

Graphical User Interface for the Digital Receiver Used in the LOIS Project

Petter Andersson
Patrik Berglund

<p>Organisation/ Organization VÄXJÖ UNIVERSITET Matematiska och systemtekniska institutionen Växjö University/ School of Mathematics and Systems Engineering</p>	<p>Författare/Author(s) Petter Andersson Patrik Berglund</p>	
<p>Dokumenttyp/Type of document Examensarbete/ Diplomawork</p>		
<p>Titel och undertitel/Title and subtitle Graphical User Interface for the Digital Receiver Used in the LOIS project</p>		
<p>Sammanfattning (på svenska) Projektets syfte var att implementera ett användarvänligt GUI till den digitala mottagaren som används med antennerna i LOIS projektet. Applikationen ska möjliggöra för användaren att konfigurera den digitala mottagaren, göra beräkningar på inkommande data och visualisera resultaten. Tre funktioner är implementerade; estimering av spektrum med FFT, estimering av riktning på inkommande elektromagnetisk våg och dess polarisation. Resultatet av dessa beräkningar visualiseras i två separata fönster. Lokaloscillatorns frekvens och basbandsbandbredden kan enkelt modifieras och den digitala mottagarens samtliga åtta kontrollord kan ändras av användaren. Data kan också sparas som en Mx6 matris i en fil som kan importeras till Matlab[®].</p>		
<p>Nyckelord Digital mottagare, LOIS, LOFAR, GUI</p>		
<p>Abstract (in English) The objective of this project is to implement a Graphical User Interface (GUI) for the Digital Receiver which is used together with the antennas in the LOIS Project. The application should enable the user to configure the Digital Receiver, make calculations using the data and visualize the results. The GUI should be user-friendly. Three functions are implemented: Spectrum estimation using FFT, estimation of the direction of arrival and the polarization state of the incoming electromagnetic wave. The results of these calculations are visualized in two separate windows. Local oscillator frequency and bandwidth can easily be modified and all eight of the digital receiver control words can be changed by the user. The data can also be saved to a Mx6 matrix in a file which can be imported to Matlab[®].</p>		
<p>Key Words Digital Receiver, LOIS, LOFAR, GUI</p>		
<p>Utgivningsår/Year of issue 2003</p>	<p>Språk/Language English</p>	<p>Antal sidor/Number of pages 54</p>
<p>Internet/WWW http://www.msi.vxu.se</p>		

Acknowledgment

We would like to thank Anders Haggren and Jonas Lundbäck for giving us this project when the one we planned to work with was cancelled.

We would also like to thank Jonas Lundbäck for all help and support during the project and Professor Sven Nordebo for making it possible for us to complete our project and improve the application further more.

Contents

1 Introduction.....	2
2 Equipment and Development Tools	3
2.1 The Hardware Chain.....	3
2.1.1 The Magnetic Tripol Antenna	3
2.1.2 The Digital Receiver	4
2.1.3 Communication Between User and the Digital Receiver	5
2.2 Qt - The Graphical User Interface Development Tool.....	6
2.3 OpenGL.....	7
3 Algorithms	7
3.1 FFT – Fast Fourier Transform.....	7
3.2 Direction of arrival for the incoming signal.....	8
3.3 Polarization	8
4 Digital Receiver Software.....	9
4.1 Naming of Variables, Functions and Files.....	9
4.2 The Main Window.....	9
4.2.1 UDP Communication.....	10
4.2.2 Main Settings Window.....	10
4.3 The Spectrum Window	11
4.3.1 Spectrum Settings Window	11
4.4 Direction and Polarization Window	12
4.4.1 Direction and Polarization Settings Window	13
5 Discussion.....	14
5.1 Future work.....	14
6 References.....	15

Figures

Figure 1 – The Hardware chain	3
Figure 2 – Magnetic Tripol Antenna.....	4
Figure 3 – The Digital Receiver.....	4
Figure 4 – How Qt provides an encapsulation of the local operating system	7
Figure 5 – The Poincaré Sphere.....	8
Figure 6 – The Main Program Window	9
Figure 7 – The Main Settings Window	10
Figure 8 – The Spectrum Window	11
Figure 9 – The Direction and Polarization Window	12

Tables

Table 1 – The control words used to configure the DDC.....	5
Table 2 – Commands used to communicate with the digital receiver	6
Table 3 – The structure of one UDP package sent by the Digital Receiver.....	6
Table 4 – The spectrum settings window	12

Appendix

Appendix A – Source Code

1 Introduction

Radio telescopes of the old days were large steel antennas, which were slow to move and could only scan one direction at a time. The new generation is built up with a large number of small dipole antennas, clustered together in a high-bandwidth, fibre optic network. No moving parts will be needed on the antennas, since all beamforming and calibration will be done electronically. The LOFAR (Low Frequency Array) project will have approximately 13000 antennas spread over an area with a diameter of 400 km. This large scale antenna array will probably be located in the Netherlands, but various locations in the USA, Australia and Sweden are being evaluated [1].

The LOIS (LOFAR Outrigger In Scandinavia) project was initiated when LOFAR invited Scandinavian scientists within the research areas of space, climate, radio and IT for discussions and co-operation. LOIS will build up a network of antennas around Växjö in the south of Sweden. Since LOIS will use new technology in almost all parts, it will create research possibilities for many different areas, e.g. space, high speed network communication, data storage, distributed systems and smart antennas [2].

The hardware used in this project is the LOIS prototype digital receiver and a prototype magnetic tri-pole antenna. To configure this digital receiver and make calculations of the collected data a user interface of some kind must be used.

The objective of this project is to develop a graphical user interface that can be used to control the digital receiver. It should have the following specifications:

- Enable the user to specify the code words that controls the functionality of the digital receiver.
- Offer an easy way to modify the local-oscillator frequency and bandwidth.
- Start and stop the sampling of the three channels of the digital receiver.
- Saving data samples to a file which should contain a $M \times 6$ matrix according to Matlab[®] syntax.
- AM-demodulation.
- Calculations should be made in real time and the results should be visualized.

2 Equipment and Development Tools

2.1 The Hardware Chain

In this section, we will describe the hardware that is used in the project. Figure 1 displays an overview of the chain.

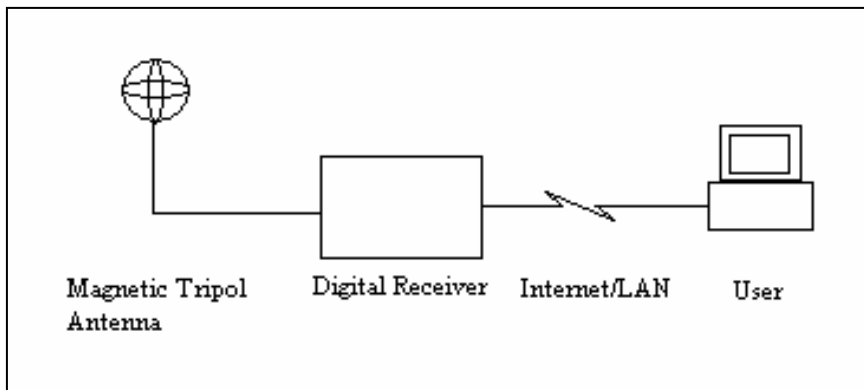


Figure 1 – The Hardware chain

The electromagnetic wave impinges on the Magnetic Tripol Antenna. The three antenna elements are each connected to the Digital Receiver on separate inputs. In the Digital Receiver the signals on the three channels are sampled, filtered and combined in to UDP packages. These Packages are sent to the user over the Internet or a LAN and can be received using the GUI developed in this project.

2.1.1 The Magnetic Tripol Antenna

The antenna used is a magnetic tripol antenna consisting of three magnetic dipoles. This is constructed using three circular elements that are mounted in three different planes that are orthogonal to each other. The elements are labeled X, Y and Z which refers to the components of the magnetic flux density field $\mathbf{B}(\mathbf{r}, t)$ that the antenna elements are measuring, see figure 2. The current induced by the magnetic flux density field in one antenna element is proportional to the voltage from the antenna amplifier. The amplifier has approximately 10 dB gain. The antenna elements are connected to the three separate inputs on the digital receiver. This enables the receiver to measure all three components of the magnetic flux density field.

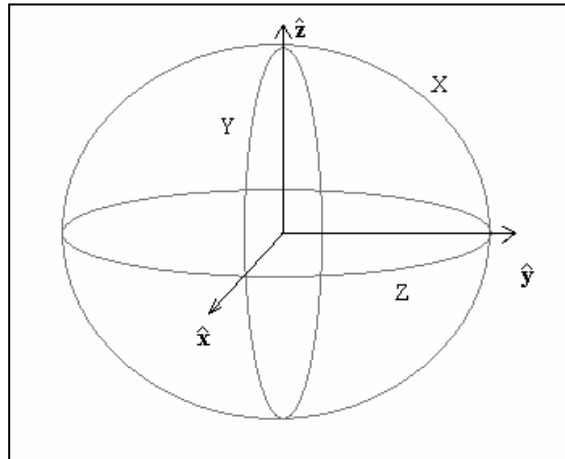


Figure 2 – Magnetic Tripol Antenna
The antenna has three elements that measures the components of the magnetic flux density field

2.1.2 The Digital Receiver

The Digital Receiver consists of several blocks, but its main block is a digital down converter (DDC). The different parts are described the following text and in figure 3. The digital receiver was built by Walter Puccio, Uppsala University, Sweden.

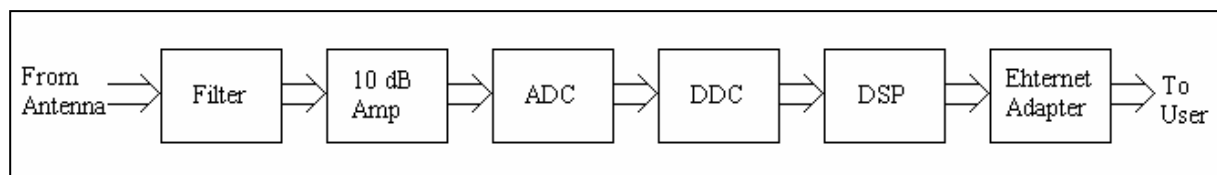


Figure 3 – The Digital Receiver
The RF- signals from the three antenna elements are first filtered and than amplified before entering a AD converter. The DDC down-converts the signals to baseband signals and the DSP combines samples from the three channels into UDP packages. These packages are sent to the user using the Ethernet adapter.

From the input to the DSP the receiver is built up by three channels which are identically constructed and therefore we will only describe one of these channels. The RF- signals with a maximum input power of 21dBm is filtered using a 15 order Butterworth low-pass filter with a cross-over frequency at approximately 10 MHz. After the low-pass filter the RF- signal is amplified by a 10dB amplifier.

The analog-to-digital (ADC) converter is a AD6644-40 from Analog Devices. It is capable of 40MSPS with an accuracy of 14 bits. It achieves a 100dB multi tone spurious-free dynamic range through the Nyquist band with a signal-to-noise ratio (SNR) of approximately 74dB.

The Digital down Converter (DDC) is a HSP50016 made by Intersil. It is a single chip synthesizer, quadrature mixer and low-pass filter. The DDC's main functions are to extract a selected frequency band from a RF- signal and down-convert it to a baseband signal. A local oscillator produces the sinusoidals used in the DDC to make a quadrature demodulation on the signal. By changing the phase increment angle the frequency of the local oscillator is modified and by doing this the center-frequency of the baseband signal can be set.

The internal filter chain is made in two parallel lines, one for the real part Q of the signal, and one for the imaginary part I . These chains are constructed by one High Decimation Filter (HDF) in series with one FIR filter. The HDF decimates the baseband signals with a factor R (the HDF Decimation Rate). In the five decimation stages of the filter the signal's LSBs are lost due to truncation. Therefore the signals have to be shifted before entering the HDF. The shift factor is a function of R . The gain of the HDF varies with R . If R is an even power two, the amplification is 1 otherwise it is between 0.5 and 1. The amplification is compensated by a scaling multiplier located after the HDF so that the amplification remains constant.

To sharpen up the total filter chain, a FIR filter for anti-aliasing is located after the HDF section. The FIR filter consists of 121 taps and has a frequency function that depends on R . The bandwidth of the baseband signal is a function of R and the FIR filter shape. Therefore R is used to set the bandwidth of the baseband signal.

An Output Formatter is located in the end of the chains to make it possible to format the output data in a preferred way. The output formatter set e.g. the timing between the I and the Q output.

