UPTEC F11037
Examensarbete 30 hp
Maj 2011

Ångström Small Radio Telescope

Henrik Lindén

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Ångström Small Radio Telescope

Henrik Lindén

For the Swedish Institute of Space Physics and Uppsala University, we have developed a working radio astronomy telescope capable of receiving the 21 cm hydrogen line; the Ångström Small Radio Telescope. The work have resulted in a functional system for positioning the dish, with built in tracking of deep space objects and scanning functions, and signal reception with filtering, mixing and digital sampling. The system is controlled via a computer through an Internet connection.

Handledare: Jan Bergman
Ämnesgranskare: Nikolai Piskunov
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F11037

Sammanfattning

För Institutet för Rymdfysik och Uppsala Universitet, har vi tagit fram att fungerande radio teleskop med kapacitet att ta emot den så kallade vätelinjen, kallat Ångström Small Radio Telescope (ÅSRT). Resultatet blev ett fungerande system för positionering, med funktioner för att följa så kallade deep-space-objekt och för att kunna göra avläsningar av hela himmeln, samt mottagning av signalen med filtrering, mixning och digital sampling. Målet har varit att systemet ska användas som laborationsutrustning av studenter i kurser om astronomi, för att de på en enkel nivå ska lära sig hur mycket av dagens forskning kring rymden som ej är direkt visuell fungerar. Eftersom systemet bygger på exakt samma principer som de stora professionella teleskopen som kan vara flera hundra meter i diameter, kommer eleverna att även kunna få en god förståelse för hur den rent tekniska delen av teleskopet är uppbyggd. Mottagarsystemet gör teleskopet även intressant för studenter med fokus på mikrovågsteknik och elektronik.

Vätelinjen i sig är intressant då väte är det mest fundamentala grundämnet i universum, i och med det så kan man bl.a. analysera spridningen av väte, och på så vis få mycket intressant information om universums utveckling. Det finns även användning för teleskopet i lite mera professionell forskning exempelvis för detektion av pulsarer. För systemet finns också förhoppningar om utbyggnad till ett interferometerteleskop eftersom ytterligare en parabol finns tillgänglig för detta ändamål, vilket kan leda till ytterligare intressanta möjligheter.

Acknowledgments

Supervisor

Dr. Jan Bergman, Swedish Institute of Space Physics

Assistant Supervisor

Prof. Anders Rydberg, Signals and Systems group, Department of Engineering Sciences

Assitant Supervisor

Dr. Roger Karlsson, Department of Physics and Astronomy

Subject Examiner

Prof. Nikolai Piskunov, Department of Physics and Astronomy

Electronics & Motor Control

Sven-Erik Jansson, Swedish Institute of Space Physics
Walter Puccio, Swedish Institute of Space Physics
Lennart Åhlén, Swedish Institute of Space Physics
Farid Shiva, Swedish Institute of Space Physics

Contents

Chapter 1

Introduction

The Ångström Small Radio Telescope consists of two 2.3 m diameter parabolic dishes located on the roofs of house 7 and 8 of the Ångström laboratory in Uppsala, Sweden. It has been a long going project which was put on ice a few years ago in favor of other projects, and has since then been left unattended. The purpose of this work is to make one of the dishes fully functional. Consequently some parts of the project was already started and could be continued upon. The first part of this work was therefore to analyze what parts of the project had been worked upon and which could be used to finish it. It turned out there where actually two separate attempts made to get the ÅSRT running: one by scientists at the Institute of Space Physics in Uppsala (IRFU), right after the acquisition of the parabolic dishes, and another attempt by Professor Anders Rydberg at the department of Signals and Systems at the University of Uppsala, some time later after the first attempt was canceled. These projects where performed separately on the different dishes and therefore had completely different control systems. To separate the different dishes we have given them names to represent their placement on the roof of the laboratory, and is consequently called the Ångström 7 and Ångström 8 dishes, Å7 and Å8 for short.

