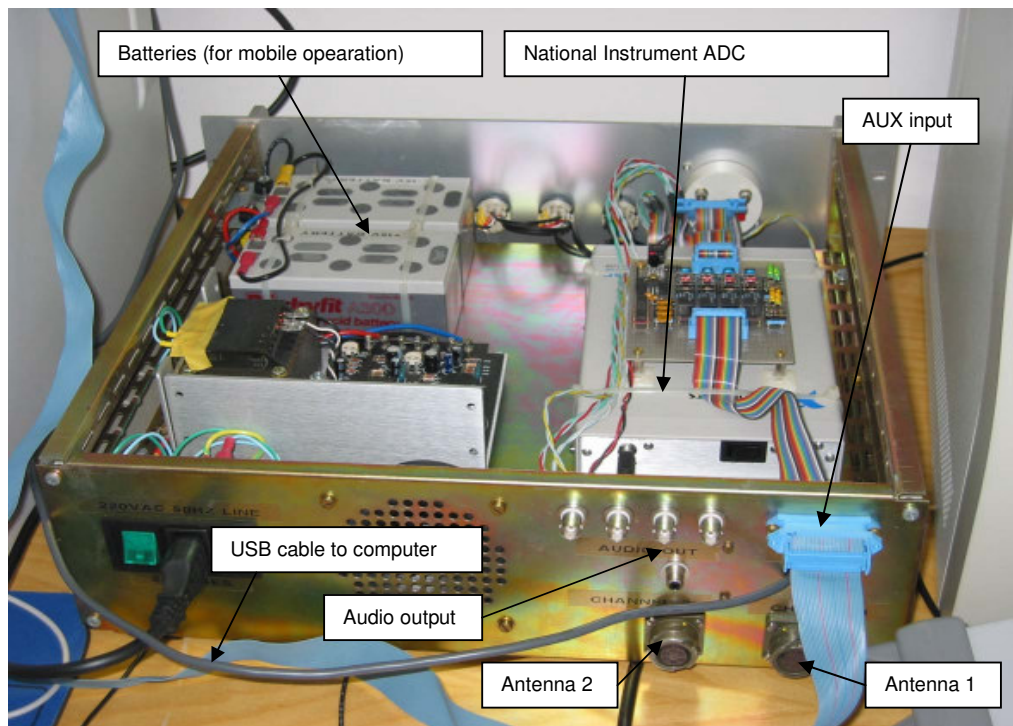


Figure 6-4: The E.L.F.O. correlation unit

6.2.2 E.L.F.O. software

Andrea Cremonini has written the E.L.F.O. software. It's written in CVI (C for Visual Instruments, where C is the programming language ANSI C), and the source code is in ANSI C programming language.

The software always displays the power spectrum of the input from both of the antennas at the same time (as long there is an antenna connected). When an alarm occurs, the software starts recording the signal from the antenna you choose to record from in the settings. There you can choose to record signals from antenna one, antenna two or both of them. See Figure 6-5 and enclosure 3.

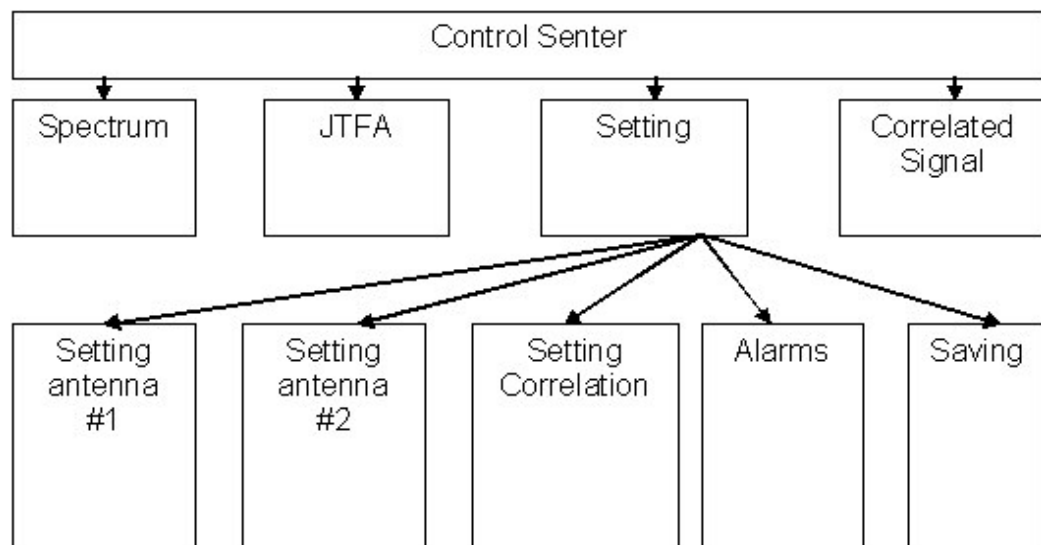


Figure 6-5: E.L.F.O. GUI structure

The E.L.F.O. software can operate in three different modes:

- Automatic: The E.L.F.O. software looks for signals over a given level. If a signal becomes stronger than this level, the software automatically records from the channels set in the settings. This “trigger” level can be set in the spectrum window for each antenna, or in the antenna settings window.
- Manual: The user can manually record signals. You can start recording the signals from the channel set in the settings, by pushing the SAVE button.
- External: An external signal, coming from a different instrument (camera, radar, etc) connected to the AUX port, can trigger the alarm. This must be activated in the settings as well.

Frequencies that are not interesting can be cut away by the software. The software also has the opportunity to display a joint time-frequency analysis of the signal.

This system is the basic tool for capturing signals from the antennas. It stores the signal in a normal WAV file which you can listen to or view in other programs afterwards.

You can read more about the E.L.F.O. software structure and usage in the user manual found as enclosure 3.

Figure 6-6 shows the basic user interface of the E.L.F.O. Software.

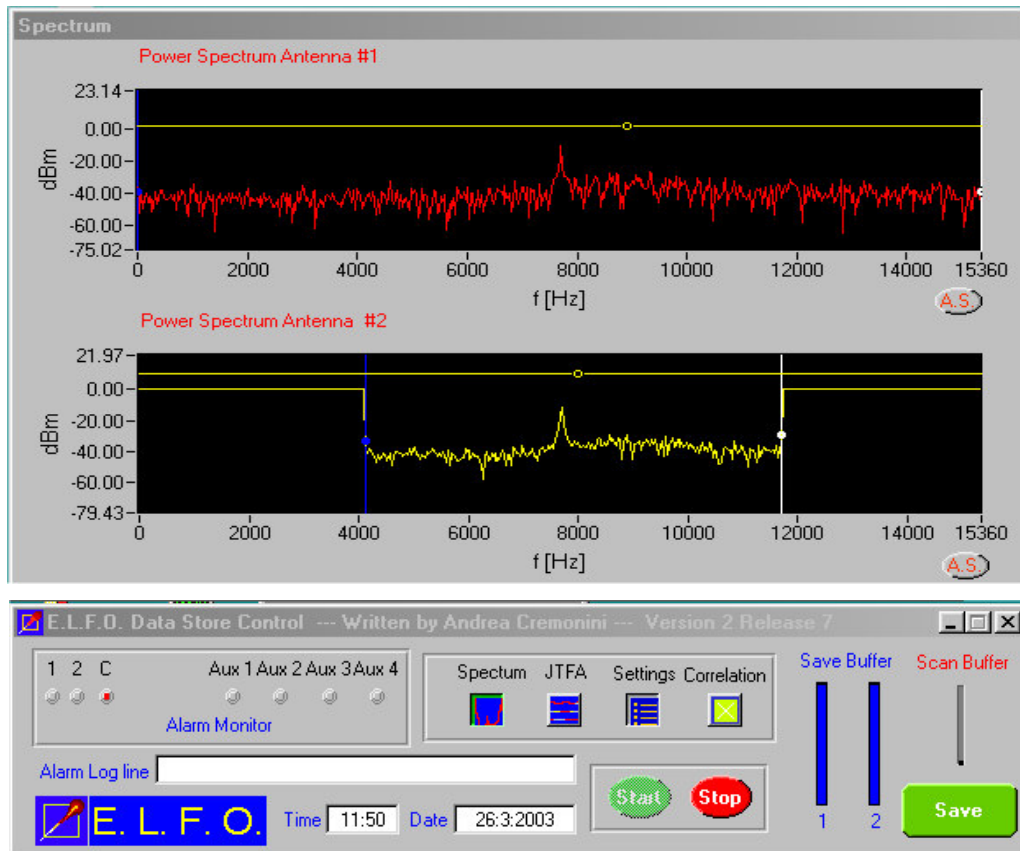


Figure 6-6: Basic E.L.F.O. software user interface.

7 User requirements

This user requirement is created as an agreement between Erling P. Strands demands and Stelio Montebugnoli, Andrea Orlati, Jader Monari, Andrea Cremonini and Marco Polonis demands.

7.1 User demands for the existing E.L.F.O. software

7.1.1 General existing E.L.F.O. software demands

There are no new demands for this software, except that the software is to be debugged. The main task is to come with suggestions on how to make the software better and more stable than it is today.

The errors are to be reported in a debug report which can be handed over to and corrected by the creator.