To completely configure the DDC, there are eight 40 bit control words in which bits 39-37 are the address of the control word and bit 36 – 0 contains data. Table 1 shows some of the functions of the 8 words.

Address	Function
000	Used only to update the other seven control words.
001	Controls the phase generator, which sets the local oscillator frequency.
010	Sets the maximum phase increment. Normally set to 0
011	Sets the phase generator output time slot length. Normally set to 0.
100	Sets the HDF gain compensation
101	Sets the HDF Decimation counter preload ($R-1$) and the scale factor.
110	Sets the input and output formats.
111	Controls among other things the FIR filter's accumulation. Normally set to 0.

Table 1 – The control words used to configure the DDC

Of these words only number one, four and five are used regularly in our software [3].

The baseband signal from the DDC is passed to a TMS 320C50PQ DSP made by Texas Instruments. The DSP builds a UDP package which consists of a package number and 122 samples from each of the three channels. This package is then passed to the Realtek Ethernet interface which transmits the package.

2.1.3 Communication Between User and the Digital Receiver

The Digital Receiver has its own IPv4 address stored within an EPROM and can therefore easily be accessed over the Internet using the UDP protocol. The DSP recognizes four commands: ID, SA, SC and SM. The functions of these commands are shown in table 2.

Command	Function
ID	Makes the digital receiver store the IP address of the sender as the address to use when data is sent.
SA	Makes the digital receiver start sending data. Must be preceded by a ID command.
SC	Makes the digital receiver stop sending data.
SM	Is of the form SMXXXXXXXXXX, where a X represents a hexadecimal value. Every time a DDC control word should be changed a SM command must be sent.

Table 2 – Commands used to communicate with the digital receiver

The Digital Receiver sends the data in UDP packages as shown in table 3. All values including the package number are 16 bit values (2 bytes). 122 samples from three channels with two values each makes $122 \cdot 3 \cdot 2 \cdot 2 = 1464$ bytes and then the two bytes of the package number is added which makes the total package size 1466 byte.

Sample 1	Re{CH1}	Im{CH1}	Re{CH2}	Im{CH2}	Re{CH3}	Im{CH3}
Sample 2	Re{CH1}	Im{CH1}	Re{CH2}	Im{CH2}	Re{CH3}	Im{CH3}
...
Sample 121	Re{CH1}	Im{CH1}	Re{CH2}	Im{CH2}	Re{CH3}	Im{CH3}
Sample 122	Re{CH1}	Im{CH1}	Re{CH2}	Im{CH2}	Re{CH3}	Im{CH3}

Table 3 – The structure of one UDP package sent by the Digital Receiver

2.2 Qt - The Graphical User Interface Development Tool

When writing Graphical User Interfaces (GUI) a good software development tool can be very usable. The one we used is called Qt and combines C++ and a design environment for setting up the GUI. Qt is a multiplatform C++ application development framework. Multiplatform means that the same code can be compiled for several supported platforms and development framework means that both development tools and class libraries are included. Qt is developed by Trolltech, a Oslo based company with about 75 employees [4].

One feature in Qt is the *signal* and *slot* handling. A signal contains data that can be received by a slot, which basically is a function. One example of how signals and slots can be used to send data from a settings window to the main window is: When the OK button in the settings window is clicked, the *sendData()* signal is send. In the main window a connection between the signal and a slot *receiveData()* has been made and this slot is now activated and processes the sent data. Not only signals can be connected to a slot, Qt also includes the concept of *actions*. Actions are used to connect menu- and toolbar items to a slot. A fundamental part of Qt is the *widgets*. Widget is a contraction of the words *Window* and *Gadget* and is called *controls* in Windows programming terms. Almost everything in a Qt based GUI is a widget e.g. buttons, labels, scrollbars and the main window. In C++ terms a widget is an object of a class that is derived from the class *QWidget* [5].

The Qt Library works as a link between the native platform and the application. This enables the application to follow the standards of the native platform [4].

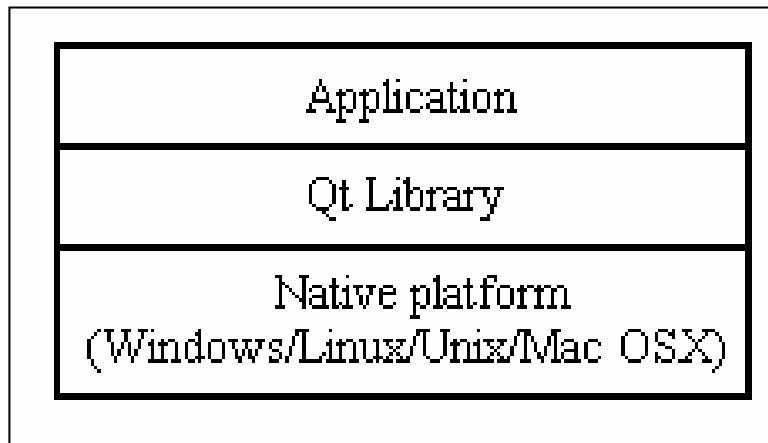


Figure 4 – How Qt provides an encapsulation of the local operating system

The Application is written in platform-independent C++ using Qt library functions. When the application is compiled the compiler builds the executable file adapted to the native platform.

2.3 OpenGL

To draw our different plots we have chosen to use OpenGL. OpenGL[®] is a widely accepted application programming interface (API) for interactive 3D graphics rendering and 2D imaging. It provides device-independent support for common low-level 3D graphics drawing. The big advantage of OpenGL[®] over standard drawing methods is that several modern graphic cards can take over the drawing procedure and therefore ease the load of the CPU [10]. To make it easier to draw different objects we use the GLUT library [11].

3 Algorithms

3.1 FFT – Fast Fourier Transform

The Fast Fourier Transform is an algorithm to determine the frequency components of a signal. To calculate our transform we choose FFTW3, a C subroutine library developed at MIT. This is one of the fastest FFT libraries available. The library includes various functions to compute the Fourier transform e.g. complex data, real data, odd-symmetric data and much more. The transform length can be of arbitrary size, it does not need to be a power of two [6].

The calculation of a FFT with FFTW3 is done in two steps. The first step is to create a “plan” which is an object that contains all the data that FFTW3 needs to calculate the FFT. There are several settings that can be applied in this stage, but we will use the standard setting which are FFTW_FORWARD because we have a negative sign in the exponent of the transform and FFTW_ESTIMATE because we don’t want FFTW3 to waste any time trying to find the best way to calculate the FFT, which can be done using FFTW_MEASURE.

The next step is to calculate the transform and the result is stored in the output array with the zero-frequency (DC) component in the first position [7].

3.2 Direction of arrival for the incoming signal

Using all three components of the magnetic flux density field we can estimate the direction of arrival of an incoming electromagnetic wave. We implemented the algorithm for direction of arrival estimation supplied by Jonas Lundbäck and the result is presented as a unit vector inside a sphere, pointing in the direction of arrival of the incoming wave.

3.3 Polarization

Polarization is an expression of the orientation of the electrical field vector in an electromagnetic field. Polarization can be linear i.e. the direction of the electrical field vector follows a straight line. It can also be elliptical or circular i.e. the direction of the electrical field rotates 360 degrees within each complete wave cycle clockwise or counter clockwise. Polarization is important in wireless communications systems. The physical orientation and shape of an antenna corresponds to the polarization of the radio waves that can be received or transmitted by the antenna. E.g. a vertical orientated dipole antenna receives and transmits vertically polarized waves [8]. To represent these different types of polarization we use the Poincaré Sphere, shown in Figure 6 [9]. The algorithm for polarization was supplied by Jonas Lundbäck and the result is presented as a unit vector inside a sphere, representing a polarization in the Poincaré Sphere.

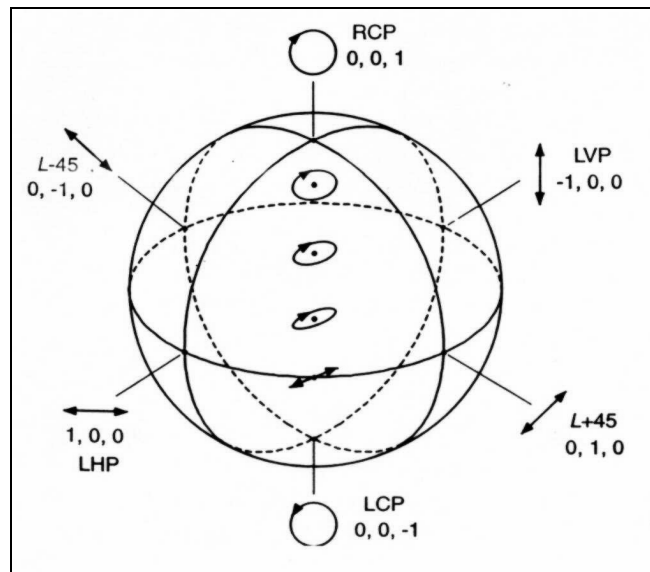


Figure 5 – The Poincaré Sphere

The figure shows the Poincaré Sphere representation of the polarization of the electromagnetic wave. RCP, LCP: Right and Left Circular Polarization; LVP, LHP: Linear Vertical/Horizontal Polarization.

4 Digital Receiver Software

In this section we describe our application, starting with the program's main-window and continuing with the different plot- and settings- windows. First we will give you a short description of the naming standards we have used for variables, functions and files. The source code is located in appendix A.

4.1 Naming of Variables, Functions and Files

We try not to use shortenings, unless the names would be unmanageably long. A file name is printed together in one word with only lower case letters, e.g. *mainform.ui.h*. All forms (windows) have names which end with Form. The forms also have upper case first letter, e.g. *MainForm*. An instance of a form is treated as a variable which is typed as the forms but with a lower case first letter e.g. *incomingDataCounter* and *mainForm*. For function we use the same standard as for variables.

4.2 The Main Window

The main window is where all global control is done. The Digital Receiver can be started, stopped and center-frequency, which equals the local-oscillator frequency and the double sided bandwidth can be chosen. The different plotting windows can be opened along with the main settings window. Several functions are allocated to this window and the most important ones will be presented here and in the section UDP Communication. A screenshot of the window is shown in figure 6.

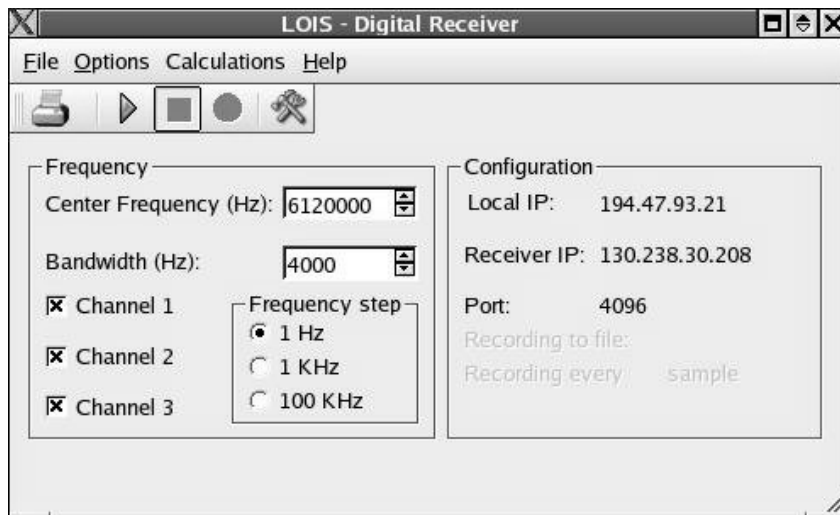


Figure 6 – The Main Program Window

Changing the center-frequency and bandwidth are some of the most important features of this window and are handled by the two slots *centerFrequencySpinBoxChanged()* and *bandwidthSpinBoxChanged()*. Both slots are automatically activated when the values of their corresponding spin box is changed. To realize the changes a DDC control word must be calculated and sent to the mixer.

4.2.1 UDP Communication

For the communication with the UDP protocol, we use the methods included in Qt. We have implemented two functions: `recvData()` and `sendData()`. The first one is automatically activated when the socket receives data. The incoming data is combined to an array of 366 complex elements. Finally the `recvData()` function emits a signal, `emitData()`, which contains the created complex array. The second function, `sendData()` takes a string, containing the command to send, as argument. If it is a two character command (SA, SC or ID) the function simply sends it to the socket. However if it is a MS command the situation becomes more complicated. The DDC must have the control word's 10 hexadecimal values coded as 5 bytes. For example the combination 2E should be coded as 0010 1110. To make this converting we implemented a function, `charToHex()`, which takes two characters (0-9, A-F) as argument and return the corresponding byte. The byte order must be reversed since little endian is used. This source code is located in the section `mainform.ui.h` in appendix A.