The major obstacle to get a running system was to get the motor control system working and this had been attempted in both previous efforts. The second attempt by Professor Rydberg was made using a system made in the U.S. that was originally made to match another system, but was thought to be able to control the ÅSRT dishes as well, although with some modifications. The trail used the Å7-dish and was partially successful but a few problems where not solved. At IRFU the project had halted after a controller-chip box was completed and most elementary command options, like positioning, reset, and stop had been implemented, using the Å8-dish. As Sven-Erik Jansson, the builder of the controller-chip box, was available during this project and the other controller system had quite little documentation, improving the existing design on the Å7-dish seemed impractical. It was decided that the IRFU system would be the one to be continued upon. Also, some modifications that had been made to the Å7-dish to adapt it to the U.S. controller made it more difficult to get started with.

1.1 The Parabolic Dishes

The parabolic dishes were purchased in 2005 from the Onsala Space Observatory [1], which in turn imported four such telescopes from the MIT Haystack Observatory [2] who manufacture and distribute these small radio telescopes (SRT). The original design mounted the dish using a single 180 degree motor, implying only elevation could be varied. To improve this the engineers at Onsala stacked two motors on top of each other tilting the bottom one 90 degrees to have the antenna move in both azimuth and elevation. On arrival at Uppsala some calibration to these motors was made and a stable tripod was constructed for each antenna. The technical specifications can be seen in Table 1.1, and as stated they can also handle frequencies above 1.42 GHz. However the

Table 1.1: Antenna Specifications[3]

Diameter	2.3 m
Focal Length	85.7 cm
Gain at 4.2 GHz	38.1 dBi
Gain at 1.4 GHz	28.1 dBi
Beam Width	7.0 Degrees at 1.4 GHz
Position Å8	$N59°50.264'$; $E38°38.924'$

specifications regarding the gain at 1.42 GHz was not provided and had to be calculated using the approximative formula [4].

$$\text{Gain at 1.42 GHz} = 17.8 + 20\log(\text{Antenna Diameter}) + 20\log(\text{Frequency}) \tag{1.1}$$

The feed horn on the dishes contain two individual dipole antennas mounted with a 90 degree inclination to one another. The fact that the two are rotated differently enables us to measure the different polarizations of the signal. This also means that we are provided with two signals which we need equipment to receive and process.

(a) (b)

Figure 1.1: The Antenna (a), with the dipoles (b)

Chapter 2

Radio Astronomy

2.1 History

The field of radio astronomy is a quite young science. Its basis was discovered during an attempt to minimize noise in telephone lines by Bell employees [5], around 1935. It was found that the interference varied during the day and the radio source was first believed to be the sun, but more investigation indicated that it came from another point in space which later was shown to be the center of our galaxy.

In the early 1950s the first conclusive measurements were made of the always so popular 21 cm Hydrogen line and radio astronomy was shown to be a field with great promise. Nowadays, radio astronomy is a vital part in our understanding of the universe, e.g. we have the cosmic microwave background radiation giving us a glimpse of the structure of the early universe when only a few hundred thousand years old. The Onsala Space Observatory situated on the coast south of Gothenburg, is the institution that conduct most of the radio astronomy research in Sweden. They have two large parabolic dish antennas, 20 m and 25 m in diameter, doing measurements of molecules and stars, and also international collaborations like VLBI (Very Long Baseline Interferometry). They also have the SRT:s mentioned earlier called SALSA [6], which is mostly used for experiments by students. The largest dish-telescope ever made is the Arecibo Radio Telescope in Puerto Rico, it spans a total 305 m [7] in diameter and is used for all kinds of research, some of the runtime is even allotted for use in the SETI project [8], Search for Extra-Terrestrial Intelligence. The Arecibo telescope is built inside a valley, or sinkhole, is fixed to the ground and steers the antenna beam by moving the receiver horn instead of moving the entire antenna.