A second task is to investigate the possibility to make the software work under a Unix/Linux operating system. Is there a possibility for the software and the hardware to work correctly under the UNIX environment?

7.1.2 Documentation requirements

There is to be created a fully functional user manual for the software, since this does not exist at present time. The user manual must be written in English, to reach a big spectre of users.

7.2 User demands for the E.L.F.O. analyze system

7.2.1 Illustrate the wav files as jpg image

The system has to convert the WAV files to JPG files, which is extremely much smaller than the WAV file. But remember that with the JPG format some information is lost, and is only useful for visualization, so the WAV files must be stored as well.

Those jpg files can be transferred to a remote location and can be displayed on a web page. This makes a great visualizing of the signal captured from the E.L.F.O. system.

7.2.2 Signal filtering / Neural Network

The system must be able to filter out known signals in order to reduce the transfer rate and to store unnecessary data. The filter can use the wav files created by the E.L.F.O. software to find similar signals based on already known signals.

If a similar signal is found, the wav file can be deleted, or kept, depending on the users' request. The file must be compressed and stored on a large disk (example 120 GB).

7.2.3 Uploading files to a remote location

The system must be able to send the jpg format files to a remote location, so the user can see if the files are interesting or not. If it's not interesting the user can decide to manually delete wav file from the E.L.F.O. computer.

This jpg file can be uploaded in the official webpage of project Hessdalen:
www.hessdalen.org.

7.2.4 File compression

When the system decides to keep the WAV file it must be compressed to save space. This can be done in several ways, but the best way is to zip down the wav files to make it smaller. In this way you will not loose any information (zip compression isn't data loss algorithm).

You can also store several wav files in the same zip archive. All wav files created the same day can be compressed into the same archive.

7.2.5 Remote administration

A remote connection to the system must be considered. In this way the user can for example reset, stop, start manual save and check logs for the whole system.

7.2.6 Documentation requirements

The complete system must have a user manual on how to set up and how to use the system. The user manual must include how to use the WAV2JPG converter program, and the neural network program since those programs are going to be a bit bigger than the rest. Besides that, no particular documentation is demanded.

Notice: This documentation is additional to the E.L.F.O. software documentation.

8 System spesification

8.1 Existing E.L.F.O. system improvement

This section refers to 7.1: User demands for the existing E.L.F.O. software.

8.1.1 Software debugging

The E.L.F.O. software has been tested on several operating systems, including: Windows 2000 Server, Windows 2000 Professional, Window XP and Windows 98.

Several errors have been found in the existing E.L.F.O. software. Underneath there is a list of some bugs found (They are spitted into two categories: Major and minor bugs).

Major problems that are found:

- **The wav files.**

The problem here is that the software stores irrelevant information in the wav files. If for example the wav file is consists of 4000000 samples, it might occur that only 3500000 samples contain information, the rest is set to 0. This may create problems for other programs that are supposed to read the wav files.

- **The graphics.**

When the program run for a long period or if there are big changes in frequencies or amplitude, the graphic sometimes stops and sometimes also disappears or the signals just roam randomly. This causes the program to stop, and not respond to alarms that might occur. This bug is more common under windows 98 and windows 2000 server version. In windows XP it seems to work a lot better. Probably because Windows XP has better threading system.

Small problems that are found (Do not effect the functionality):

- **The windows.**

Some of the windows in the program cannot be moved. This can be irritating for the user and should be fixed.

- **The filenames.**

The program stores the files with a filename containing the time when the alarm occurred. This might be 15.03.56 for example. But then the file is

stored as 15_3_56. This makes the files unsorted by default. It should have the same number of figures. Another problem discovered in the filename is the # character. For examples a web browser cannot read that character, and this can cause problems. So instead, maybe it's better to call the files for example: 15_03_56_1.wav?

For more information about the debugging, please see the debugging report, created for the IRA and CNR. The report can be found as enclosure 2.

8.1.2 User manual

The user manual for the E.L.F.O. software has been written by Jan Steinar Kvilesjø, and can be found as enclosure 3.

8.1.3 Running the software on Unix/Linux platform

After several researches it seems that this is not possible at present time. Inside the E.L.F.O. receiver there is a DAQ unit created by National Instruments (as described in 6.2.1). It's that unit which is connected to the PC using the USB interface. At present time there are no drivers developed for Unix/Linux operative system. The software can be compiled for Linux.

Please see [1] for more information about running National Instrument units under Unix/Linux. This article describes some possibilities for other National Instrument units under Unix/Linux, but not for the Ni-DAQ 6020 yet.

8.2 Total E.L.F.O. system description

This section refers to section 7.2: User demands for the E.L.F.O. analyze system.

8.2.1 Development of the system

In Figure 8-1 you can see the system. First the E.L.F.O. software captures the signal for the E.L.F.O. receiver and stores the wav files in a folder named with the date they were captured (This is a built in function in the E.L.F.O. software). Once a day, the scheduler starts compressing the wav files into archives. After compressing the files, a neural network is scanning through all the files looking for known or non interesting signals. The signals that are left after the filtration are converted into jpg images and sent over to a remote location for web presentation. You can read more about the different software in section 9.2. You can also see the overall float diagram in Figure 8-2.

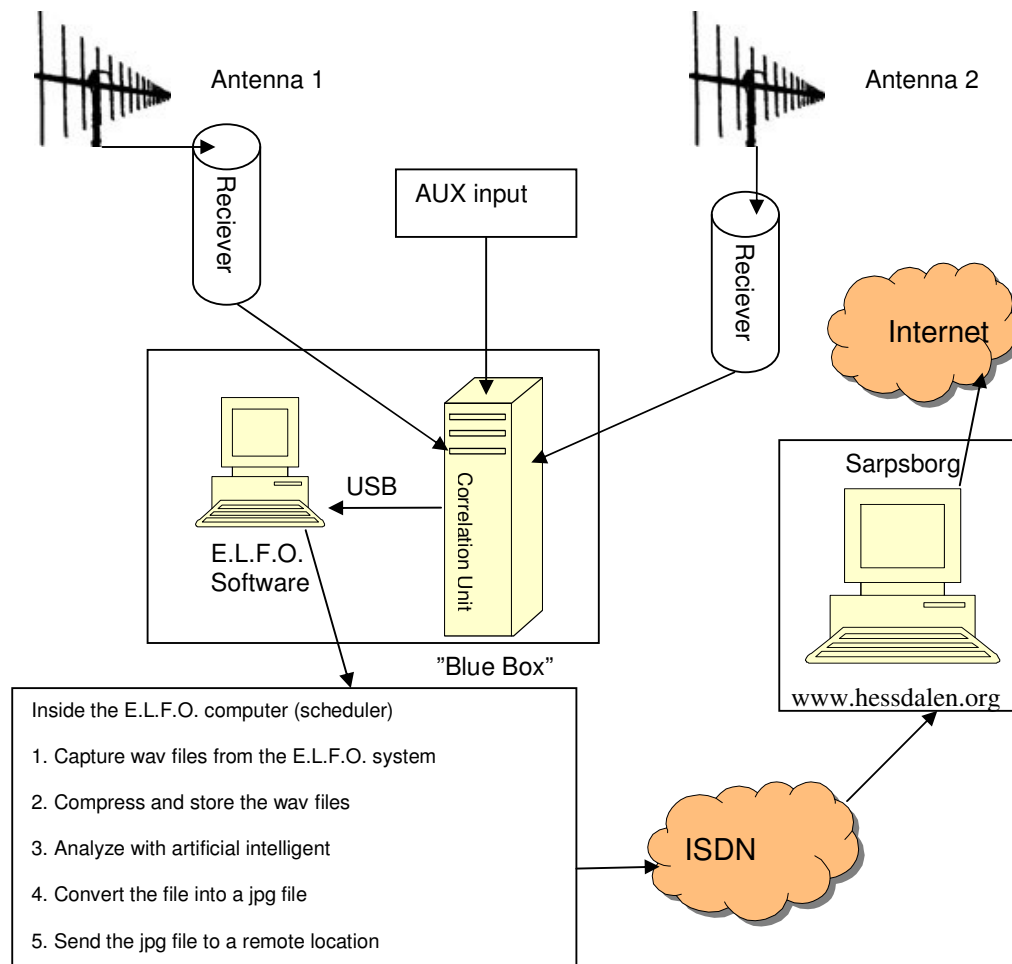


Figure 8-1: Figure of system overview.