4.2.2 Main Settings Window

In the Main Settings Window there are four tabs: Main, Frequency, Mixer Words and Recorder. In the Main tab the user can select which network interface to use, change IP-address to digital receiver and change the UDP port. In the frequency tab center-frequency and bandwidth is changed. The code for automatic detection of network interfaces is copied and slightly modified to suit our application [12]. Under the Mixer Word tab all eight control words of the DDC can be modified. A change of center-frequency and bandwidth in the main window automatically updates the affected control words. A screenshot of the settings window is shown in figure 7.

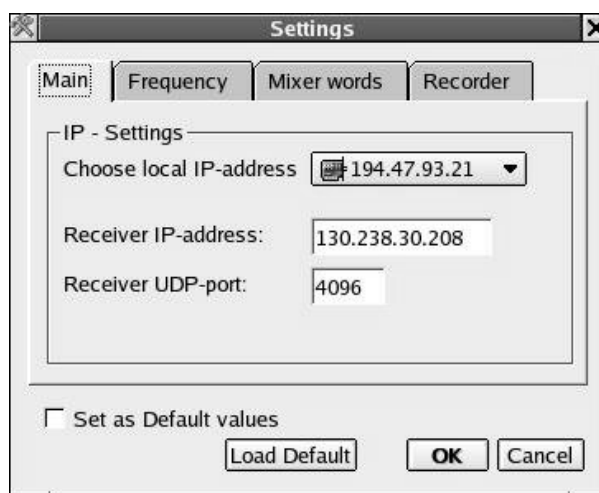


Figure 7 – The Main Settings Window

Under the Recorder tab a Matlab[®] file and a modulo value can be specified. These are used when the save to file button in the main window is activated. A modulo N means that every N:th sample from the digital receiver is being saved to the file. The file constructed as a Mx6 matrix to make it easy to import data to Matlab[®]. This source code is located in the section `mainformsettings.ui.h` in appendix A.

4.3 The Spectrum Window

The spectrum window is built around a standard Qt main window widget with an OpenGL[®] widget (QGLWidget) inside. The spectrum window receives data from our main window using a standard signal and slot technique. When sufficient amount of samples are collected we calculate the FFT, convert the output to dBm and then wait until next update of the spectrum. The time between each update is chosen in the spectrum settings window.

To determine the transform length n we use the following formula:

$$n = \frac{\text{windowWidth} - 60}{\text{step}},$$

where windowWidth is the window width in pixels and step is the step between each draw point. The reason we subtract 60 is because we want a margin of 30 pixels from each side of the x-axis. A screenshot of the window is shown in figure 8 and the source code is located in the sections `fftform.ui.h` and `plotGl.cpp` in appendix A.

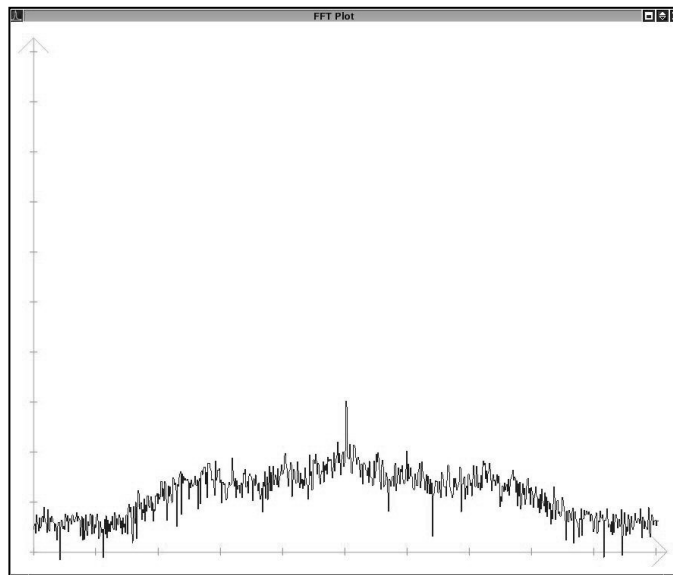


Figure 8 – The Spectrum Window

The figure shows a spectrum with frequency on the x-axis and the amplitude in dBm on the y-axis.

4.3.1 Spectrum Settings Window

To bring up the spectrum settings window you right click on the spectrum window and then click Settings. This source code is located in the section `fftformsettings.ui.h` in appendix A. See table 4 for different settings that can be made.

Settings name	Function
Refresh rate	This controls how often the plot will be update per second.
X- scale draw step	This value affects the step between each draw point along the X-axis. A small value gives a more accurate spectrum.
Y-scale and Y-scale step	Controls the maximum Y- axis value and each step along the axis.
Plot	Here you can choose which channel to plot. The combined plot adds all the channels together and plots the mean value.
Plot type	This setting only applies to visual appearance of the spectrum, you can choose between some different ways to plot the spectrum. The redraw function controls whether the plot should be cleared or not before each update.

Table 4 – The spectrum settings window

4.4 Direction and Polarization Window

In this window we visualize the estimation of the direction of arrival and the polarization of the incoming electromagnetic wave. This window is built in the same way as the spectrum window. It has a main window, a QGLWidget where the drawing is made and a settings window. Both the direction of arrival (yellow vector) and the polarization (blue vector) is represented as a unit vector within the Poincaré Sphere. To make the visualizations of the direction of arrival and polarization more stable we create a mean value of 99 old values and one new. This results in a short delay in change of the two vectors. A screenshot is shown in figure 9 and the source code is located in the sections `direcpolarform.ui.h` and `direcpolarplotgl.cpp` in appendix A.

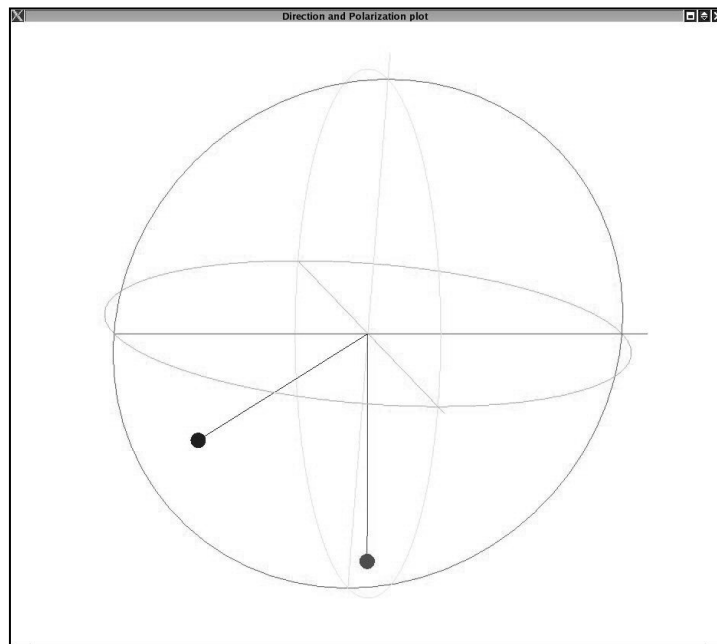


Figure 9 – The Direction and Polarization Window
One vector points out the direction of arrival and the other one the polarization of the incoming electromagnetic wave.

4.4.1 Direction and Polarization Settings Window

In this window you can choose how often the plot should be update (refresh rate) per second. You can also rotate the Poincaré Sphere along each axis. This source code is located in the section `direcpolarformsettings.ui.h` in appendix A.

5 Discussion

In this section we discuss the result of our project with focus on how it ended up compared to the objectives. We will also discuss possible further developments and changes which can be applied to digital receiver and the software.

On the positive side we think that the application is easy to work with. Most of the controls are self explaining and the visualizations are fairly easy to interpret. All key features from the objectives except the AM-demodulation are implemented.

Further updates includes:

- Naming of windows, variables and functions is not consequent.
- Support for AM-demodulation.
- Text labels in the plot windows.
- Optimization of the source code.
- Long term test for memory leaks.
- Handling of both user and system errors.
- Scale figures in both of the plot windows.
- dBm Y- scale in the spectrum window.

5.1 Future work

It is desirable to make the application work on other platforms than Linux Redhat 9.0. To make it work on other platforms a compilation has to be done on each platform. This should be possible since Qt, GLUT and FFTW3 all can be used on different platforms.

The LOIS project is based on an antenna array construction and therefore multi-antenna support is a key feature. This would also make it possible to use the application in educational purposes e.g. laboratory experiments in courses in adaptive filtering.

6 References

- [1] MIT Haystack Observatory (2002), *What is LOFAR?*
<http://web.haystack.mit.edu/lofar/overview.html> (Acc. 2003-05-19)
- [2] IRFU (Swedish Institute of Space Physics, Uppsala), *LOIS – Sveriges bidrag till världens största digitala rymdobservatorium*, http://www.physics.irfu.se/LOIS/LOIS_A4.pdf (Acc. 2003-05-19)
- [3] Intersil (2000), *HSP50016 Data Sheet*, <http://www.intersil.com/data/fn/fn3288.pdf>, (Acc. 2003-05-18)
- [4] Trolltech, *Qt Technical Presentation*,
<http://www.trolltech.com/products/qt/qtpresentation.pdf>, (Acc. 2003-05-20)
- [5] Dalheimer, Matthias Kalle (2002), *Programming with Qt*, Köln, O'Reilly, Second Edition
- [6] FFTW, *FFTW Home Page*, <http://www.fftw.org/> (Acc. 2003-04-15)
- [7] FFTW, *FFTW 3.0*, http://www.fftw.org/fftw3_doc/Introduction.html#Introduction (Acc. 2003-04-15)
- [8] http://whatis.techtarget.com/definition/0,,sid9_gci843762,00.html
- [9] Allen, Christopher T, *Polarization-Mode Dispersion*,
<http://www.itc.ku.edu/~arunc/PMD%20Tutorial.doc>, (Acc. 2003-05-20)
- [10] OpenGL, *Overview of OpenGL*, <http://www.opengl.org/developers/about/overview.html>
(Acc. 2003-05-03)
- [11] OpenGL, *GLUT – OpenGL Utility Toolkit*,
<http://www.opengl.org/developers/documentation/glut.html> (2003-5-10)
- [12] Source Forge (2002), *The Network Clipboard*, <http://netclipboard.sourceforge.net> (Acc. 2003-04-20)

Appendix A

Mainform.ui.h

This is the code to the main window of the application.

*/*Class variables, in the real source code this is NOT placed here.*

This structure holds all the settings data

```
struct data{QString sm[8];           Mixer conf. words 0 - 7
            QString myIpAddress;   Computer IP- address
            QString ipAddress;     Receiver IP- address
            QString udpPort;       Receiver UDP- port
            QString filename;      File name for the recorder
            int centerFrequency;   Center frequency
            int bandwidth;         Bandwidth
            int modulo;};         The modulo value that is used when
                                recording to file
```

```
data mainSettings;           The mainSettings structure, declared as data
bool updateConnection;       Shows if the user has changed the local ip and port
bool recordToFile;          If the user has chosen to record samples
*/
```

```
//Where to place our register keys,
//folder LOIS with the subfolder "DIGITAL_RECEIVER"
const QString WINDOWS_REGISTRY = "/LOIS";
const QString APP_KEY = "/DIGITAL_RECEIVER/";
```

//Saves all the current settings to registry.

```
void MainForm::saveSettings()
{
    QSettings settings;
    settings.insertSearchPath( QSettings::Windows, WINDOWS_REGISTRY );
    //Window position, X and Y
    settings.writeEntry( APP_KEY + "WindowX", x());
    settings.writeEntry( APP_KEY + "WindowY", y());
    //Computer IP- address
    settings.writeEntry( APP_KEY + "myIpAddress",
        mainSettings.myIpAddress );
    //Receiver IP- address
    settings.writeEntry( APP_KEY + "ipAddress", mainSettings.ipAddress );
    //Receiver UDP- port
    settings.writeEntry( APP_KEY + "udpPort", mainSettings.udpPort);
    //Center frequency
    settings.writeEntry( APP_KEY + "centerFrequency",
        mainSettings.centerFrequency);
    //Bandwidth
    settings.writeEntry( APP_KEY + "bandwidth", mainSettings.bandwidth);
```