Other notable telescopes are the Radio Telescope Effelsberg at the Max Planck Institute for Radio Astronomy in Bonn, Germany. It is the largest telescope in Europe with its 100 m diameter dish, which is also fully mechanically steerable, 360 degrees azimuth and 90 degrees elevation [9]. We also have the new Chinese telescope FAST (Five hundred meter Aperture Spherical Telescope) [10], which will be completed in 2013. It will be the worlds largest single-aperture telescope with a diameter of 500 m, and will be situated inside a natural hollow just like the Arecibo telescope. An even larger telescope KARST (Kilometer-square Area Radio Synthesis Telescope) [11], which will be a successor to FAST, is also planed in China.

2.2 The Hydrogen line

The hydrogen line is an electromagnetic wave at a frequency of 1.42 GHz and a wavelength of 21 cm, which is in the microwave region. It is created by energy released during a forbidden transition in natural hydrogen atoms. A hydrogen atom consist of one electron and one proton which have their own spin states. As only two spin states are possible for each of the particles,

they can either have parallel spin or anti-parallel spin with respect to each other. In a quantum mechanical analysis it can be shown that the parallel state has a slightly higher amount of energy, which is released when a transition to the anti-parallel state occurs for one of the particles. The energy released forms the so called 21 cm hydrogen line. This is however a so called forbidden transition, meaning that it has a very low probability to occur, only about $2.6 \cdot 10^{-15} s^{-1}$ [12]. Fortunately the amount of hydrogen atoms in the interstellar medium is so incredibly large that the transition can be detected continuously with radio telescopes.

2.2.1 Why use the hydrogen line?

The hydrogen line has many properties which are important when working in the field of radio astronomy. Hydrogen is the simplest of all natural elements and has been around since shortly after the big bang, and because of that one can see important structures regarding the evolution of the Universe, galaxies, and solar systems. Other elements also send out signals as part of similar processes but they transmit even higher frequencies, which places higher demands on the equipment used for detection. A great advantage with microwaves is that they do not dissipate in the atmosphere, unlike lower frequencies below 100 MHz, which interact with the ionosphere and are very difficult or even impossible to detect from Earth. For those frequencies one can use radio telescope satellites but very few have been built. There has also been international agreements to only use the hydrogen line for radio astronomy and not for any commercial or military transmissions, though some countries use frequencies very close to the line which might yield some interference.

2.3 Possible Experiments

When the ÅSRT is operational there are some interesting research that could be accomplished either with one or both of the dishes.

2.3.1 Hydrogen Distribution

The first and most basic experiment is to do a mapping of the distribution of natural hydrogen in the Universe, we will in all likelihood not however be able to detect any signals beyond our own galaxy, since they will be much to weak. Mapping is accomplished by pointing the telescope at some elevation while Earth rotates before it and then repeating this for all elevations until the entire hemisphere has been scanned. However a blind region will occur since we can not watch further than the horizon. A scan made by another telescope of the same model can bee seen in Fig. 2.1. The figure shows the difference in intensity in the various areas, with the concentration being the highest in the plane of the Milky Way galaxy. A continuation of this mapping is to use both of the dipoles on the dish, since they are oriented perpendicularly they are able to pick up different polarizations of the same signal. It would be interesting to be able to compare the mappings with different polarizations and see how they differ in intensity. The radio galaxy Cygnus A which is one of the strongest radio sources we can observe in the sky has for example circular polarization [13] which we could detect and make comparisons.