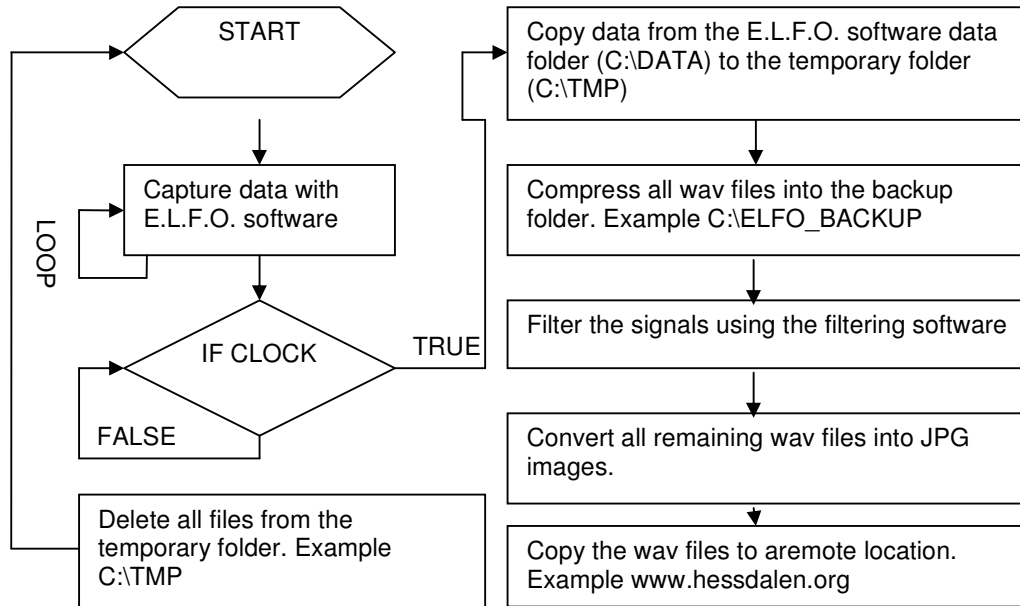


Figure 8-2: E.L.F.O. system float diagram

8.2.2 Software

The system consists of five programs. With this solution, you can easily replace some of the programs with new ones. The Recognition program is an example of this. Other benefits are that you can use some parts of this system on other systems. The wav2jpg converter is an example of this. All of them have to work together as a team to make the system fully functional.

The software used in the system is the following:

- E.L.F.O. Software (Read more in section 6.2.2)
- PSCP for Windows (Read more in section 8.2.2.3.3)
- Wav2Jpg Converter (Read more in section 8.2.2.1)
- Scheduler (Read more in section 8.2.2.3)
- Neural Network Analyze software (Read more in section 8.2.2.2)

All software developed by us is created using JAVA technology. This is because it would be easy to use the software between platforms (Windows / UNIX / Linux) depending on what platform you are running. The JAVA technology is known, not to be the fastest programming language. But since none of the programs uses much graphics the speed is maintained in a good way, and the Java technology is good language to use.

8.2.2.1 Wav2Jpg converter

This program consists of three different parts. One part is the automatic mode whose main task is to automatically convert wav files captured by the E.L.F.O. software system (see section 8.2.2.1.1). The second part is the manual mode. Here, wav files can be converted into jpg images at a users request (see section 8.2.2.1.2). The last part is the 3D display function. The program can here take an image (at users request) and display it in 3D (see section 8.2.2.1.3).

Wav2JPG Converter - Structure

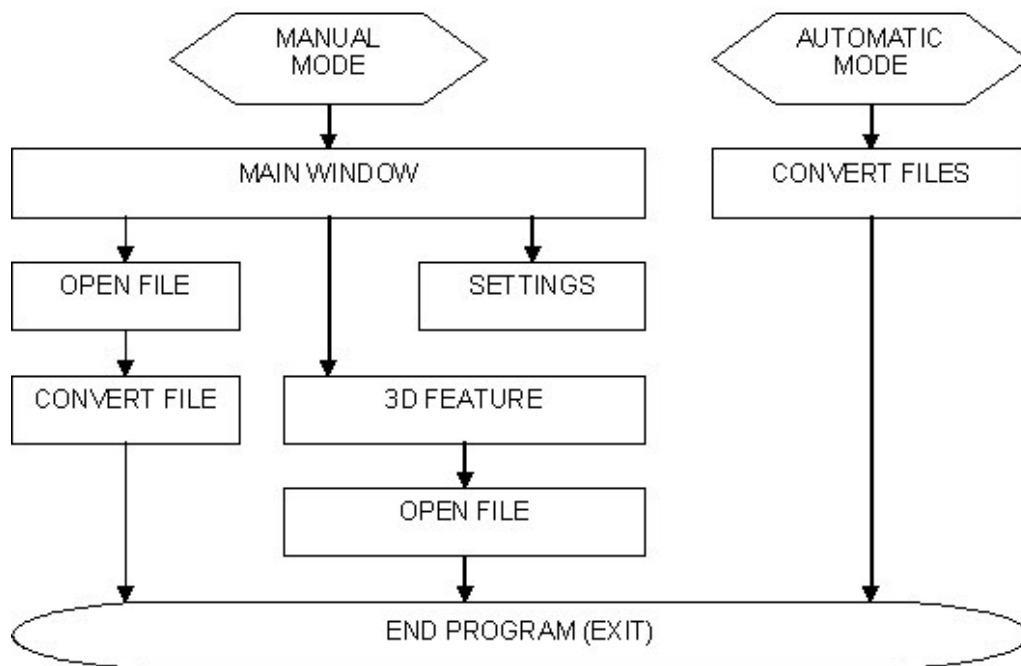


Figure 8-3: Wav2Jpg Converter – GUI overview diagram

8.2.2.1.1 Automatic Mode

In automatic mode, the user of the program can select a particular folder that he/she wants the program work in. The program takes the selected folder and scans it for wav files. Each of the wav files that the program finds are converted into jpg images and placed in a subfolder called “Processed_Images”.

If a subfolder is found, it also scans that one, and process the wav files in the same way as in the parent folder. It continues like that until all subfolders are processed.

8.2.2.1.2 Manual Mode

In manual mode, the user can select which wav file he/she wants to convert. The user selects the filename of the file to convert, and the filename of the image to store. After conversion, the image representing the wav file will be stored as the user demanded. (See more about how the program is built up in section 8.2.2.1.4).

8.2.2.1.3 3D feature

The program also has function used to display different images in 3D. This function can be useful for a better understanding of the picture created by this program. Here you can follow the timeline in the picture and see what amplitude the signal has in a specific frequency. This function also has the possibility to display other pictures in 3D, but it's meant to be used for pictures created by this program and nothing else. Remember that the picture created cannot be saved, and therefore only be viewed in this program. See the 3D plot here: Figure 8-4: 3D plot of the signal.

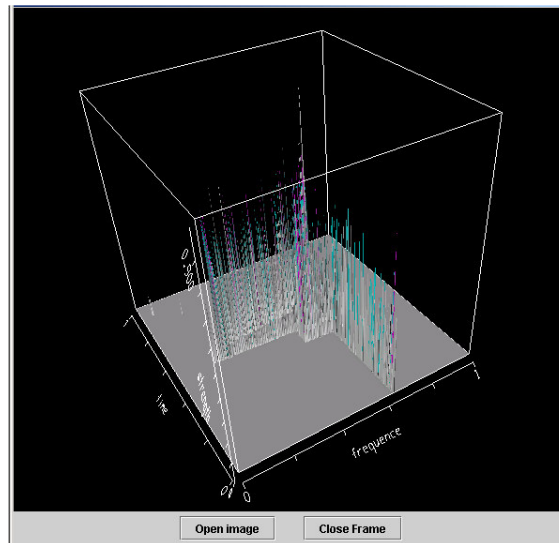


Figure 8-4: 3D plot of the signal.

8.2.2.1.4 Technical solution

In general the program uses the built in *Javax.sound.sampled API* [2] (See section 9.3.4.1) which can open and read essential information from a standard Microsoft wav file. To perform the conversion we need to select a window size to use. This window size must be the power of 2 (for example 2048, 4096 and so on). The window size can also be called number of samples to read.

This window signal is then put into an FFT algorithm (see section 8.2.2.1.4.2) which is processing the signal part and then returns a number of channels with the corresponding amplitude or signal strength. The returned signal from the FFT now has a set of values. This row of values is putted into a two dimensional array as a new row. When the entire file is processed we have a two dimensional array with the number of channels as columns, and the number of windows read from the file as rows. Every cell holds the signal strength for each window with its corresponding channel.

Now the values of each row in this array are distributed within a spectre of 16581375 different numbers (representing all possible colours in RGB using hex values. See section 8.2.2.1.4.3). Each one of those values is spitted into three different values representing the RED component, the GREEN component and the BLUE component.

After the entire two dimensional array has been processed we have three two dimensional arrays (red, green and blue). Those are put into a pre made API called *imageprocessing* [3] which creates the final JPG image.

See Figure 8-5 for the float diagram of the technical solution.