Appendix A – mainform.ui.h

```
//Mixer configuration words 0 - 7
settings.writeEntry( APP_KEY + "sm0", mainSettings.sm[0]);
settings.writeEntry( APP_KEY + "sm1", mainSettings.sm[1]);
settings.writeEntry( APP_KEY + "sm2", mainSettings.sm[2]);
settings.writeEntry( APP_KEY + "sm3", mainSettings.sm[3]);
settings.writeEntry( APP_KEY + "sm4", mainSettings.sm[4]);
settings.writeEntry( APP_KEY + "sm5", mainSettings.sm[5]);
settings.writeEntry( APP_KEY + "sm6", mainSettings.sm[6]);
settings.writeEntry( APP_KEY + "sm7", mainSettings.sm[7]);
}

//Loads settings from the registry and stores the value in the mainSettings
//structure. Example, readEntry( APP_KEY + "sm7", "F000000002" )
//reads the key "sm7" from registry, if not available return "F000000002"

void MainForm::loadSettings()
{
    QSettings settings;
    settings.insertSearchPath( QSettings::Windows, WINDOWS_REGISTRY );
    int windowX = settings.readNumEntry( APP_KEY + "WindowX", 0 );
    int windowY = settings.readNumEntry( APP_KEY + "WindowY", 0 );
    mainSettings.myIpAddress = settings.readEntry( APP_KEY + "myIpAddress",
        "127.0.0.1" );
    mainSettings.ipAddress = settings.readEntry( APP_KEY + "ipAddress",
        "194.47.95.95" );
    mainSettings.udpPort = settings.readEntry( APP_KEY + "udpPort", "4096" );
    mainSettings.centerFrequency = settings.readNumEntry( APP_KEY +
        "centerFrequency", 6550000 );
    mainSettings.bandwidth = settings.readNumEntry( APP_KEY + "bandwidth",
        10000 );
    mainSettings.sm[0] = settings.readEntry( APP_KEY + "sm0", "0001000000" );
    mainSettings.sm[1] = settings.readEntry( APP_KEY + "sm1", "3200000001" );
    mainSettings.sm[2] = settings.readEntry( APP_KEY + "sm2", "5000000000" );
    mainSettings.sm[3] = settings.readEntry( APP_KEY + "sm3", "7000000000" );
    mainSettings.sm[4] = settings.readEntry( APP_KEY + "sm4", "9100000051" );
    mainSettings.sm[5] = settings.readEntry( APP_KEY + "sm5", "B01FF00001" );
    mainSettings.sm[6] = settings.readEntry( APP_KEY + "sm6", "D009526001" );
    mainSettings.sm[7] = settings.readEntry( APP_KEY + "sm7", "F000000002" );
    move( windowX, windowY ); //Move window to last know position
}

void MainForm::filePrint()
{

}

//If user exits the program, send stop to receiver, save current settings,
//delete the UDP -connection and then exit
void MainForm::fileExit()
{
```

Appendix A – mainform.ui.h

```
    sendData("SC");
    saveSettings();
    delete qsd;
    delete qsn;
    QApplication::exit(0);
}

//The About message
void MainForm::helpAbout()
{
    QMessageBox::information(this, "About LOIS Digital Reciver",
        "LOIS Digital Reciver\n""Made by:\n""Petter Andersson\n"
        "Patrik Berglund\n""Jonas Lundbäck\n");
}

//If user changes the bandwidth spinbox
void MainForm::bandwidthSpinBoxChanged()
{
    char mixerWord[5], tmp[2];
    unsigned int HDFdcp, HDFg, R;
    QString str;
    //Get current bandwidth value and store it in our settings structure
    mainSettings.bandwidth = bandwidthSpinBox->value();

    //These calculations where copied from Walter Puccio's radio5 program.
    R = (unsigned int)((0.1375*25000000)/(mainSettings.bandwidth));

    //HDF gain compensation number
    mixerWord[0] = 0x01|(((char)(75-ceil(5.0*3.32*log10(R))))<<1);
    mixerWord[1] = 0x00;
    mixerWord[2] = 0x00;
    mixerWord[3] = 0x00;
    mixerWord[4] = 0x91;

    //Convert hex number to character.
    //Example, the value F is converted to the character F.
    for(int i = 4; i >= 0; i--){
        hexToChar(mixerWord[i], tmp);
        str[8 - 2*i] = tmp[0];
        str[8 - 2*i+1] = tmp[1];
    }
    mainSettings.sm[4] = str;

    //These calculations where copied from Walter Puccio's radio5 program
    HDFdcp = R-1; //HDF Decimation Counter Preload
    HDFg=(unsigned int)(32768.0*pow(2.0,ceil(16.60964*log10((double)R))
        -16.60964*log10((double)R))); //Scale factor

    mixerWord[0] = 0x01|(HDFg<<5);
    mixerWord[1] = HDFg>>3;
```

Appendix A – mainform.ui.h

```
mixerWord[2] = (HDFg>>11)|(HDFdcp<<5);
mixerWord[3] = HDFdcp>>3;
mixerWord[4] = 0xB0| (HDFdcp>>11);

for(int i = 4; i >= 0; i--){
    hexToChar(mixerWord[i], tmp);
    str[8 - 2*i] = tmp[0];
    str[8 - 2*i+1] = tmp[1];
}
mainSettings.sm[5] = str;

//If start button is pressed, send the new mixer conf. words
if(optionsStartAction->isOn()) {
    sendData(QString("MS% 1").arg(mainSettings.sm[4]));
    sendData(QString("MS% 1").arg(mainSettings.sm[5]));
}
}

//If user changes the center frequency spinbox
void MainForm::centerFrequencySpinBoxChanged()
{
    char mixerWord[5], tmp[2];
    long dFreq;
    QString str;
    double compensation = 1.0 - (23.0/6120000.0);

    //Get current value from the center frequency spinbox
    mainSettings.centerFrequency = centerFrequencySpinBox->value();

    //This calculation where copied from Walter Puccio's radio5 program
    dFreq = (long)((((double)mainSettings.centerFrequency/1000)
        * compensation * 1342.17728); //Phase Increment

    mixerWord[0] = 0x01;
    mixerWord[1] = dFreq<<4;
    mixerWord[2] = dFreq>>4;
    mixerWord[3] = dFreq>>12;
    mixerWord[4] = ((dFreq>>20)&0xF)|0x30;

    for(int i = 4; i >= 0; i--){
        hexToChar(mixerWord[i], tmp);
        str[8 - 2*i] = tmp[0];
        str[8 - 2*i+1] = tmp[1];
    }
    mainSettings.sm[1] = str;

    //If start button is pressed, send the new mixer conf. words
    if(optionsStartAction->isOn())
        sendData(QString("MS% 1").arg(mainSettings.sm[1]));
}
```

Appendix A – mainform.ui.h

```
void MainForm::ch1CheckBoxChanged()
{
}
void MainForm::ch2CheckBoxChanged()
{
}
void MainForm::ch3CheckBoxChanged()
{
}

//If user presses the stop button
void MainForm::optionsStop()
{
    //Stop the digital receiver
    sendData("SC");

    //Change the buttons in the menu and toolbar,
    //make it possible to click start and settings but not stop.
    optionsStopAction->setEnabled(FALSE);
    optionsStartAction->setEnabled(TRUE);
    optionsSettingsAction->setEnabled(TRUE);

    //If the user have chosen to record samples to file,
    //stop recording and change the buttons
    if(optionsRecordAction->isOn()){
        optionsRecordAction->setOn(FALSE);
        optionsRecord(FALSE);
    }
    optionsRecordAction->setEnabled(FALSE);
}

//If user presses the start button
void MainForm::optionsStart()
{
    //If the user have chosen a file to save samples to,
    //make it possible to press the record button
    if(mainSettings.filename != "")
        optionsRecordAction->setEnabled(TRUE);

    optionsStopAction->setEnabled(TRUE);
    optionsStartAction->setEnabled(FALSE);
    optionsSettingsAction->setEnabled(FALSE);

    //Send the identity word to the digital receiver, ID
    sendData("ID");
    //Send the mixer conf. words
    for (int i=0; i<8; i++)
        sendData(QString("MS%1").arg(mainSettings.sm[i]));
    //Send the start word, SA
```

Appendix A – mainform.ui.h

```
    sendData("SA");
}

//If the record button is toggled
void MainForm::optionsRecord(bool enable)
{
    //If enabled, set text in main window and change recordToFile to TRUE.
    if(enable){
        recordToFile = TRUE;
        moduloTextLabel->setEnabled(TRUE);
        modulo3TextLabel->setEnabled(TRUE);
        modulo2TextLabel->setText(QString("%1").arg(mainSettings.modulo));
        filenameTextLabel->setEnabled(TRUE);
        filename2TextLabel->setText(mainSettings.filename);
    }
    else{
        recordToFile = FALSE;
        moduloTextLabel->setEnabled(FALSE);
        modulo3TextLabel->setEnabled(FALSE);
        modulo2TextLabel->setText("");
        filenameTextLabel->setEnabled(FALSE);
        filename2TextLabel->setText("");
    }
}

//If the Settings button is pressed
void MainForm::optionsSettings()
{
    settings = new SettingsForm(this); //Allocate memory for the settingsform
    //Set values in the settingsform
    settings->setValues(mainSettings.sm,
                       mainSettings.ipAddress,
                       mainSettings.udpPort,
                       mainSettings.centerFrequency,
                       mainSettings.bandwidth);
    //Connect the signal "emitsettings" from the settingsform to the slot
    // "updateValues" in mainform.
    //Sends all the settings made in the settings form
    connect(settings, SIGNAL(emitSettings(QString*, QString, QString, QString,
                                       QString, int, int, int)), this, SLOT(updateValues(QString*, QString,
                                       QString, QString, QString, int, int, int)));
    //Bring up the settingsform
    settings->exec();
}

//When the program is started
void MainForm::init()
{
    //Load our settings from registry
    loadSettings();
}
```

Appendix A – mainform.ui.h

```
//Update all the fields with values from the registry
 //(center frequency, mixer conf. words, etc.)
updateFields();

//Make it impossible to click record and stop.
optionsRecordAction->setEnabled(FALSE);
optionsStopAction->setEnabled(FALSE);

mainSettings.filename = "";
recordToFile = FALSE;

//Setup the UDP- connection
qsd = new QSocketDevice(QSocketDevice::Datagram);
//Set receive data buffer
qsd->setReceiveBufferSize(49152);
//Tell us when a UDP- packet arrives,
//runs the receiveData function everytime a package arrives
qsn = new QSocketNotifier(qsd->socket(),QSocketNotifier::Read );
connect( qsn, SIGNAL( activated(int) ),this, SLOT( receiveData(int)));
}

//When user exits the program by not clicking File -> Exit
void MainForm::closeEvent(QCloseEvent *)
{
    fileExit();
}

//Set receiver IP -address
void MainForm::setupConnection()
{
    myIp.setAddress(mainSettings.myIpAddress);
    //Listen to data from IP (myIP) on port (mainSettings.udpPort)
    qsd->bind(myIp, mainSettings.udpPort.toUInt(0,10));
    updateConnection = FALSE;
}

//Send the Qstring data to the digital receiver
int MainForm::sendData(QString data)
{
    int strlength = 0;
    char buffer[20];
    QHostAddress *ip = new QHostAddress;

    //Set the destination IP -address
    ip->setAddress(mainSettings.ipAddress);

    //If changes have been made to the receiver/computer IP -address
    if(updateConnection)
        setupConnection();
    //If the data string is a mixer conf. word, convert char to hex.
    //Example, the character F is converted to F hex

```

Appendix A – mainform.ui.h

```

if (data[0] == 'M' && data[1] == 'S'){
    for(int j = 11, k=2; j>=2; j-=2, k++)
        buffer[k] = charToHex(data.ref(j), data.ref(j-1) );
    buffer[0] = 'M';
    buffer[1] = 'S';
    strlength=7;
}
else {
    strncpy(buffer, data.ascii(), 20);
    strlength = strlen(buffer);
}
//Send the data string
return qsd->writeBlock(buffer, strlength, ip->ip4Addr(),
    mainSettings.udpPort.toUInt(0, 10));
delete ip;
}

//This funtion is automatically run every time a udp -package arrives
void MainForm::receiveData(int socket)
{
    char buffer[1466];
    int nr;
    complex<double> complexArray[366];

    if(qsd->bytesAvailable() >= 1466){ //If 1466 (One package) is available
        nr = qsd->readBlock(buffer, 1466); //If 1466 could be read from buffer
        if(nr >= 1466){
            //If the user want's to save samples to a file (record).
            if(recordToFile){
                //Open file in append mode
                ofstream file(mainSettings.filename, ios::app);
                if (file.is_open()){
                    for(int j = 0, k = 2; j < 122; j++){
                        if(!(j%mainSettings.modulo)){
                            for(int i = 0; i < 6;i++){
                                file << (signed short)((((unsigned char)
                                    buffer[i*2+k]) | ((unsigned char)
                                    buffer[i*2+k+1]) << 8) << " ";
                            }
                            file << endl;
                        }
                        k+=12;
                    }
                }
                file.close();
            }
            //each packet includes 122 x 3 samples (x,y,z)
            for(int j = 0, k = 2, l = 0; j < 122; j++){
                //Extract x, y, z channels and store them in complexArray,
                //each complex value represents 2 byte

```

Appendix A – mainform.ui.h

```
        for(int i = 0; i < 3;i++, l++)
            complexArray[l] = complex<double>(
                (double)(signed short)(((unsigned char) buffer[i*2+k])
                    | ((unsigned char) buffer[i*2+k+1]) << 8),(double)
                (signed short)(((unsigned char) buffer[i*2+k+2])
                    | ((unsigned char) buffer[i*2+k+3]) << 8));
            k+=12;
        }
        emit emitData(complexArray); //Send the array as a signal
    }
}
}
```

//Called when you click Ok in the settings form.