2.3.2 Interferometry

At the time when both dishes are operational they can be used for interferometry, which will increase the precision of the system. Radio interferometry uses a method called aperture synthesis in which the collection of baselines between the dishes each yields a component for the Fourier transform of the brightness of the object observed [14]. Compared to using a single dish, a radio interferometer is quite complicated. Each of the dishes must measure both amplitude and phase of the signal, synchronously using GPS or other high accuracy time source. One benefit of using two dishes rather than one is the larger collecting area, which makes the telescope more sensitive,

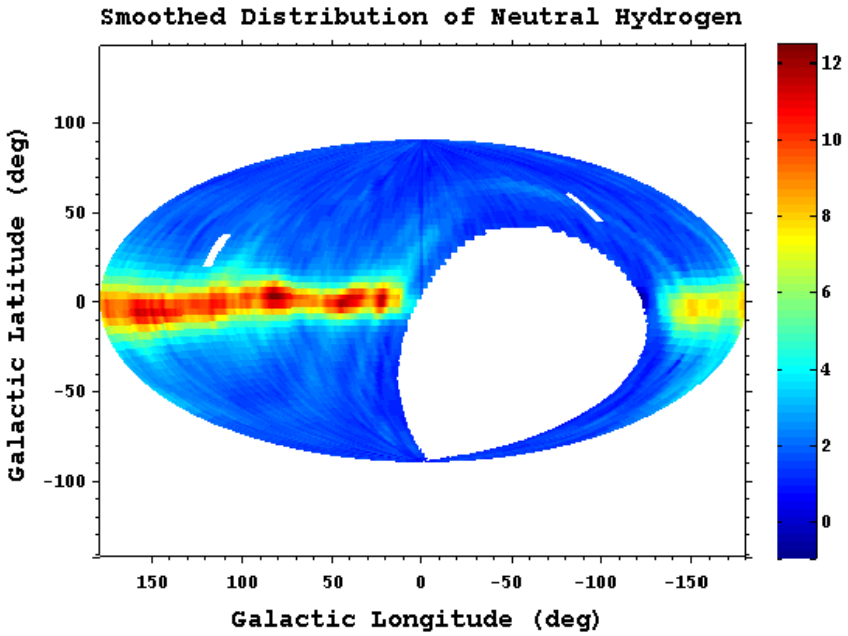

Figure 2.1: Hemispherical scan made by the small radio telescope at Glasgow University Observatory, showing the distribution of hydrogen.

but the greatest advantage is the vastly improved resolution, which scales as λ/D, where λ is the wavelength of the signal and D is the distance between the dishes. If one of the dishes were mobile, *i.e.* placed on a rail, one could change the baseline length and get more components for the Fourier transform resulting in better images of the object.

2.3.3 Pulsars

An interesting experiment would be to try to detect the high energy bursts emerging from pulsars. The idea would be to track a pulsar for a long time and measure the pulses as they pass by. The pulses would be largely widened after traveling through the plasma in space but by stacking all of the measurements one would slowly build up a clear signal.

Chapter 3

Control System

3.1 Introduction

A fundamental function of the ÅSRT is to be easily controlled by both students and researchers, which could have largely different understandings of the system. Software for and by researchers are often made with text based interaction since it requires the least amount of effort to get the program running. Since the ÅSRT will be used primarily by students in experiments the software needs to be quite intuitive, this is most easily accomplished if the program has a Graphical User Interface, also known as GUI. GUI:s also save a lot of time since scripts or loops can be streamlined more easily and run by the press of a button instead of writing a command in a terminal window. In our opinion, it also looks more appealing and professional.

3.2 The Controller-box

The controller-box is a hardware controller made to control the movement of the Å8-dish. The dish has two engines, one for moving in the azimuth direction and one for the elevation. The dish is calibrated so that the zero of azimuth is pointing north. However, because the engines can only move 180 degrees each in their respective directions, the reset point was chosen to be azimuth 270 degrees and elevation 0 degrees, which makes the dish point west. The way reset works is that both motors move until they reach their maximum range, always in the same direction. The benefit of using azimuth 270 degrees as reset is that when commanding the dish, the motors would need to move a shorter distance overall since the azimuth would move as much in the east as in the west quadrant. This is assumed since the preferred objects would more often differ in the east-west direction than in the north-south direction, based on the rotation of the Earth being west-east, and thereby limiting the amount of times the elevation has to do a complete flip-over to cover objects in those areas.