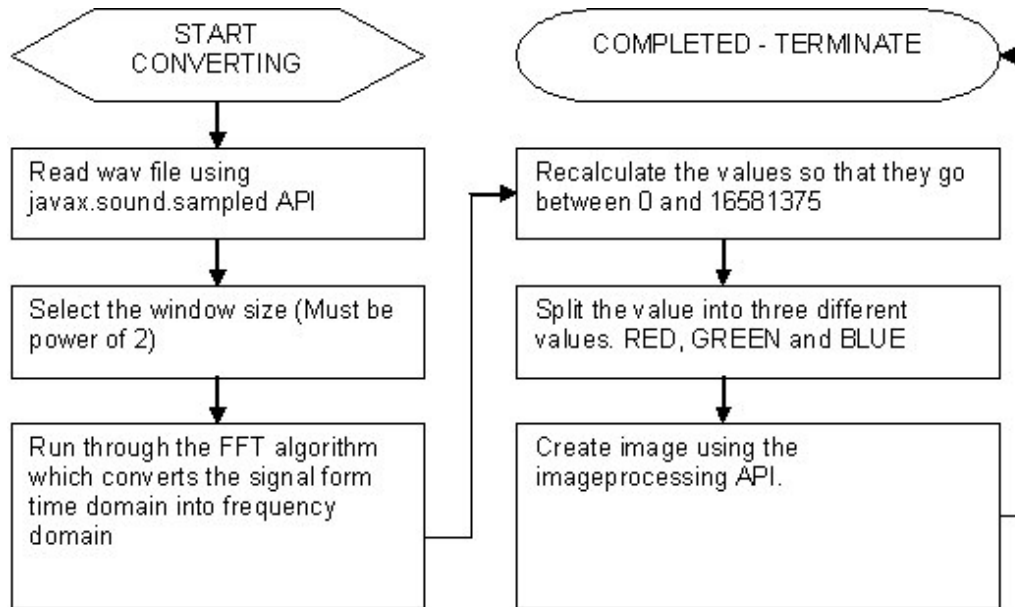


Figure 8-5: Technical solution flow diagram

For the entire code see enclosure 6.

For the java documentation see enclosure 5.

Notice: In this report we are concentrating on the idea on how to convert a sampled sound file (wav) into a jpg image, and not every programming detail.

8.2.2.1.4.1 Retrieving signal using the javax.sound.sampled API

To read the audio stream from the file we need to know where on the file to start, and how many samples to read. The number of samples to read has to be the power of 2 (2048, 4096 ...), because the FFT algorithm requires that to work.

When you have decided what sample to start reading from and how many samples to read you can start calculating the number of bytes to read, and the value coming from the wav file. This calculation can be done by following this example:

Let's say that a window size of 4096 samples has been selected and that you read the sample size from the file. The sample size can for example be 16 bits.

1. As known one byte consists of 8 bits, that means that if the wav file has 16 bits, each sample must consist of 2 bytes. This also means that the wav file is recorded in 16 bits. Now that the byte per sample is known, we can calculate the number of bytes to read from the file. This is done using this formula:

$$\text{numberOfSamples} \cdot \text{bytesPerSample} = 4096 \cdot 2 = 8192 \text{ bytes} \quad \text{Equation 1}$$

2. If the file is 16 bit the calculation of the byte value is different that it is if the file is 8 bits. This example takes the 16 bit version since that is the sample rate for the E.L.F.O. system. We must now check if the file is BigEdian or not. If the file is BigEdian the high order byte comes before the low order and for not BigEdian visa versa. The high byte is found by taking the byte value and & it with 0xff (hex), and the low order byte is found in the exact same way. Now remember that the high order and the low order can be placed differently depending on BigEdian. If the file is BigEdian or not can be read directly from the file. After converting bytes into integer values, the real sample value can be calculated by this formula:

$$\frac{(\text{highorder} \ll 8 / \text{loworder})}{2 \ll 15} \quad \text{Equation 2}$$

3. Point 3 and 4 can now be repeated for every sample (4096) until all 8192 bytes have been processed into sample values. The window (containing 4096 samples) can now be used to calculate the FFT (see section 8.2.2.1.4.2).

See enclosure 6 for the source code

See enclosure 5 for the java documentation

8.2.2.1.4.2 The FFT Algorithm

The FFT algorithm (Fast Fourier Transform) is an algorithm which transforms a time domain signal into a frequency domain signal. That means that you can go from a signal like the one shown in figure Figure 8-6, which is a standard wav file signal, to a signal represented as a power spectrum, as shown in Figure 8-7. The FFT algorithm works on windows of the original time domain signal. And that window is the one retrieved from the wav files in section 8.2.2.1.4.1. And we can continue to use the windows size of 4096 samples. (Remember that this window size must be the power of 2).

The FFT algorithm then produces two different tables of $4096 / 2 = 2048$ samples each. One of the tables consists of real numbers, and the other consists of imaginary numbers. Those numbers are calculated together after the following formula:

$$\sqrt{R^2 + I^2} = P$$

Equation 3

and returned as the FFT transformed signal of the window.

As mentioned, the return for the FFT algorithm is an array consisting of 2048 numbers. Those can be called channels in the power spectrum. Now the frequency of each channel can be retrieved.

Let's say that the signal used consists of 22000 Hz (this can be read out of the wav file directly), each of the channels returned from the FFT represents $\frac{11000}{2048} = 5,37 \text{ Hz}$

The amplitude of a frequency can now be read from the table. If you read out the value in column 4, you will get the amplitude of the frequency which is:

$$5,37 \cdot 4 = 21,48 \text{ Hz}.$$

More about the FFT transform can be found here [5]. The FFT algorithm used in the program code can be found here [6].

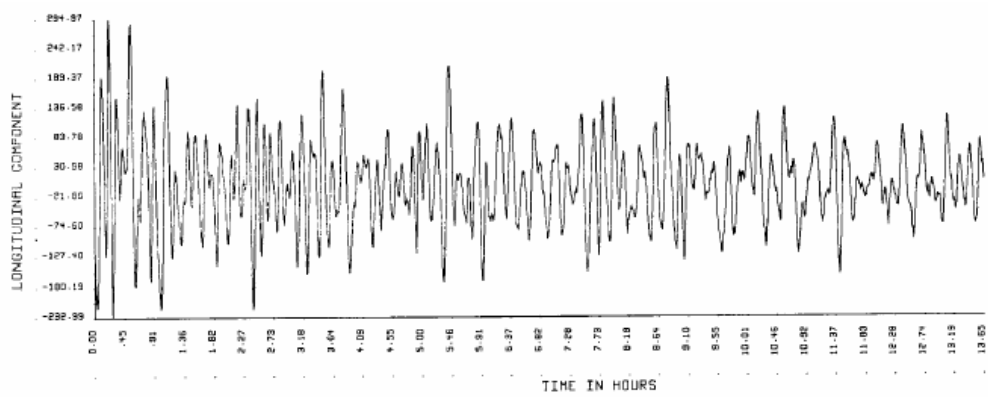


Figure 8-6: Standard wav file signal

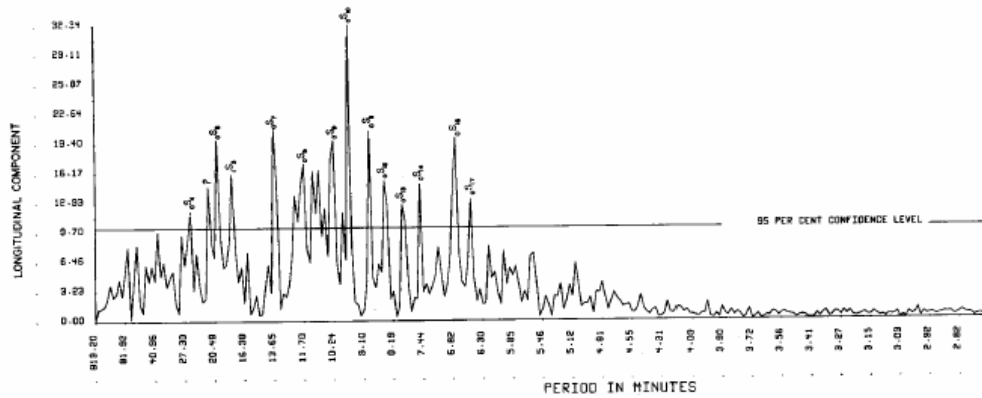


Figure 8-7: FFT processed signal for one window

8.2.2.1.4.3 Integer to RGB conversion

This part is essential to create an image. After the FFT transform of the whole wav file, we have a two dimensional array containing wav signal amplitude values. Each value in that array is assigned a new value, using this formula:

$$\frac{\text{current_value} - \text{minimum_value_in_the_row}}{\text{maximum_value_in_the_row} - \text{minimum_value_in_the_row}} \cdot 16777215$$

Equation 4

This value is the hexadecimal value that is used to create the three two dimensional arrays; red, green and blue. Which each table represent the corresponding colour.

Each value is then converted to hex value. For example if we have the value 345646, this is 0x05462E in hex. Here 05 is representing the red component, 46 representing the green component and 2E representing the blue component. Those values are converted back to standard decimal value (Example: 05 = 5, 46 = 70, 2E = 46). Those values are used to create three arrays representing the image.

This part of the program also has a filter. This means that you can set a threshold value. Every signal that is below this value will be black. This helps you get a cleaner picture without a lot of noise. If you skip the threshold you will get a picture full of different colours, and it will almost be impossible to see the interesting signal in the picture. But be careful, if the threshold value is too high you will eliminate much of the information in the picture.

One problem with using the hex value of RGB is that every colour is mixed with blue, since the blue colour is to the right in the hex value. The benefit of this scale, is that we get a high resolution on the scale with 16777215 values.

8.2.2.1.4.4 Creating the JPG image

The three arrays created by the hex conversion are being used to create the image. This is done by using the pre created API called *imageprocessing* [3]. This API has a pre designed method for creating the jpg image just by putting in the three tables; red, green blue.

8.2.2.1.4.5 Complete conversion example

Now let's illustrate the whole process with a practical example.