//Saves the changes made to the mainSettings structure

```
void MainForm::updateValues(QString *sm, QString myIp, QString ip,
    QString udp, QString filename, int cf, int bw, int mod)
{
    for (int i=0; i<8; i++)
        mainSettings.sm[i] = sm[i];
    mainSettings.myIpAddress = myIp;
    mainSettings.ipAddress = ip;
    mainSettings.udpPort = udp;
    mainSettings.centerFrequency = cf;
    mainSettings.bandwidth = bw;
    mainSettings.filename = filename;
    mainSettings.modulo = mod;
```

//Update the textlabel and spinboxes in the main window

```
updateFields();
}
```

//Updates the textlabel and spinboxes in the main window

```
void MainForm::updateFields()
{
    centerFrequencySpinBox->setValue(mainSettings.centerFrequency);
    bandwidthSpinBox->setValue(mainSettings.bandwidth);
    ipShowTextLabel->setText(mainSettings.ipAddress);
    portShowTextLabel->setText(mainSettings.udpPort);
    localIpShowTextLabel->setText(mainSettings.myIpAddress);

    updateConnection = TRUE;
}
```

//If the FFT button is clicked

```
void MainForm::calcFft()
{
    //Allocates memory for the form, sets the parent and shows the form
    fftPlot = new FftForm(this, 0,Qt::WDestructiveClose | Qt::WType_TopLevel);
    fftPlot->setParent(this);
}
```

Appendix A – mainform.ui.h

```
fftPlot->show();
}

//Convert two characters to a hexadecimal value
char MainForm::charToHex( char c2, char c1 )
{
    if (c1>='0' && c1<='9') c1-=48;
    else c1-=55;
    if (c2>='0' && c2<='9') c2-=48;
    else c2-=55;
    return (c1<<4) | c2;
}

//Convert a hexadecimal value to a character, the result is written to c1
void MainForm::hexToChar(char c, char *c1)
{
    c1[0] = (c & 0xF0) >>4;
    c1[1] = c & 0x0F;

    if(c1[0] < 10) c1[0]+= 48;
    else c1[0]+=55;
    if(c1[1] < 10) c1[1]+= 48;
    else c1[1]+=55;
}

//Controls the step length of the spinboxes in the main window
void MainForm::setStepLength()
{
    if(freqRadioButton1->isOn()){
        centerFrequencySpinBox->setLineStep(1);
        bandwidthSpinBox->setLineStep(1);
    }
    else if(freqRadioButton2->isOn()){
        centerFrequencySpinBox->setLineStep(1000);
        bandwidthSpinBox->setLineStep(1000);
    }
    else{
        centerFrequencySpinBox->setLineStep(100000);
        bandwidthSpinBox->setLineStep(100000);
    }
}

//If the Direction and polarization button is pressed
void MainForm::calcDirecPolar()
{
    //Allocates memory for the form, sets the parent and shows the form
    direcPolarPlot = new DirecPolarForm(this, 0, Qt::WDestructoriveClose
    | Qt::WType_TopLevel);
    direcPolarPlot->setParent(this);
}
```

Appendix A – mainform.ui.h

```
    direcPolarPlot->show();  
}
```

settingsform.ui.h

This is the code for the main settings window.

```

//Where to place our register keys,
//folder LOIS with the subfolder "DIGITAL_RECEIVER_DEFAULT"
const QString WINDOWS_REGISTRY = "/LOIS";
const QString APP_KEY = "/DIGITAL_RECEIVER_DEFAULT/";

//Automatically run when the window is opened
void SettingsForm::init()
{
    //Populate the network interface combobox.
    interfaceList.populateComboBox(cb);
}

//When the button "save default values" is pressed, saves all values to
//registry.
//Example settings.writeEntry(APP_KEY + "sm0", sm0LineEdit->text());
//Write the value of sm0LineEdit (Mixer word 0) to the key sm0
void SettingsForm::saveDefaultSettings()
{
    //Save values to registry
    QSettings settings;
    settings.insertSearchPath(QSettings::Windows, WINDOWS_REGISTRY);
    settings.writeEntry(APP_KEY + "sm0", sm0LineEdit->text());
    settings.writeEntry(APP_KEY + "sm1", sm1LineEdit->text());
    settings.writeEntry(APP_KEY + "sm2", sm2LineEdit->text());
    settings.writeEntry(APP_KEY + "sm3", sm3LineEdit->text());
    settings.writeEntry(APP_KEY + "sm4", sm4LineEdit->text());
    settings.writeEntry(APP_KEY + "sm5", sm5LineEdit->text());
    settings.writeEntry(APP_KEY + "sm6", sm6LineEdit->text());
    settings.writeEntry(APP_KEY + "sm7", sm7LineEdit->text());
    settings.writeEntry(APP_KEY + "ipAddress", ipLineEdit->text());
    settings.writeEntry(APP_KEY + "myIpAddress", cb->currentItem());
    settings.writeEntry(APP_KEY + "udpPort", udpPortLineEdit->text());
    settings.writeEntry(APP_KEY + "centerFrequency",
        centerFrequencyLineEdit->text());
    settings.writeEntry(APP_KEY + "bandwidth", bandwidthLineEdit->text());
    settings.writeEntry(APP_KEY + "filename", recordLineEdit->text());
    settings.writeEntry(APP_KEY + "recorderModulo", recordSpinBox->value());
}

//When the Ok button is pressed
void SettingsForm::okPushButtonClicked()
{
    //If save as default values is checked, save current values as default
    if(defaultCheckBox->isChecked())
        saveDefaultSettings();
}

```

```

QString sm[8] = {
    sm0LineEdit->text(),
    sm1LineEdit->text(),
    sm2LineEdit->text(),
    sm3LineEdit->text(),
    sm4LineEdit->text(),
    sm5LineEdit->text(),
    sm6LineEdit->text(),
    sm7LineEdit->text()};
QString ip = ipLineEdit->text();
QString myIp = cb->currentText();
QString udp = udpPortLineEdit->text();
QString filename = recordLineEdit->text();
int cf = centerFrequencyLineEdit->text().toInt(0, 10);
int bw = bandwidthLineEdit->text().toInt(0, 10);
int mod = recordSpinBox->value();
//Emit the settings, signal captured by main window
emit emitSettings(sm, myIp, ip, udp, filename, cf, bw, mod);
accept();
}

//When Load default button is pressed, load all values from registry
void SettingsForm::loadDefaultPushButtonClicked()
{
    //Get values from the registry
    QSettings settings;
    settings.insertSearchPath( QSettings::Windows, WINDOWS_REGISTRY );
    sm0LineEdit->setText(settings.readEntry(APP_KEY + "sm0", "0001000000"));
    sm1LineEdit->setText(settings.readEntry(APP_KEY + "sm1", "3200000001"));
    sm2LineEdit->setText(settings.readEntry(APP_KEY + "sm2", "5000000000"));
    sm3LineEdit->setText(settings.readEntry(APP_KEY + "sm3", "7000000000"));
    sm4LineEdit->setText(settings.readEntry(APP_KEY + "sm4", "9100000051"));
    sm5LineEdit->setText(settings.readEntry(APP_KEY + "sm5", "B01FF00001"));
    sm6LineEdit->setText(settings.readEntry(APP_KEY + "sm6", "D009526001"));
    sm7LineEdit->setText(settings.readEntry(APP_KEY + "sm7", "F00000002"));
    cb->setCurrentItem(settings.readNumEntry(APP_KEY + "myIpAddress", 0));
    recordSpinBox->setValue(settings.readNumEntry(APP_KEY + "modulo", 1));

    ipLineEdit->setText(
        settings.readEntry(APP_KEY + "ipAddress", "194.47.95.95"));

    recordLineEdit->setText(settings.readEntry(APP_KEY + "filename", ""));
    udpPortLineEdit->setText(settings.readEntry(APP_KEY + "udpPort", "4096"));

    centerFrequencyLineEdit->setText(
        settings.readEntry(APP_KEY + "centerFrequency", "6550"));

    bandwidthLineEdit->setText(
        settings.readEntry(APP_KEY + "bandwidth", "1000"));
}

```

Appendix A – settingsform.ui.h

//Called from the main window, sets all the values in the settings window

```
void SettingsForm::setValues(QString *sm, QString ip,
    QString udp, int cf, int bw)
{
    sm0LineEdit->setText(sm[0]);
    sm1LineEdit->setText(sm[1]);
    sm2LineEdit->setText(sm[2]);
    sm3LineEdit->setText(sm[3]);
    sm4LineEdit->setText(sm[4]);
    sm5LineEdit->setText(sm[5]);
    sm6LineEdit->setText(sm[6]);
    sm7LineEdit->setText(sm[7]);
    ipLineEdit->setText(ip);
    udpPortLineEdit->setText(udp);
    centerFrequencyLineEdit->setText(QString( "%1" ).arg(cf, 0, 10 ));
    bandwidthLineEdit->setText(QString( "%1" ).arg(bw, 0, 10 ));
}
```

//When the Choose file button is pressed

```
void SettingsForm::fileSaveAs()
{
    //Bring up the file dialog
    QString filename = QFileDialog::getSaveFileName(
        QString::null, "Mx3 matrix (*.mx3)", this,
        "file save as", "LOIS digital receiver -- File Save As" );
    if ( ! filename.isEmpty() ) {
        int ans = 0;
        //The overwrite file warning
        if ( QFile::exists( filename ) )
            ans = QMessageBox::warning(
                this, "LOIS digital receiver -- Overwrite File",
                QString( "Overwrite\n'%1'?" ).
                    arg( filename ),
                "&Yes", "&No", QString::null, 1, 1 );
        //If Ok, set the filename
        if ( ans == 0 ) {
            recordLineEdit->setText(filename);
            return;
        }
    }
}
```

fftform.ui.h

This is the base form for the spectrum plot.

```

/*Class variables, in the real source code this is NOT placed here. .
  int complexDataLength;           The length (size n) of the FFT.
                                   This value changes depending on
                                   window size
  int complexDataCounter;         How many samples we have collected
                                   from each channel
  fftw_plan p;                    The FFTW3 plan, used to calculate
                                   the FFT
  plotGLWidget *fftGLPlot;        The Open GL drawing area
  QTimer *timer;                  The timer "interrupt
  complex<double> complexData[3][2048];The incoming data array
  double argAbsArray[3][2048][2]; The array where the spectrum is
                                   stored
  FftFormOptions *fftSettings;    The Options window
  data plotSettings;              The data structure, holds different
                                   settings
  bool fftDone;                   Shows if the FFT is calculated or
                                   not

  struct data{int redrawWait;      Time between each update (ms)
               int step;          Step between each draw point
               int plotType;      Type of plot
               int yScale;        Y- scale maximum
               int yScaleStep;    Y- scale step
               bool redraw;       Redraw the window between each
                                   update
               bool plotCh1;      Plot ch1 spectrum ?
               bool plotCh2;
               bool plotCh3;
               bool plotCh123Combined;}; Plot the combined spectrum ?
*/

//Where to place our register keys,
//folder LOIS with the subfolder "DIGITAL_RECEIVER_FFT"
const QString WINDOWS_REGISTRY = "/LOIS";
const QString APP_KEY = "/DIGITAL_RECEIVER_FFT/";

//Automatically run when the window is opened
void FftForm::init()
{
    //Load default settings from registry
    loadSettings();

    //Allocate memory for the fft plot (the drawing area)
    fftGLPlot = new plotGLWidget(this, "Plot GI", 0, Qt::WDestroyiveClose);
    //Set the size of the drawing area to the same as window size