The box has an integrated power supply for the motors, delivering a voltage of 24 V and the motors combined require a current of about 1.8 A. The current is limited to 2 A through a fuse for safety reasons, since the power supply has the ability to deliver up to 4 A. The controller uses a microprocessor from Digi® called 'Rabbit 3000' which handles all of the instructions. For communication the Rabbit 3000 is equipped with an Ethernet connection and uses the TCP/IP protocol. This results in some helpful features: the controller does not have to be positioned relatively close to the user like the use of for example serial ports might require, and it can be controlled from any computer as long as one has the GUI-software.

3.3 Programming Language

When writing a computer program there are a lot of programming languages to choose from. Depending on what results the project aims for some languages are more appropriate than others. My own experience in programming involves C++, Java, and Visual Basic. All are sufficiently high level languages suitable for this project. As my knowledge in working with larger software projects is poor, I could not rely much on previous experience when choosing a programming language. Instead, I set some goals to help single out the most appropriate of these languages for this work. Using a language of which I was not familiar with would waste time in development.

The program was to be written primarily for a Windows environment as it is the most used operating system and it should be familiar to most of the users. It should however be noted that making a Linux compatible version as well, should be possible. The program should also be somewhat low on system resources as it would later be required to save data as it was received from the dishes, which might require data processing on arrival.

- Visual Basic
  As Visual Basic is a Windows-only language it was not considered to be applicable, as it would not enable the program to be easily changed and adapted for Linux if desired.
- Java
  One advantage of Java is that it is quite easy to use across multiple operating systems. It was also the language I had the most recent experience with so it could be easier to work with. But Java requires separate software installation on every computer used for the controller program, which makes it less user friendly for students and researchers. The Java software must also be run in the background to enable the controller to work, which consumes a lot of computer resources making Java a bad alternative.
- C++
  Having a reputation of low resource consumption, if programmed correctly, C++ has become one of the most popular programming languages. Low resource consumption comes from that C++ is somewhat less a high level language than the others, giving the programmer more control of the underlying systems but it also makes the programming a little more time consuming. It can be used for both Windows and Linux systems, making it the most viable choice of the considered languages. Additionally, since the controller-box firmware is written in C and some firmware modification might be needed, it would be useful to use C++ for the GUI since they are fundamentally the same, making the overall development simpler.

3.4 GUI Library

C++ does not contain any library for making graphical user interfaces. To create GUI:s in C++ one must either write a custom GUI library or use an external library which adds the required functions to C++. Some of the most popular ones are Visual C++, GTK+ and Qt. Visual C++ like Visual Basic is used only in the Windows environment and would not cover our needs. GTK+[15] and Qt[16] are both multi platform supported but suffers from kind of the same issue, the library is not standard in either Windows nor Linux and must be installed on the computer running the program. There are however ways to work around this and Qt was deemed the easiest, where only a few files placed in the folder with the executable-file would do the trick. More advanced measures like building the program 'static', where the required files are included inside the executable, is also supported by Qt.

3.4.1 Qt

Qt is currently being developed by Nokia and has a syntax very similar to C++. It contains functions specifically for developing GUI:s, and can be used with or without graphical designer

Table 3.1: Commands for the controller-box

Command string	Description
Az,X;El,Y	Move the dish to aim at the point with azimuth X and elevation Y. X and Y are integer degrees of azimuth and elevation respectively.
stop	Stop the movement before it has reached its target position.
reset	Reset the dish to the starting point.
ver	Send the version number of the current firmware.

tools. The graphical tool named Qt Designer makes the design part a lot easier, it has a simple drag-and-drop system and saves hours upon hours of time, which would otherwise be spent coding the GUI [17].

3.5 Commands

The main function of the GUI is to send commands to the controller as per the user's request. The controller was setup to receive all commands as text strings, the most fundamental functions of the controller can be seen in Table 3.1.