For this example we are using a 16 bit file with 4866048 samples totally ("newfile.wav"). We choose to use 4096 samples per window in this example. When the file is opened we can start reading the first 4096 samples. When all samples in the window are read, we have a table with 4096 columns. That table can now be putted into the FFT algorithm which returns a new array consisting only of 2048 samples.

That array is now added as a new row in a two dimensional array. Then the same thing is done with the next 4096 samples until there are below 4096 samples left. We now have a two dimensional array which has 2048 columns and $4866048/4096 = 1188$ rows.

Then the maximum and minimum value of each row is found. Let us say that the minimum value is 2 and the maximum value is 700 (those values might not be realistic, but for the example only). We can now use the equation 4 to get the new value of each of the old values. For the minimum value we get:

$$\frac{2-2}{700-2} \cdot 16777215 = 0$$

For the maximum value 700 we get:

$$\frac{700-2}{700-2} \cdot 16777215 = 16777215$$

The highest value is now the maximum amplitude shown in the picture. If we now take 16777215 and convert it into hex value, we get: 0xFFFFFFFF. That means; RED = 255, GREEN = 255, BLUE = 255, which in a jpg image will be white. If some of the values come below the threshold value, it will automatically be set to 0. This helps us remove all the noise in the signal.

When every value in the two dimensional array has been converted, we can create the image which in the case of "newfile.wav" will look like Figure 8-8. And there you can see the axis on the picture. Down; you have the time axis. And from left to right; you have the frequency axis.

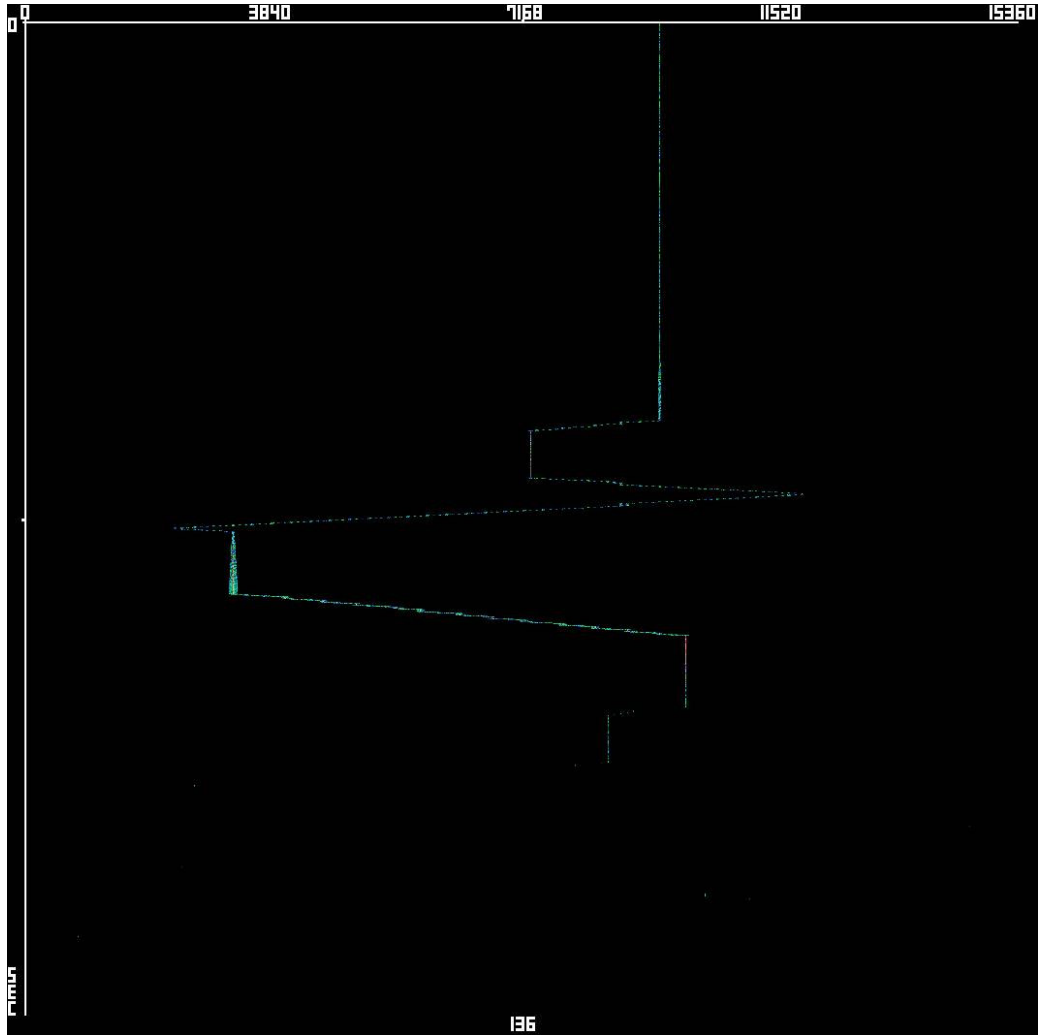


Figure 8-8 - Picture for newfile.wav

8.2.2.1.4.6 Displaying an image in 3D

In this part we are using a pre designed API called VISAD. After some modification this API can take a two dimensional array with values between 0 and 16777215 (must be in decimal format).

Those values are used to display the image as a 3D drawing with the column number as frequency, and the rows as time and the value as amplitude.

8.2.2.1.4.7 Known errors and possible improvements

There is only one error which is known at present time.

If the program is going to convert extremely big wav files, it can happen that an OutOfMemory exception occurs. This error might occur because of an unknown memory leak. But the program is tested in a lot of situations and the following conclusion has been found:

If the wav file that is to be converted has a size around 300 MB, a memory exception occurs. It also happens if the total number of wav files to convert in automatic mode has a total size bigger than 500 MB.

This problem can easily be avoided by splitting the big wav files into smaller parts, and try to avoid many big files in the same folder.

8.2.2.2 Neural network filtering software

The recognition program is used for identifying files. The program gets the input file as an argument. First the file is pre-processed and then it is run through a neural network. If the file is recognized the file will be deleted.

To train the program, you must start the program without any arguments. There is more information about how to use the program in enclosure 4.

See enclosure 5 for the source code.

See enclosure 6 for the java documentation.

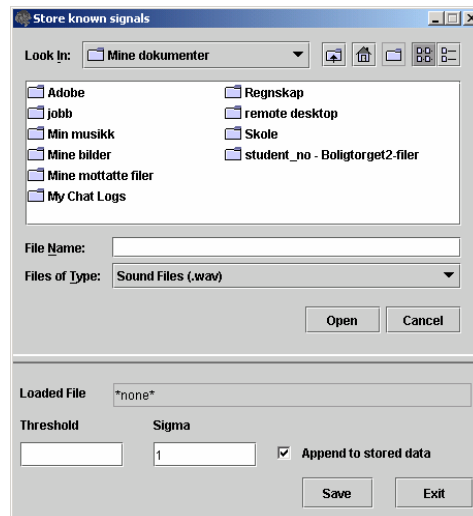


Figure 8-9 - The GUI for training the network.

8.2.2.2.1 Tested algorithms

We have tested two different algorithms which can be used for filtering out known signals from a signal stored as a wav file. One of them is the O-Algorithm, and the other is the Feed-forward Back-propagation network. Both of them are described below.

8.2.2.2.2 The O-Algorithm

Before we use the O-algorithm [9] to classify the signal, we pre-process the signal with Haar wavelets.

8.2.2.2.2.1 Pre-processing the signal with Haar wavelets

Even though a small segment of an audio signal may be unique in some sense to a particular signal, significant variations occur. To compensate for some of these perturbations, the Haar wavelet transform is used as a preprocessing tool. Haar wavelets are known to have a dampening effect to rich amplitudal variations often present in audio signal, thus presenting an average of adjacent sample values. The family of Haar wavelets $\psi_{m,n}(t)$ is described by

$$\psi_{m,n}(t) = 2^{-m/2} \psi(2^{-m}t - n) \quad \text{Equation 5}$$

$$\psi(t) = \begin{cases} 1 & 0 < t \leq 0,5 \\ -1 & 0,5 < t \leq 1 \\ 0 & \text{elsewhere} \end{cases}$$

Where $\psi(t)$ represent the mother wavelet. Here, n denotes the translations parameter and m is the amount of scaling. The application of Haar wavelets to a characteristic segment of a signal produces a set of N Haar wavelet coefficients, $w = [w_1, w_2, \dots, w_j, \dots, w_N]$, where

$$w_j = \langle x, \psi_{m,n} \rangle, \quad \text{Equation 6}$$

and $x = [x_1, x_2, \dots, x_j, \dots, x_N]$ are recorded samples of sound forming the characteristic region chosen to represent the audio signal. The translations and scaling parameters are integers chosen such that

$$m = 1, 2, \dots, \log_2(N) + 1, \\ n = 0, 1, \dots, \lceil 2^{-m} N - 1 \rceil,$$

where $\lceil \cdot \rceil$ denotes the ceiling operator. By performing this procedure for P different signals belonging to P different classes each coefficients set

$$w_p = [w_{p1}, w_{p2}, \dots, w_{pj}, \dots, w_{pN}], \quad p = 1, 2, \dots, P$$

thus gives a template of Haar coefficients representing a specific audio signal. For more information about Haar wavelets and how it works see [11].