```

Appendix A – fftform.ui.h

```
fftGIPlot ->resize(size().width(), size().height());
//Set different settings to the plot function
fftGIPlot->setValues(plotSettings.step,
                    plotSettings.plotType,
                    plotSettings.yScale,
                    plotSettings.yScaleStep,
                    plotSettings.redraw,
                    plotSettings.plotCh1,
                    plotSettings.plotCh2,
                    plotSettings.plotCh3,
                    plotSettings.plotCh123Combined);
    //Show the drawing area
fftGIPlot ->show();

complexDataCounter = 0;

complexDataLength = 128;

//We haven't calculated the FFT yet
fftDone = FALSE;

//The timer gives a "interrupt" that tells us when it's time
//to update the plot
timer = new QTimer(this);
//When timer reaches 0 the updatePlot() function will be called
connect(timer, SIGNAL(timeout()), this, SLOT(updatePlot()));
//Start the timer
timer->start(plotSettings.redrawWait, TRUE );
}

//This function is automatically called when the window size changed
void FftForm :: resizeEvent ( QResizeEvent *)
{
    //Resize the drawing area to fit the window
    fftGIPlot -> resize(size().width(), size().height());
    //Calculate the length (size n) of the FFT
    complexDataLength = (int)((size().width()-60)/plotSettings.step);
}

// Connect the receiveData() slot to the emitData() signal from main window
void FftForm::setParent(QWidget *parent)
{
    //When a package arrives, call receiveData()
    connect(parent, SIGNAL(emitData(complex<double>*)),
            this, SLOT(receiveData(complex<double>*)));
}

void FftForm::receiveData(complex<double> *data)
{
```

Appendix A – fftform.ui.h

```

    //Always 366 values, 122 from each channel
    for(int i = 0; i < 366; i++){
        //If sufficient amount of samples are collected
        if(complexDataCounter >= complexDataLength){
            //If the FFT is not calculated
            if(!fftDone){
                //Calculate the FFT
                calcFft();
                fftDone = TRUE;
            }
            break; //No need to receiver more samples, the FFT is done,
                //now we wait for the timer "interrupt" to accure
        }
        if ((i % 3) == 0) //Store channel 1 sample in complexdata[0]
            complexData[0][complexDataCounter] = data[i];
        else if ((i % 3) == 1) //Store channel 2 sample in complexdata[1]
            complexData[1][complexDataCounter] = data[i];
        else{ //Store channel 3 sample in complexdata[2]
            complexData[2][complexDataCounter] = data[i];
            complexDataCounter++;
        }
    }
}

//Called by the timer
void FftForm::updatePlot()
{
    //if all the samples that are needed have been collected, update the plot
    if(complexDataCounter >= complexDataLength){
        fftGPlot->updatePlot(
            argAbsArray[0], argAbsArray[1], argAbsArray[2], complexDataLength);

        //Allow the receivedata() to fill our array with new data
        complexDataCounter = 0;
        //The FFT have not been calculated
        fftDone = FALSE;
        //Redraw the timer
        timer->start(plotSettings.redrawWait, TRUE );
    }
    else //Redraw the timer and acquire more samples
        timer->start(plotSettings.redrawWait/2, TRUE );
}

//Called from "receiveData()", calculates the FFT
void FftForm::calcFft()
{
    //If channel 1 is activated in the settings form.
    if(plotSettings.plotCh1){
        //Construct a FFTW3 Plan
        p = fftw_plan_dft_1d(complexDataLength,

```

```

        reinterpret_cast<fftw_complex*>(complexData[0]),
        reinterpret_cast<fftw_complex*>(complexData[0]),
        FFTW_FORWARD, FFTW_ESTIMATE);
    //Calculate the FFT
    fftw_execute(p);
    //Deallocate the plan, no longer needed
    fftw_destroy_plan(p);
}
if(plotSettings.plotCh2){
    p = fftw_plan_dft_1d(complexDataLength,
        reinterpret_cast<fftw_complex*>(complexData[1]),
        reinterpret_cast<fftw_complex*>(complexData[1]),
        FFTW_FORWARD, FFTW_ESTIMATE);
    fftw_execute(p);
    fftw_destroy_plan(p);
}
if(plotSettings.plotCh3){
    p = fftw_plan_dft_1d(complexDataLength,
        reinterpret_cast<fftw_complex*>(complexData[2]),
        reinterpret_cast<fftw_complex*>(complexData[2]),
        FFTW_FORWARD, FFTW_ESTIMATE);
    fftw_execute(p);
    fftw_destroy_plan(p);
}
//If the combined plot is chosen
if(plotSettings.plotCh123Combined){
    //Calculate the mean value for the three channels
    for(int i = 0; i < complexDataLength; i++){
        complexData[0][i] += complexData[1][i] + complexData[2][i];
        complexData[0][i]/=3;
    }
    //calculate the FFT
    p = fftw_plan_dft_1d(complexDataLength,
        reinterpret_cast<fftw_complex*>(complexData[0]),
        reinterpret_cast<fftw_complex*>(complexData[0]),
        FFTW_FORWARD, FFTW_ESTIMATE);
    fftw_execute(p);
    fftw_destroy_plan(p);
}

for(int i = 0; i < complexDataLength; i++){
    //Convert each value to dBm
    if(plotSettings.plotCh1 || plotSettings.plotCh123Combined)
        argAbsArray[0][i][1] =
            10*log10(1000*pow(abs(complexData[0][i]), 2)/complexDataLength);
    if(plotSettings.plotCh2)
        argAbsArray[1][i][1] =
            10*log10(1000*pow(abs(complexData[1][i]), 2)/complexDataLength);
    if(plotSettings.plotCh3)
        argAbsArray[2][i][1] =

```

Appendix A – fftform.ui.h

```
        10*log10(1000*pow(abs(complexData[2][i]), 2)/complexDataLength);
    }
}

//The Options menu
void FftForm::contextMenuEvent( QContextMenuEvent * )
{
    QPopupMenu* contextMenu = new QPopupMenu( this );
    Q_CHECK_PTR( contextMenu );
    //When Options is clicked, run plotSettingsWindow()
    contextMenu->insertItem("Options", this, SLOT(plotSettingsWindow()));
    contextMenu->exec(QCursor::pos());
    delete contextMenu;
}

void FftForm::plotSettingsWindow(){
    //Allocate memory for the options window
    fftSettings = new FftFormOptions(this);

    //Connect the "emitSettings" signal from the
    //settings window with the Slot updateValues in this window
    connect(fftSettings, SIGNAL(
        emitSettings(int, int, int, int, int, bool, bool, bool, bool, bool)),
        this, SLOT(updateValues(
            int, int, int, int, int, bool, bool, bool, bool, bool)));

    //Set values to different fields in the settings window
    fftSettings->setValues(
        plotSettings.step, plotSettings.redrawWait, plotSettings.yScale,
        plotSettings.yScaleStep, plotSettings.plotType, plotSettings.redraw,
        plotSettings.plotCh1, plotSettings.plotCh2, plotSettings.plotCh3,
        plotSettings.plotCh123Combined);

    //Bring up the settings window
    fftSettings->exec();
}

//This function is called when the settings window is closed
void FftForm::updateValues(int refreshR, int xStep, int yScale,
    int yScaleStep, int plotType, bool redraw, bool plotCh1, bool plotCh2,
    bool plotCh3, bool plotCh123Combined)
{
    plotSettings.step = xStep;
    plotSettings.redrawWait =
        (int)(1000/(double)refreshR);
    plotSettings.yScale = yScale;
    plotSettings.yScaleStep = yScaleStep;
    plotSettings.plotType = plotType;
    plotSettings.redraw = redraw;
}
```

Appendix A – fftform.ui.h

```
plotSettings.plotCh1 = plotCh1;
plotSettings.plotCh2 = plotCh2;
plotSettings.plotCh3 = plotCh3;
plotSettings.plotCh123Combined = plotCh123Combined;
complexDataLength = (int)((size().width()-60)/
    plotSettings.step); //calculate a new FFT length (size n)

    //Set the new values to the drawing function
fftGPlot->setValues(xStep, plotType, yScale, yScaleStep, redraw,
    plotCh1, plotCh2, plotCh3, plotCh123Combined);
}

//Loads settings from the registry and stores it in our settings structure
//"plotSettings".
void FftForm::loadSettings()
{
    QSettings settings;
    settings.insertSearchPath( QSettings::Windows, WINDOWS_REGISTRY);
    int windowX = settings.readNumEntry(APP_KEY + "WindowX", 0);
    int windowY = settings.readNumEntry(APP_KEY + "WindowY", 0);
    plotSettings.step = settings.readNumEntry(APP_KEY + "step", 5);
    plotSettings.redrawWait = settings.readNumEntry(APP_KEY+"redrawWait", 50);
    plotSettings.yScale = settings.readNumEntry(APP_KEY + "yScale", 1000);
    plotSettings.yScaleStep =settings.readNumEntry(APP_KEY+"yScaleStep", 100);
    plotSettings.plotType = settings.readNumEntry(APP_KEY + "plotType", 1);
    plotSettings.redraw = (bool)settings.readNumEntry(APP_KEY + "redraw", 1);
    plotSettings.plotCh1 = (bool)settings.readNumEntry(APP_KEY + "plotCh1",0);
    plotSettings.plotCh2 = (bool)settings.readNumEntry(APP_KEY + "plotCh2",0);
    plotSettings.plotCh3 = (bool)settings.readNumEntry(APP_KEY + "plotCh3",0);
    plotSettings.plotCh123Combined =
        (bool)settings.readNumEntry(APP_KEY + "plotCh123Combined", 1);
    move(windowX, windowY);
}

//Saves settings to the registry, not yet implemented
void FftForm::saveSettings()
{
    QSettings settings;
    settings.insertSearchPath( QSettings::Windows, WINDOWS_REGISTRY );
    settings.writeEntry(APP_KEY + "WindowX", x());
    settings.writeEntry(APP_KEY + "WindowY", y());
    settings.writeEntry(APP_KEY + "step", plotSettings.step);
    settings.writeEntry(APP_KEY + "redrawWait", plotSettings.redrawWait);
    settings.writeEntry(APP_KEY + "yScale", plotSettings.yScale);
    settings.writeEntry(APP_KEY + "yScaleStep", plotSettings.yScaleStep);
    settings.writeEntry(APP_KEY + "plotType", plotSettings.plotType);
    settings.writeEntry(APP_KEY + "plotCh1", plotSettings.plotCh1);
    settings.writeEntry(APP_KEY + "plotCh2", plotSettings.plotCh2);
    settings.writeEntry(APP_KEY + "plotCh3", plotSettings.plotCh3);
    settings.writeEntry(APP_KEY + "plotCh123Combined",
```

Appendix A – fftform.ui.h

```
    plotSettings.plotCh123Combined);  
}
```

fftformoptions.ui.h

This is the code for the spectrum settings window.

```

/*Class variables, in the real source code this is NOT placed here. .
int plotType; //Shows witch plot type the user has chosen
*/

void FftFormOptions::init()
{
}

//This functions is called from the FFT window, sets all the current values
//to the settings window
void FftFormOptions::setValues(int step, int redrawWait, int yScale,
                               int yScaleStep, int plotType, bool redraw,
                               bool ch1, bool ch2, bool ch3, bool combined)
{
    //Convert ms -> Hz and set the value to the Refresh rate spinbox
    refreshRateSpinBox->setValue(1000/redrawWait);
    stepSpinBox->setValue(step);
    yScaleSpinBox->setValue(yScale);
    yScaleStepSpinBox->setValue(yScaleStep);
    redrawCheckBox->setChecked(redraw);
    ch1CheckBox->setChecked(ch1);
    ch2CheckBox->setChecked(ch2);
    ch3CheckBox->setChecked(ch3);
    combinedCheckBox->setChecked(combined);

    //Sets the plot type
    if(plotType == 1)
        barRadioButton->setChecked(TRUE);
    else if(plotType == 2)
        lineRadioButton->setChecked(TRUE);
    else if(plotType == 3)
        barLineRadioButton->setChecked(TRUE);
    else
        filledRadioButton->setChecked(TRUE);
}

//When the user clicks the Ok button
void FftFormOptions::okPushButtonClicked()
{
    //See witch plot type the user has chosen
    plotTypeToggled();
    //Emit the settings, will be captured by the FFT window
    emit emitSettings(refreshRateSpinBox->value(),
                     stepSpinBox->value(),
                     yScaleSpinBox->value(),
                     yScaleStepSpinBox->value(),