These commands were handled quite well by the controller at the start of the project but a few bugs in the firmware had to be corrected as it progressed.

3.6 Functionality

To make the usage more streamlined the GUI was extended with other functions than the basic commands. The functions are made to simplify the use of the telescope, so that it can operate with as little interaction from the user as possible. Since most research will require the telescope to run for a very long time and changes in position is very common.

3.6.1 Scanning

One of the goals for this project was to be able to create a hydrogen-map of the hemisphere. For this to be possible some functions had to be developed which would automate the movement of the dish, and there are in principle two possible methods to reach the desired result. One method known as the drift-scan technique, is to have the dish fixed at a certain elevation and have Earth rotate before it, collecting data during one full revolution, 24 hours that is. Then increasing the elevation and repeating the procedure up to 180 degrees (180 refers to the motors total movement range and not the coordinate system), see Fig. 3.1. Depending on the beam width of the dish this will take a long time. Using the 7 degree beam width of our antenna one can calculate it to take almost 26 days to map the sky completely using this method. This is good if high sensitivity is a priority, but far too long to be satisfactory when using the telescope for lab work with students.

A less time consuming method would be to have the dish scan vertically instead. The Earth move about 15 degrees in one hour, the dish must then elevate 180 degrees and back over approximately that time to be able to scan the hemisphere, see Fig. 3.2. This will result in a total scan only taking 24 hours to complete, much more reasonable for lab work. It will however result in more complex handling of the data since a lot of overlapping will occur. The short scanning time was deemed the more important component and the work was centered around adapting the scanning routine for it.

After the development of the scan it was realized that with some modification the function could handle both patterns, it will however require some adaptation from the user.

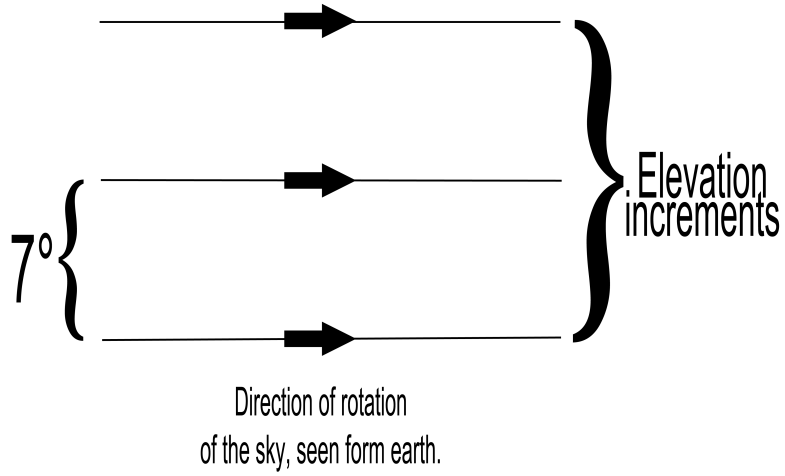

Figure 3.1: Movement pattern of how a horizontal drift-scan would cover the hemisphere in 26 *days.*