8.2.2.2.2 O-algorithm (Technical description)

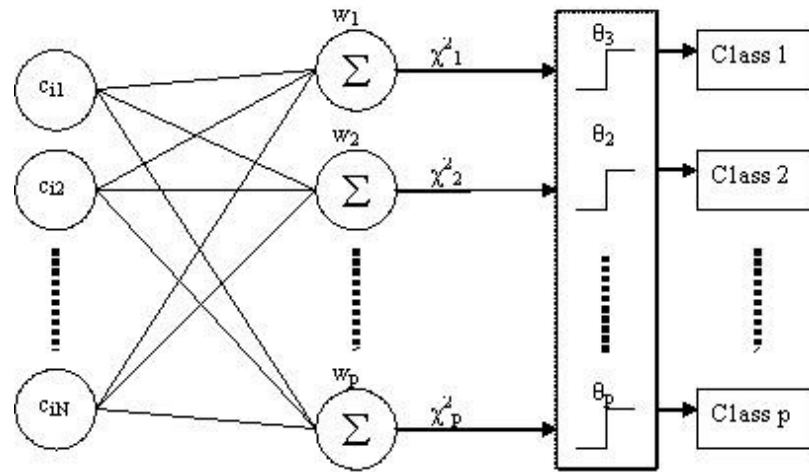


Figure 8-10 The O-algorithm formed as a neural network.

The O-algorithm can be viewed as a neural network (see Figure 8-10) with N inputs and P active neurons each performing the similarity match in (Equation 7).

The O-algorithm performs a similarity match between two set of data. Similarity between a predefined data set $a = [a_1 a_2, \dots, a_N]$ and a candidate data set $b = [b_1 b_2, \dots, b_N]$ is calculated by:

$$\chi^2 = \sum_{j=1}^N \frac{(a_j - b_j)^2}{\sigma^2} \tag{Equation 7}$$

Where σ is a scaling parameter representing the expected deviation between a and b . The two data sets a and b are considered to be similar if $\chi^2 < \theta$, where θ is a predefined threshold parameter. In practice, σ can be difficult to determine and justification of θ is used instead. There can be several predefined data sets $a_p = [a_{p1}$

$a_{p2}, \dots, a_{pN}]$ where $p = 1, 2, \dots, P$ and the O-algorithm check the similarity between b and P different data sets a_p .

8.2.2.2.2.3 Training data

The data for classification is from the EMBLA project July and August 2000. There are 13 compact discs with known signals from Hessdalen stored with the E.L.F.O. system. These CDs can be used to train both of the networks.

All the data that the O-algorithm use is first divided in windows with a window size powered of 2 and then transformed with the Haar wavelet transform (8.2.2.2.2.1). The signals must also have the same samplings rate, for example 30720Hz.

8.2.2.2.2.4 Advantages with O-algorithm

This algorithm is useful for cases where there is only one known class that the system can be trained with. In this case we have only one such classes: it's the identified signal (which is known) and the unidentified signal. The identified signal is recorded and can be used for training. The O-algorithm can then compare the unknown signal with the known signal data class and then decide if the input is a known or unknown signal.

8.2.2.2.2.5 Known problems with O-algorithm

To identify a signal the algorithm compares the input signal to the known signals. Before comparing the Haar coefficients are calculated to better describe the signal. If the sum of differences is under the threshold level (θ), the signal is identified. The algorithm takes one window at a time and compares it to a window in the known signal.

If each window in the input signal is to be compared with every window in every known signal that we got, we will get a large number. For example: We got 13 CDs with known data. 1 sample takes 2 bytes. If we got 650 Mbyte on 13 CDs, we have about 4225 million samples. The system can use a window of for example 2048

samples. This will sum up to 2 million windows which is too large if the system is running on normal PC.

A solution to this problem is to find a characteristic part of a signal to compare with. One way to find this characteristic part is you manually go through the file and look for parts that repeats it self. This is very hard because it is not so easy to find a characteristic part of the signal by looking. Another way to do this is to compare each window to each other in a file to find what window that is most frequent. The window most frequent is the characteristic window. The problem of this, is that the quiet parts are often most frequent in the data files and very often this is the result of this solution.

8.2.2.2.3 The Feed-Forward Back-Propagation algorithm

Before using the Feed-Forward Back-Propagation algorithm, it is necessary with preprocessing of the signal. FFT transform is an existing tool in Matlab and a good way to describe the signal. Matlab is also a good program for making a Feed-Forward Back-Propagation network because of existing toolbox.

8.2.2.2.3.1 Pre-processing the signal with FFT transformation

This pre-processing with FFT transformation is the same as described in 8.2.2.1.4.2

8.2.2.2.3.2 Feed-Forward Back-Propagation algorithm (Technical description)

Referring to Figure 8-11 and Figure 8-12. The network functions as follows: Each neuron receives a signal from the neurons in the previous layer, and each of those signals is multiplied by a separate weight value. The weighted inputs are summed, and passed through a limiting function which scales the output to a fixed range of values. The output of the limiter is then broadcast to all of the neurons in the next layer. So, to use the network to solve a problem, we apply the input values to the inputs of the first layer, allow the signals to propagate through the network, and read the output values.

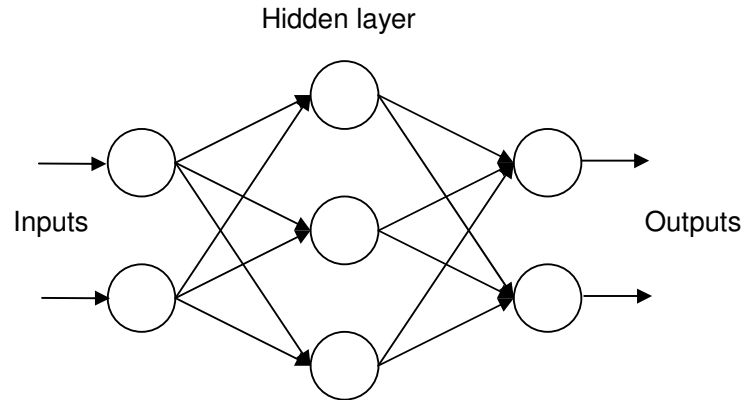


Figure 8-11: A Generalized Network. Stimulation is applied to the inputs of the first layer, and signals propagate through the middle (hidden) layer(s) to the output layer. Each link between neurons has a unique weighting value.

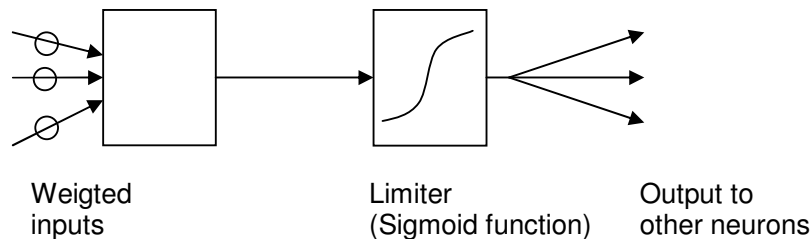


Figure 8-12: The Structure of a Neuron: Inputs from one or more previous neurons are individually weighted, then summed. The result is non-linearly scaled between 0 and +1, and the output value is passed on to the neurons in the next layer.

The Back-Propagation learning process works in small iterative steps: one of the example cases is applied to the network, and the network produces some output based on the current state of its synaptic weights (initially, the output will be random). This output is compared to the known-good output, and a mean-squared error signal is calculated. The error value is then propagated backwards through the network, and small changes are made to the weights in each layer. The weight changes are calculated to reduce the error signal for the case in question. The whole process is repeated for each of the example cases, then back to the first case again, and so on. The cycle is repeated until the overall error value drops below some pre-determined threshold. At this point we say that the network has learned the problem "well enough" - the network will never exactly learn the ideal function, but rather it will asymptotically approach the ideal function

Training algorithm:

1. Initialize weights (set to small random values)
2. Present inputs and desired output for the known signals
3. Calculate actual outputs
4. Adapt weights
5. Repeat from 2

Termination: Terminate when error is below a threshold value.

8.2.2.2.3.3 Training data

In this network we can use the same training data as we used to train the O-algorithm (see section 8.2.2.2.3).

8.2.2.2.3.4 Advantages with Feed-Forward Back-Propagation algorithm

This is a network that works fine with large training data. The feed forward back propagation network trains weights that calculate the right output. These weights are corrected in a training progress. In this progress, we use a set of known data and calculate the output. If the output is wrong, the weights are corrected. This means that this kind of network can be used on big amount of data, since only the weights are stored. Then when recognizing, the computer only has to use those weights instead of comparing to hundreds of numbers.

8.2.2.2.3.5 Problems with feed-forward back-propagation network

The big problem for this algorithm is that the network usually must have data from known signals and data from unknown signals. This algorithm works well if we can train the network with data from both of the classes. Since we don't have any data from the unknown signals, we must train the network with only one class.

8.2.2.2.4 Algorithm conclusion

The algorithm that we suppose as best of the two algorithm is the O-algorithm. That is because of the possibility to train the network with only one known class. The O-algorithm will work if we find characteristic signals for the known signals and then can reduce the amount of window for comparison. The feed forward back propagation network will not work well because of the unknown class that we are interesting in.