```

Appendix A – fftformoptions.ui.h

```
        plotType,
        redrawCheckBox->isChecked(),
        ch1CheckBox->isChecked(),
        ch2CheckBox->isChecked(),
        ch3CheckBox->isChecked(),
        combinedCheckBox->isChecked());
    //Close the window
    accept();
}

//When a user changes the plot type
void FftFormOptions::plotTypeToggled()
{
    if(barRadioButton->isOn())
        plotType = 1;
    else if(lineRadioButton->isOn())
        plotType = 2;
    else if(barLineRadioButton->isOn())
        plotType = 3;
    else
        plotType = 4;
}

//When a user changes what to plot
void FftFormOptions::combinedCheckBoxClicked()
{
    ch1CheckBox->setChecked(FALSE);
    ch2CheckBox->setChecked(FALSE);
    ch3CheckBox->setChecked(FALSE);
}

void FftFormOptions::chCheckedClicked()
{
    combinedCheckBox->setChecked(FALSE);
}
```

plotGl.cpp

this is the code for plotting the spectrum

```

/*Class variables, in the real source code this is NOT placed here. .

double dataArrayLocal[3][2048][2];  Local data array,
                                   holds the spectrum to plot

int dataArrayLength;                Length of the array

GLuint  scale;    The Open GL list for the x- and y- axis
*/

#include "plotGl.h"
#include "../moc/moc_plotGl.cpp"

plotGIWidget::plotGIWidget(QWidget * parent, const char * name,
                           const QGLWidget * shareWidget, int wFlags) :
                           QGLWidget(parent, name, shareWidget, wFlags)
{
}
plotGIWidget::~~plotGIWidget()
{
}

//This function does all the drawing
void plotGIWidget :: paintGL()
{
    //If redraw is TRUE, clear the screen
    if(redraw) glClear(GL_COLOR_BUFFER_BIT);

    //Draw the scale, uses Open GL lists to speed up the drawing procedure.
    //This only works with objects that does not change their form
    glCallList(scale);

    //Draw the different spectrums
    if(plotCh1){glColor3f(1,0,0); drawCurve(dataArrayLocal[0]);}
    if(plotCh2){glColor3f(0,1,0); drawCurve(dataArrayLocal[1]);}
    if(plotCh3){glColor3f(0,0,1); drawCurve(dataArrayLocal[2]);}
    if(plotCh123Combined){glColor3f(1,1,0); drawCurve(dataArrayLocal[0]);}

    //Execute all the Open GL commands
    glFlush();
}

void plotGIWidget :: initializeGL()
{
    //Clear the screen and set background color to black

```

Appendix A – plotGl.cpp

```
    glClearColor(0.0,0.0,0.0,0.0);
    glColor3f(1.0,0.0,0.0);
}

//Automaticly run everytime the window changes it size
void plotGlWidget :: resizeGL(int w, int h)
{
    glMatrixMode(GL_PROJECTION);
    glViewport( 0, 0, (GLint)w, (GLint)h );
    glLoadIdentity();
    glOrtho(0, w, 0, h, -1.01, 1.01);
    glMatrixMode(GL_MODELVIEW);
    windowWidth = w;
    windowHeight = h;
    drawScale(); //Draw a new scale
}

void plotGlWidget :: drawScale()
{
    typedef GLfloat point2[2];
    point2 t, p;
    int j;
    double i;
    const int offset = 30, xMax = 1000, xStep = 100;
    double u = (windowWidth - 2 * offset) / (xMax / xStep);
    double v = (windowHeight - 2 * offset) / (yMax / yStep);

    scale=glGenLists(1);
    glNewList(scale, GL_COMPILE);

    glColor3f(1.0,0.0,0.0);

    glBegin(GL_LINES);

    //X- axis and arrow
    p[0] = 25;
    p[1] = offset;
    t[0] = windowWidth-20;
    t[1] = offset;
    glVertex2fv(p);
    glVertex2fv(t);

    p[0] = windowWidth-20;
    p[1] = offset;
    t[0] = windowWidth-40;
    t[1] = 10;
    glVertex2fv(p);
    glVertex2fv(t);
}
```

Appendix A – plotGl.cpp

```
p[0] = windowWidth-20;
p[1] = offset;
t[0] = windowWidth-40;
t[1] = 50;
glVertex2fv(p);
glVertex2fv(t);

//Y- axis and arrow
p[0] = offset;
p[1] = windowHeight-20;
t[0] = offset;
t[1] = 25;
glVertex2fv(p);
glVertex2fv(t);

p[0] = offset;
p[1] = windowHeight-20;
t[0] = 10;
t[1] = windowHeight-40;
glVertex2fv(p);
glVertex2fv(t);

p[0] = offset;
p[1] = windowHeight-20;
t[0] = 50;
t[1] = windowHeight-40;
glVertex2fv(p);
glVertex2fv(t);

glColor3f(1.0,0.0,1.0);

//Draw scale marks
for(i = offset; i < windowWidth - offset; i+=u){
    p[0] = (int)i;
    p[1] = offset - 5;
    t[0] = (int)i;
    t[1] = offset + 5;
    glVertex2fv(p);
    glVertex2fv(t);
}
for(i = offset; i < windowHeight - offset; i+=v){
    p[0] = offset - 5;
    p[1] = (int)i;
    t[0] = offset + 5;
    t[1] = (int)i;
    glVertex2fv(p);
    glVertex2fv(t);
}
glEnd();
```

Appendix A – plotGl.cpp

```

    glEndList();

    //Draw scale figures
}
void plotGlWidget :: drawCurve(double dataArray[][2])
{
    typedef GLfloat point2[2];
    point2 t, p, t1, tOld;
    int j, offset = 30;
    double i;

    //Draw bar spectrum
    if(plotType == 1 || plotType == 3){
        glBegin(GL_LINES);
        for(i = 0, j = 0; j < dataArrayLength; i+=drawStep, j++){
            p[1] = 30;
            p[0] = (int)i + 30;
            t[1] = dataArray[j][1] * (((double>windowHight - 2 *
                (double)offset) / (double)yMax)+30.0;
            t[0] = (int)i + 30;
            glVertex2fv(p);
            glVertex2fv(t);
        }
        glEnd();
    }
    //Draw line spectrum
    if(plotType == 2 || plotType == 3){
        glBegin(GL_LINE_STRIP);
        for(i = 0, j = 0; j < dataArrayLength; i+=drawStep, j++){
            t1[1] = dataArray[j][1] *
                (((double>windowHight - 2 * (double)offset) / (double)yMax)+30.0;
            t1[0] = (int)i + 30;
            glVertex2fv(t1);
        }
        glEnd();
    }
    //Draw filled spectrum
    if(plotType == 4){
        tOld[0] = 31;
        tOld[1] = 30;
        for(i = 0, j = 0; j < dataArrayLength; i+=drawStep, j++){
            glBegin(GL_POLYGON);
            t1[1] = dataArray[j][1] * (((double>windowHight - 2 *
                (double)offset) / (double)yMax)+30.0;
            t1[0] = (int)i + 30;
            glVertex2fv(t1);
            t1[1] = 31;
            t1[0] = (int)i + 30;
            glVertex2fv(t1);
            t1[1] = 31;
        }
    }
}

```

Appendix A – plotGl.cpp

```

        t1[0] = tOld[0];
        glVertex2fv(t1);
        glVertex2fv(tOld);
        tOld[1] = dataArray[j][1] * (((double>windowHight - 2 *
            (double)offset) / (double)yMax)+30.0;
        tOld[0] = (int)i + 30;
        glEnd();
    }
}

//This function is called from the plot window, it gives this drawing
//area new values
void plotGLWidget :: updatePlot(double dataArrayCh1[][2],
    double dataArrayCh2[][2],double dataArrayCh3[][2], int length){
    //Copy the incoming data array to the local array
    for(int i = 0; i < length; i++){
        dataArrayLocal[0][i][1] = dataArrayCh1[i][1];
        dataArrayLocal[1][i][1] = dataArrayCh2[i][1];
        dataArrayLocal[2][i][1] = dataArrayCh3[i][1];
    }
    //Length of the data array
    dataArrayLength = length;
    //Call paintGL()
    updateGL();
}

//Set different values to the drawing area
void plotGLWidget :: setValues(int step, int pT, int yM, int yS,
    bool rd, bool ch1, bool ch2, bool ch3, bool ch123){
    yStep = yS; //Y- scale Step
    yMax = yM; //Y- scale maximum
    drawStep = step; //Draw step along the x- axis
    plotType = pT; //Which plot type the user has chosen
    redraw = rd; //Clear the plot before each update
    plotCh1 = ch1; //Plot ch1 ?
    plotCh2 = ch2;
    plotCh3 = ch3;
    plotCh123Combined = ch123;
}

```

direcpolarform.ui.h

This is the baseform for the direction and polarization plot

```

/*Class variables, in the real source code this is NOT placed here. .
    int nNormFIFOCounter;           //Position in the FIFO buffer
    double nNormFIFO[100][3];       //The FIFO buffer
    bool direcPolarDone;           //SHows if the direction and
                                   //polarization are calculated
    double direcCoord[3];           //Direction coordinates for the plot
    int complexDataCounter;         //How many samples of each channel we
                                   //wan't to collect

    DirecPolarFormOptions *direcPolarSettings; //The settings window
    data plotSettings;             //The plotSettings structure
    complex<double> complexData[3]; //Incoming data
    QTimer *timer;                 //Timer
    direcPolarPlotGLWidget *plot;  //The Open GL drawing area
    double polarCoord[3];          //Polarization coordinates for the plot

    struct data{int redrawWait;     //Time between each update (ms)
                int rotateX;       //Rotation along the X -axis (degree)
                int rotateY;
                int rotateZ;};
*/

//Where to place our register keys, folder LOIS with the
//subfolder "DIGITAL_RECEIVER_DIRECPOLAR"
const QString WINDOWS_REGISTRY = "/LOIS";
const QString APP_KEY = "/DIGITAL_RECEIVER_DIRECPOLAR/";

//When the window is opened
void DirecPolarForm::init()
{
    //Load default settings from registry
    loadSettings();

    //Allocate memory for the Direction & plarization plot (the drawing area)
    plot = new direcPolarPlotGLWidget(this,"Plot GL",0,Qt::WDestructiveClose);

    //Set the size of the drawing area to the same as window size
    plot ->resize(size().width(), size().height());

    //Set different settings to the plot function
    plot ->setValues(plotSettings.rotateX,plotSettings.rotateY,
                    plotSettings.rotateZ);

    //Show the drawing area
    plot ->show();
}

```

Appendix A – direcpolarform.ui.h

```
complexDataCounter = 0;

//Where we are in the FIFO buffer, start at position 0
nNormFIFOCounter = 0;

//We haven't calculated the direction & polarization yet
direcPolarDone = FALSE;

//The timer gives a "interrupt" that tells us when it's time
//to update the plot
timer = new QTimer(this);
//When timer reaches 0 the updatePlot() function will be called
connect(timer, SIGNAL(timeout()), this, SLOT(updatePlot()));
//Start the timer
timer->start(plotSettings.redrawWait, TRUE );
}

//This function is automatically called when the window size changed
void DirecPolarForm::resizeEvent ( QResizeEvent *)
{
    //Resize the drawing area to fit the window
    plot -> resize(size().width(), size().height());
}

// Connect the receiveData() slot to the emitData signal() from main window
void DirecPolarForm::setParent(QWidget *parent)
{
    //When a package arrives, call receiveData()
    connect(parent, SIGNAL(emitData(complex<double>*)), this,
            SLOT(receiveData(complex<double>*)));
}

void DirecPolarForm::receiveData(complex<double> *data)
{
    //Copy samples from incoming buffer (data) to the local
    //buffer (complexData)
    for(;complexDataCounter < 3;complexDataCounter++)
        complexData[complexDataCounter] = data[complexDataCounter];

    //If we have one sample from each channel
    if((complexDataCounter >= 3) && !direcPolarDone){
        //The calculation is done
        direcPolarDone = TRUE;
        //Make the calculation
        calcDirecPolar();
    }
}

void DirecPolarForm::updatePlot()
```

Appendix A – direcpolarform.ui.h

```

{
    //The direction and polarization have been calculated
    if(direcPolarDone){
        //update the plot
        plot->updatePlot(direcCoord, polarCoord);

        //Allow the receiveData() to collect more samples
        complexDataCounter = 0;
        direcPolarDone = FALSE;

        //Redraw the timer
        timer->start(plotSettings.redrawWait, TRUE );
    }
    else
        timer->start(plotSettings.redrawWait/2, TRUE );
}

void DirecPolarForm::calcDirecPolar()
{
    double Hr[3] = {0,0,0}, Hi[3] = {0,0,0}, n[3], nNorm[3],
        nSum, fi, myS, my, s[3], sSum, nNormSum;
    complex<double> H[3], E[3],Efi, Emy, J[2][2];

    //Get the real and imaginary part of the three samples
    for(int j = 0; j < 3; j++){
        Hr[j] = real(complexData[j]);
        Hi[j] = imag(complexData[j]);
    }

    //Calculations made according to Appendix X
    n[0] = Hr[1] * Hi[2] - Hi[1] * Hr[2];
    n[1] = Hr[2] * Hi[0] - Hi[2] * Hr[0];
    n[2] = Hr[0] * Hi[1] - Hi[0] * Hr[1];

    nSum = n[0]*n[0] + n[1]*n[1] + n[2]*n[2];

    if(nSum > 0.0000001){
        //Insert values to the FIFO buffer
        for(int i = 0; i < 3; i++)
            nNormFIFO[nNormFIFOCounter][i] = n[i]/sqrt(nSum);

        (nNormFIFOCounter >= 99) ? (nNormFIFOCounter = 0) : nNormFIFOCounter++;

        for(int i = 0; i < 100; i++)
            for(int j = 0; j < 3; j++)
                nNorm[j]+= nNormFIFO[i][j];

        nNormSum = nNorm[0]*nNorm[0] + nNorm[1]*nNorm[1] + nNorm[2]*nNorm[2];
    }
}

```

```

//Direction calculation done, continues with the polarization
for(int i = 0; i < 3; i++)
    direcCoord[i] = nNorm[i]/sqrt(nNormSum);

fi = acos(nNorm[2]);

if(fi == 0 || fi == M_PI){
    //Abort the calculation
    complexDataCounter = 0;
    direcPolarDone = FALSE;
    timer->start(plotSettings.redrawWait/2, TRUE );
    return;
}
myS = asin(nNorm[1]/sin(fi));
if(myS >= 0)
    my = acos(nNorm[0]/sin(fi));
else
    my = -acos(nNorm[0]/sin(fi));

E[0] = complex<double>(Hr[1]*nNorm[2] - Hr[2]*nNorm[1],
                      Hi[1]*nNorm[2] - Hi[2]*nNorm[1]);
E[1] = complex<double>(Hr[2]*nNorm[0] - Hr[0]*nNorm[2],
                      Hi[2]*nNorm[0] - Hi[0]*nNorm[2]);
E[2] = complex<double>(Hr[0]*nNorm[1] - Hr[1]*nNorm[0],
                      Hi[0]*nNorm[1] - Hi[1]*nNorm[0]);

Efi = E[0] * cos(fi) * cos(my) + E[1] * cos(fi) * sin(my) -
      E[2] * sin(fi);
Emy = - E[0] * sin(my) + E[1] * cos(my);

J[0][0] = Efi * conj(Efi);
J[0][1] = Efi * conj(Emy);
J[1][0] = Emy * conj(Efi);
J[1][1] = Emy * conj(Emy);

s[0] = real(J[0][0] + J[1][1]);
s[1] = (2 * real(J[0][1]));
s[2] = (2 * imag(J[0][1]));

sSum = s[0]*s[0] + s[1]*s[1] + s[2]*s[2];

//Polarization calculation done
for(int i = 0; i < 3 ; i++)
    polarCoord[i] = s[i]/sqrt(sSum);
}
}

//The options menu
void DirecPolarForm::contextMenuEvent( QContextMenuEvent * )
{

```

Appendix A – direcpolarform.ui.h

```
QPopupMenu* contextMenu = new QPopupMenu( this );
Q_CHECK_PTR( contextMenu );
contextMenu->insertItem("Options", this, SLOT(plotSettingsWindow()));
contextMenu->exec(QCursor::pos());
delete contextMenu;
}

//This function is called when the user clicks Options in the menu
void DirecPolarForm::plotSettingsWindow(){
    //Allocate memory for the options window
    direcPolarSettings = new DirecPolarFormOptions(this);

    //Connect the "emitSettings" signal from the
    //settings window with the slot updatevalues in this window
    connect(direcPolarSettings, SIGNAL(emitSettings(int, int, int, int)),
            this, SLOT(updateValues(int, int, int, int)));

    //Set values to different fields in the settings window
    direcPolarSettings->setValues(plotSettings.redrawWait,
        plotSettings.rotateX,plotSettings.rotateY, plotSettings.rotateZ);

    //Bring up the settings window
    direcPolarSettings->exec();
}

//This function stores the settings made in the plotSettings structure
//and then updates the drawing area
void DirecPolarForm::updateValues(int rr, int x, int y, int z)
{
    plotSettings.redrawWait = (int)(1000/(double)rr);//Convert Hz -> ms
    plotSettings.rotateX = x;
    plotSettings.rotateY = y;
    plotSettings.rotateZ = z;
    plot->setValues(plotSettings.rotateX,plotSettings.rotateY,
        plotSettings.rotateZ);
}

//Loads settings from the registry and stores them in the
//plotSettings structrure
void DirecPolarForm::loadSettings()
{
    QSettings settings;
    settings.insertSearchPath( QSettings::Windows, WINDOWS_REGISTRY);
    int windowX = settings.readNumEntry(APP_KEY + "WindowX", 0);
    int windowY = settings.readNumEntry(APP_KEY + "WindowY", 0);
    plotSettings.redrawWait = settings.readNumEntry(APP_KEY + "redrawWait",50);
    plotSettings.rotateX = settings.readNumEntry(APP_KEY + "rotateX", 0);
    plotSettings.rotateY = settings.readNumEntry(APP_KEY + "rotateY", 0);
    plotSettings.rotateZ = settings.readNumEntry(APP_KEY + "rotateZ", 0);
    move(windowX, windowY);
}
```

Appendix A – direcpolarform.ui.h

}

direcpolarformoptions.ui.h

This is the code for the direction and polarization options window.

```

//This event occurs when the user changes the
//x- slider (rotates the sphere along the x -axis)
void DirecPolarFormOptions::xSliderMoved(int value)
{
    xLineEdit->setText(QString( "%1" ).arg(value));
    valueChanged();
}
void DirecPolarFormOptions::ySliderMoved(int value)
{
    yLineEdit->setText(QString( "%1" ).arg(value));
    valueChanged();
}
void DirecPolarFormOptions::zSliderMoved(int value)
{
    zLineEdit->setText(QString( "%1" ).arg(value));
    valueChanged();
}

void DirecPolarFormOptions::okPushButtonClicked()
{
    valueChanged();
    accept();
}

//Set current values to the sliders and the spinbox
void DirecPolarFormOptions::setValues(int redrawWait, int x, int y, int z)
{
    refreshRateSpinBox->setValue(1000/redrawWait);
    xSlider->setValue(x);
    ySlider->setValue(y);
    zSlider->setValue(z);
}
//Emit settings, captured by the updateValue()
// in the polarization and direction window
void DirecPolarFormOptions::valueChanged()
{
    emit emitSettings(refreshRateSpinBox->value(),
                     xSlider->value(),
                     ySlider->value(),
                     zSlider->value());
}

```

direcpolarplotgl.cpp

this is the code for plotting the direction and polarization.

```
#include "direcpolarplotgl.h"
#include <math.h>
#include <GL/glut.h>
#include "../moc/moc_direcpolarplotgl.cpp"

direcPolarPlotGLWidget::direcPolarPlotGLWidget(QWidget * parent,
        const char * name, const QGLWidget * shareWidget, int wFlags) :
        QGLWidget(parent, name, shareWidget, wFlags)
{
}

direcPolarPlotGLWidget::~direcPolarPlotGLWidget()
{

}

//This function does all the drawing
void direcPolarPlotGLWidget :: paintGL()
{
    int radius;
    //Clear screen
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    //Calculate the sphere radius
    if(windowWidth < windowHeight)
        radius = windowWidth/2 - 60;
    else
        radius = windowHeight/2 - 60;

    //Put us in the middle of the screen
    glTranslatef( windowWidth/2, windowHeight/2, 0 );

    //Do some rotation,
    //values from the x, y and z sliders in the settings window
    glRotatef(rotation[0],0,1,0);
    glRotatef(rotation[1],1,0,0);
    glRotatef(rotation[2],0,0,1);

    //Draw the axis
    glBegin( GL_LINES );
    glColor3f(0.0,1.0,0.0);
    glVertex3f(-radius,0,0); glVertex3f(radius*1.1,0,0);

    glColor3f(0.0,0.0,1.0);
```

Appendix A – direcpolarplotgl.cpp

```
glVertex3f(0,-radius,0); glVertex3f(0,radius*1.1,0);

glColor3f(1.0,0.0,0.0);
glVertex3f(0,0,-radius); glVertex3f(0,0,radius*1.1);

//Draw the direction vector
glColor3f(1,1.0,0);
glVertex3f(0,0,0); glVertex3f(direcCoord[1]*radius,
                             direcCoord[2]*radius,direcCoord[0]*radius);

//Draw the polarization vector
glColor3f(0.0,1.0,1.0);
glVertex3f(0,0,0); glVertex3f(polarCoord[1]*radius,polarCoord[2]*radius,
                             polarCoord[0]*radius);

glEnd();

//Draw the x- sphere
glColor3f(0.0,1.0,0.0);
glBegin(GL_LINE_LOOP);
for(float ang=0; ang <= 2*M_PI; ang += 0.1)
    glVertex3d(radius * cos(ang), radius * sin(ang), 0);
glEnd();

//Draw the y- sphere
glColor3f(0.0,0.0,1.0);
glBegin(GL_LINE_LOOP);
for(float ang=0; ang <= 2*M_PI; ang += 0.1)
    glVertex3d(0, radius * cos(ang), radius * sin(ang));
glEnd();

//Draw the z- sphere
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
for(float ang=0; ang <= 2*M_PI; ang += 0.1)
    glVertex3d(radius * cos(ang),0,radius * sin(ang));
glEnd();

//Move to the outer direction vector and draw a small sphere
glTranslatef(direcCoord[1]*radius,direcCoord[2]*radius,
            direcCoord[0]*radius );
glColor3f(1.0,1.0,0.0);
glutSolidSphere(10, 20, 20);

//Move to the outer polarization vector and draw a small sphere
glTranslatef(-direcCoord[1]*radius + polarCoord[1]*radius,
            -direcCoord[2]*radius + polarCoord[2]*radius,
            -direcCoord[0]*radius+polarCoord[0]*radius);

glColor3f(0.0,1.0,1.0);
glutSolidSphere(10, 20, 20);
```

Appendix A – direcpolarplotgl.cpp

```
//Execute all Open GL commands
glFlush();
}
void direcPolarPlotGIWidget :: initializeGL()
{
    //Clear the screen and set background color to black
    glClearColor(0.0,0.0,0.0,0.0);
    glColor3f(1.0,0.0,0.0);
    //enables depth and smothing
    glShadeModel(GL_SMOOTH);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

}

//Automatically run everytime the window changes it size
void direcPolarPlotGIWidget :: resizeGL(int w, int h)
{
    glViewport( 0, 0, (GLint)w, (GLint)h );
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, w, 0, h, -1000, 1000);
    glMatrixMode(GL_MODELVIEW);
    windowWidth = w;
    windowHeight = h;
}

//This function is called from the direcPolarForm, it gives this drawing
//area new values
void direcPolarPlotGIWidget :: updatePlot(double dCoord[3], double pCoord[3]){
    for(int i = 0; i < 3; i++){
        direcCoord[i] = (GLfloat)dCoord[i];
        polarCoord[i] = (GLfloat)pCoord[i];
    }
    updateGL();
}

//This functions sets the rotation. It is called from the direcPolarForm
void direcPolarPlotGIWidget :: setValues(int x, int y, int z){
    rotation[0] = (GLfloat)x;
    rotation[1] = (GLfloat)y;
    rotation[2] = (GLfloat)z;
}
```



Växjö
universitet

Matematiska och systemtekniska institutionen
SE-351 95 Växjö

tel 0470-70 80 00, fax 0470-840 04
www.msi.vxu.se