8.2.2.2.5 Software structure and use of the chosen algorithm

We have considered using the filter on the inspire system. The filter can be further developed to read windows from the input port on the soundcard, and store the windows that are not identified. Since the filter doesn't work as well as we have hoped, we have not developed the system to have this opportunity. With the O-algorithm the filter will work to slow if the amount of data is too big.

8.2.2.3 Scheduler

The scheduler is a simple Java programs created to manage the different tasks in the E.L.F.O. analyze system. Its primary task is to start the different operations in a static order. The program runs with no graphics, except text output to the operating system consol. For the scheduler to run properly, you need to have the following programs installed (All of them are bundled in the same installation package):

- Wav2Jpg Converter (Section 8.2.2.1)
- PSCP for Windows Consol (Section 8.2.2.3.3)
- The Neural Network Analyze Software (Section 8.2.2.2)

8.2.2.3.1 Program structure

As mentioned above, the program follows a particular order (see Figure 8-13 for illustration, float diagram).

The program has a built in timer which prevent it from running if the clock is not between for example 01:30 and 02:00. This is because the E.L.F.O. software stores files in directories with the date on. When the clock passes 00:00, the E.L.F.O. software starts on a new folder. But if the E.L.F.O. starts capturing files at 23:59:59, it will store in yesterdays folder until the next time it starts capturing. There for the time is set to 01:30, to prevent moving files which are not yet is finished, but you have the ability to change that time in the setup file which you can read more about in 8.2.2.3.2. But remember that the scheduler software does not stop if the trigger time is not reached; it only waits until the clock passes the trigger time. You can say that it runs in an infinite loop, which only can be cancelled by terminating the process manually.

So, first it starts searching the folder where the E.L.F.O. software stores its wav files (default: c:\data). Then it moves all the files in the folders, which don't have today's date, with all subfolders and contents to a temporary location (for example C:\TMP, selected by the user).

The scheduler then starts the ZIP routine, which is built in to the scheduler. All the wav files in the different folders in the temporary directory are now compressed. If for example C:\TMP contains the folder 28_4_2003 with four different wav files. An archive called 28_4_2003.zip is created, and all the files in that folder are compressed into the archive.

After the compressing is done it's time to analyse the wav files. The scheduler starts the analyze software. You can read more about the analyze software in section 8.2.2.2. The user can also choose not to run the analyze software. This must be done in the setup file, which you can read more about in section 8.2.2.3.2.

Now, the scheduler starts the Wav2Jpg converter. The wav2jpg converter creates images of the remaining wav files after the neural network filtration. Those files are stored in a subdirectory called "Processed_Images". Those folders are now renamed to its parent folders name. That means that if the parent folder is named 25_12_03 the Processed_Images folder is renamed to 25_12_03.

All the wav files are also renamed in this part of the scheduler. All wav files captured by the E.L.F.O. have a filename as the following: 12_14_23_#1.wav. In the scheduler the # character is replaced with "NR" instead. This is because the web browser has problems displaying filenames containing that particular character (#).

And now it's time to start the PSCP software with the following command:

```
pscp.exe -r -q -pw <password> <dir_to_copy_from>  
<username>@<server>:<dir_to_copy_to>
```

When every images in the temporary folder has been sent to a remote location all the files in the temporary folder is deleted. Since all of the wav files have been backed up in another folder, it's no problem deleting the files.

Now the scheduler has finished its work, and it goes back to sleep until the next time the clock passes 01:30. Notice that the scheduler do not exit on completion, it just sleeps. To exit the program it must be terminated manually. It's also important notice that if an error occurs somewhere in the scheduler, the work is aborted, and the scheduler goes back to start and then to sleep. It does not exit.

Please see the float diagram in Figure 8-13.

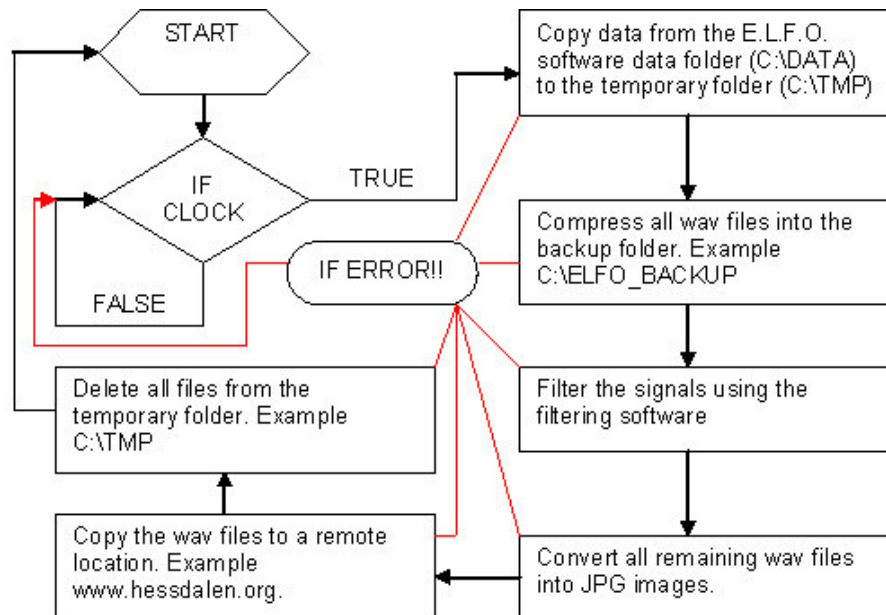


Figure 8-13: The scheduler loop

For the entire program code and java documentation, please see enclosure 5 and 6.

8.2.2.3.2 Program usage

Usage of the program is quit simple. The only thing you have to watch out for is that the **scheduler.setup** file is correctly configured. The file should look like this:

```
C:\DATA                - The E.L.F.O. files folder
C:\TMP                  - The temporary folder
C:\ELFO_BACKUP         - The backup folder
hessdalen.org          - The server to copy jpg files to
Italia                  - The username on the server
Password                - The password on the server
/home/hessdalen/html/italia/images/ - The remote folder on the server.
HOURE                  - The heure to start the scheduler
MIN                    - The minute to start the scheduler
FILTER                 - Run the filter or not (0 or 1),
```

It's made with a setup file for more flexibility. The user can now change the different setting without recompiling the source code.

The password is not encrypted, but since the software is only going to be on a local computer it will not be a security threat.

The program is started by running the **schedulerJava.exe** file.

8.2.2.3.3 Uploading files to remote location

This task is done by a tiny program called PSCP [8]. This program works as the Unix/Linux version of SCP (Secure Copy).

To make this program work, you first have to get a cryption key. This is very important, since the scheduler doesn't get the key automatically, and that will make it hang. To get that connection key you can simply run a small test transfer to the remote system you are going to use. Try for example the following command: `pscp -r <testfile> username@server:<remote folder>`.

This command will ask you if you would like to store a server key. Please answer yes, and you are ready to use the pscp program in the scheduler.

Read more about PSCP at [8].

8.2.2.3.4 Compressing files

The compression is done by a built in Java Zip routine, which can operate in Unix/Linux as well as Windows. The routine is compressing all selected files into a archive of a desired name. The compression is set to maximum, to save space on the hard drive. The compressed file will be a standard .zip file which can be opened and viewed in any zip compatible program.

Various testes have indicated that the wav files are compressed up to 7 times using this algorithm. That means that if a file is 15 MB of size, it's compressed down too

$\frac{15}{7} = 2,14$. And that helps a lot, when it comes to saving space on the disk.

8.2.3 Webpage

The web page is created on www.hessdalen.org/italia. This page is created to display all interesting signals coming for the E.L.F.O. system in Hessdalen, Norway. The web page is created in a way so that when the scheduler has uploaded new jpg files, they will automatically become visible on the web site. This page only contains the two dimensional pictures and no 3D features.

The pictures are sorted into different catalogues, having the creation date as name. Every file has its creation time and from which antenna (1 or 2) the signal was recorded.

The web page is created in the PERL language.
Source code can be found in enclosure 7.

8.2.3.1 Web-Page structure

The web page is spitted into three parts. First there is a page which is displaying all the directories stored in the image directory on www.hessdalen.org. Those directories are displayed as the following example: 12 of April 2003. Each of the directories is clickable and will take you to a new page which displays the content of the folder.

In the content display page you will receive a list with all the files in the selected directory. On each file you will get information on when the file was captures (clock), remember that the folder holds the date when the file where captured. You will also get information on which antenna the signal where captured on (1 or 2). And last you will get the filename. A directory on the web page could look something like this:

Alarm occurred:	On antenna number:	Captured Picture:
11 : 21 : 39	1	11_21_39_NR1.jpg
11 : 21 : 39	1	11_21_39_NR1_2.jpg

When clicking one of the created pictures. The selected picture will show in a new page.

You can see the different scripts in enclosure 7.

8.2.3 Remote administration

The E.L.F.O. system must have the opportunity to be administrated from a remote computer. There are several programs for remote administration that are already developed. Many of these programs can be used for this system.

We have tried to run the E.L.F.O. software on different windows system (Section 8.1). The software runs best on Windows XP. This operating system has remote administration built into the system. This is not activated by default, but you can read how to set this up at this site: [12]. The client software is also included in XP, but if you want to use a different Windows system, you can download the client at this site [12]. This program is found in the start menu, under "Start Menu-> Programs-> Accessories-> Communications".

This program gives the user a graphical view of the remote machine and also has the possibility to transfer files from the client to the host computer.

The logon screen for this program can be seen in Figure 8-14.

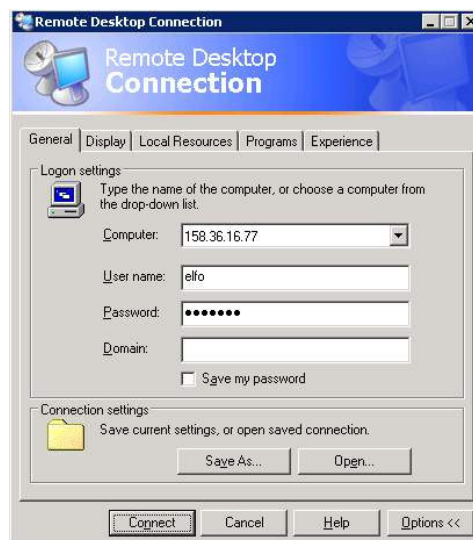


Figure 8-14 - The logon screen to remote desktop

8.3 Conclusion

Our projects, E.L.F.O. analysis, consist of further develop on the E.L.F.O. system. Our system compressed and takes backups of all the wav files that are stored and then analyse the signal. If the signals is a known signal the files is deleted. If the signals are unknown the signals are converted to images. All the images of unknown signals are then sent to www.hessdalen.org for presentation on the web. All the system goes automatically and can run on a computer in Hessdalen.

The problem in our system is the filtering algorithm in the neural network. We have tried two different algorithms, the O-algorithm and Feed-Forward Backpropagation network. In the O-algorithm the problem is to find parts in the signal that are characteristic for the whole signal. Without characteristic parts the amount of windows are too big, and the O-algorithm will work slowly and not well. In the Feed-Forward Backpropagation network the problem is that we don't have the unknown signals and then we only have data from one class to train the network with.

The algorithm that we suppose as best of the two algorithm is the O-algorithm. That is because of the possibility to train the network with only one known class. The O-algorithm will work if we find characteristic signals for the known signals and then can reduce the amount of window for comparison.

9 Project scheme

9.1 Milestone chart

See enclosure 1.

9.2 Activity and responsibility chart for the project

9.2.1 Activity chart

R = Responsible
 C = Carries out
 I = To be informed
 A = Approve

FAT TEXT = Main tasks

Act. Nr.	Activity	Jan Steinar Kvilesjø	Lars Martin Dobbe	Christian A. Bråthe	Contractor: Stello Montebugnoli	Councilor: Erling P. Strand
		1	2	3	4	5
<i>1</i>	<i>Reports</i>			<i>R</i>		
1.1	Pre project report	C	C	RC	IA	IA
1.2	System development report	RC	C	C	IA	IA
1.3	Final report	C	RC	C	IA	
<i>2</i>	<i>Web Site</i>			<i>R</i>		
2.1	Layout	C		RC		
2.2	Contents	C		RC		
<i>3</i>	<i>Needed equipment</i>		<i>R</i>			
3.1	E.L.F.O. system hardware		RC		C	
3.2	E.L.F.O. system software (existing)		RC		C	
3.3	Other equipment (ex. own)	RC	RC	RC		
<i>4</i>	<i>Existing E.L.F.O. System</i>	<i>R</i>				
4.1	User manual of existing software	RC	C			
4.2	Debugging the existing software		C	RC		
<i>5</i>	<i>E.L.F.O. Analyzing system</i>		<i>R</i>			
5.1	The Neural Network	C	RC	C		
5.2	WAV2JPG converter			RC		
5.3	Scheduling	RC				
5.4	Compressing	RC				
<i>6</i>	<i>EXPO</i>	<i>R</i>				
6.1	Presentation	RC		C		
6.2	Stand	C	RC			
6.3	Equipment		C	RC		
6.4	Catalogue		RC			

9.2.2 The responsibility distribution

The main responsibility is shown only.

Jan Steinar Kvilesjø:

1. System development report
2. Own or other needed equipment
3. User manual for existing E.L.F.O. system
4. EXPO presentation
5. Scheduling
6. Compressing WAV files

Lars Martin Dobbe:

1. Final report
2. Existing E.L.F.O. system hardware
3. Existing E.L.F.O. system software
4. Own or other needed equipment
5. The Neural Network
6. EXPO stand
7. The EXPO catalogue

Christian André Bråthe:

1. Pre project report
2. Web site layout
3. Web site contents
4. Own or other needed equipment
5. Debugging the existing E.L.F.O. system
6. WAV2JPG converter
7. Needed EXPO equipment

9.3 Project budget (sketch)

Travel costs

Cost for each kilometre	kr	2,41
Total amount of kilometres		4800,00
Total travel cost	kr	11 568,00

Stand material

Decoration	kr	806,00
Refreshments	kr	403,00
Clothing	kr	1 612,90
Total stand cost	kr	2 821,90

Salary

Cost for each hour	kr	403,00
Number of persons		3
Number of hour for each person		375
Total cost for each person	kr	151 125,00
Total salary cost	kr	453 375,00

Total project cost excluded tax	kr	467 764,90
Tax		24 %
Total project cost included tax	kr	580 028,48

In EURO (€) that makes:

77337,13

10 Abbreviasjons survey

Abbreviation	Stands for / is
E.L.F.O.	Extreme low frequency observer
JPG	A compressed image format. Most common used on web graphics
SIV	Serndip IV format. Developed on Barkley university, but probably only used by IRA today.
WAV	Waveform sound format. The simplest way to store audio.
CNR	National Research Council
IRA	The Institute of radio astronomy
USB	Universal Serial Bus (common on all new PC's)
kHz	Kilo Hertz. 1000 Hertz
ELF	Extremely low frequencies
VLF	Very low frequencies
ADC	Analog to digital converter
CVI	Technology created by National Instruments. It is C language for Virtual Instruments.
ZIP	A compressed package which can be used on any file format
GB	Giga Byte. Used to describe hard drive size.
dBm	Signal strength. $10 \cdot \log[1000 - \text{power_in_milliwatts}]$
FFT	Fast Fourier Transform
DAQ	Digital Acquisition

11 Sources

- [1] www.ni.com/linux - DAQPad-6020 under Linux.
- [2] <http://java.sun.com/j2se/1.4.1/docs/api/javax/sound/sampled/package-summary.html> - javax.sound.sampled.
- [3] <http://www.ia.hiof.no/bildeb/imageprocAPI/> - Imageprocessing API.
- [4] <http://mindprod.com/jglossendian.html> - Little and Big Endian
- [5] <http://www-2.cs.cmu.edu/afs/andrew/scs/cs/15-463/pub/www/notes/fourier/fourier.pdf> - About the FFT algorithm.
- [6] <http://ling.upenn.edu/~tklee/dsp/FFTDemo.html> - Code for the FFT.
- [7] <http://www.ssec.wisc.edu/~billh/visad.html> - Visad (3D in Java).
- [8] <http://www.chiark.greenend.org.uk/~sgtatham/putty/> - PSCP information
- [9] Martin Kermit, Åge J. Eide, Audio signal identification via pattern capture and template matching
- [10] http://www.ni.com/pdf/products/us/3daqsc215-218_190_185-186_221-228_233-237.pdf - Ni-DAQ 6020 USB Device
- [11] Rafael C. Gonzalez, Richard E. Woods, Digital Image Processing
- [12] <http://www.microsoft.com/windowsxp/pro/using/howto/gomobile/remotedesktop/default.asp>
- Remote Desktop in Windows XP

12 Picture references

Figure 5-1: Hessdalen station	7
Figure 6-1: Test setup overview	8
Figure 6-2: E.L.F.O. system overview.....	9
Figure 6-3: Correlation unit overview	10
Figure 6-4: The E.L.F.O. correlation unit	11
Figure 6-5: E.L.F.O. GUI structure	12
Figure 6-6: Basic E.L.F.O. software user interface.....	14
Figure 8-1: Figure of system overview.....	20
Figure 8-2: E.L.F.O. system float diagram	21
Figure 8-3: Wav2Jpg Converter – GUI overview diagram.....	23
Figure 8-4: 3D plot of the signal.	24
Figure 8-5: Technical solution float diagram.....	26
Figure 8-6: Standard wav file signal.....	29
Figure 8-7: FFT processed signal for one window.....	29
Figure 8-8 - Picture for newfile.wav.....	32
Figure 8-9 - The GUI for training the network.	34
Figure 8-10 The O-algorithm formed as a neural network.	36
Figure 8-11: A Generalized Network.	39
Figure 8-12: The Structure of a Neuron.....	39
Figure 8-13: The scheduler loop.....	44
Figure 8-14 - The logon screen to remote desktop.....	48

13 Enclosure Index

Enclosure number:	Title:
1.	Milestone chart
2.	E.L.F.O. Software: Debug Report
3.	E.L.F.O. Software: User Manual
4.	User Manual
5.	API documentation
6.	Source Code
7.	www.hessdalen.org : Web Site Scripts (Source Code)