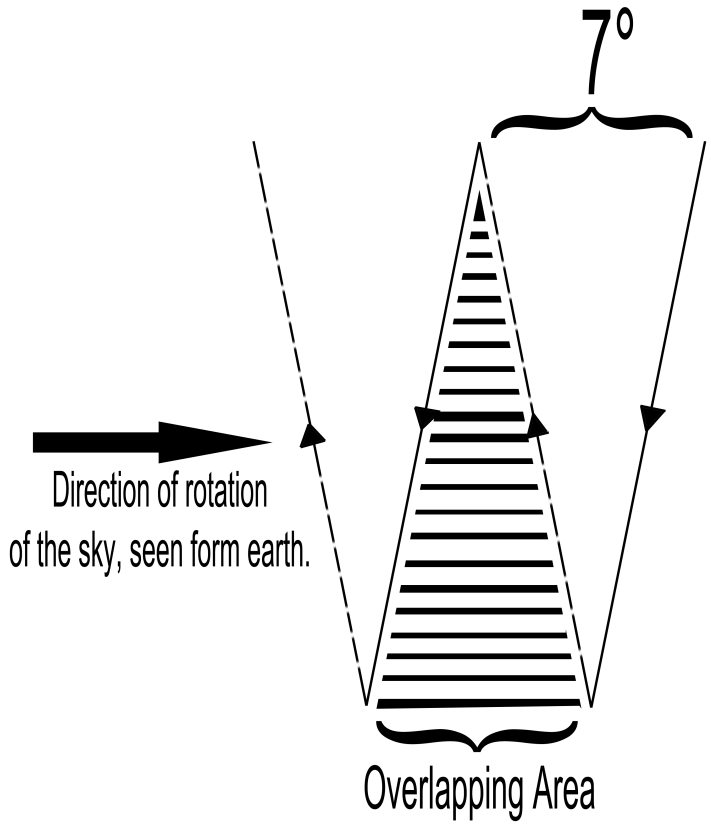

Figure 3.2: Movement pattern of how a vertical scan would cover the hemisphere in only 24 *hours. Some overlapping will take place.*

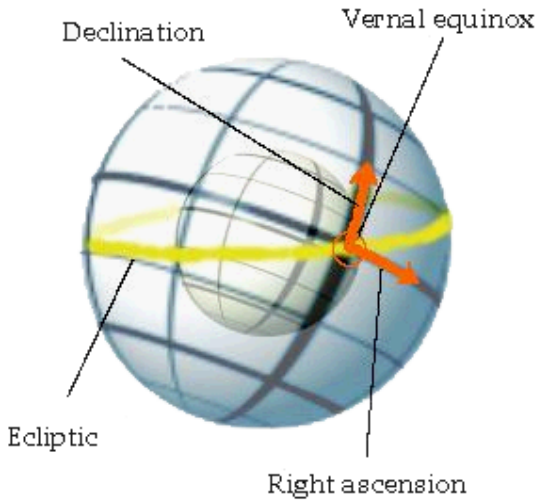

Figure 3.3: Equatorial Coordinate system, with Earth inside a depiction of the hemisphere. The Earth is tilted to match the difference between the ecliptic and equatorial plane. (© Wikipedia under GFDL.)

3.6.2 Tracking

After mapping the sky some other observations may be of interest, tracking objects is one of them. All objects that can be considered fix, *i.e.* stars, galaxies and other radio sources have been given permanent coordinates in the equatorial coordinate system [18] expressed in what is called right ascension and declination, see Fig. 3.3. Consider the plane in which the Sun moves during one day seen from Earth, this is called the ecliptic. Then take the plane made up of Earth's equator (the equatorial plane) the point where they cross is called the vernal equinox and is the starting point for right ascension and declination. The important thing to notice is that these coordinates mark where objects are positioned when observed from a point on Earth. Some telescopes use equatorial mountings and gradings, which makes for easy observation: align with the northern star and you can simply set the position of your object.

If the telescope does not have an equatorial mount, like in our case where instead there is a horizontal mount, commonly know as an altazimuth mount, pointing becomes more difficult since some coordinate transformation will be needed. When transforming from equatorial coordinates to horizontal one must be aware that the objects position changes with time not only on small scales but also on large scales. For this purpose one follows an algorithm which takes Earths movement variations into account and yields the altitude (angle of elevation) and azimuth of the object. To track an object one must simply repeat the algorithm continuously during the observation, and each time a slight variation in position will occur. The motors on the dish have a precision of 1 degree, which means that continuous exact calculations are not needed, and the algorithm is only repeated once every minute. Equatorial coordinates are often given in hours, minutes and seconds instead of degrees, to convert use the 15 degree per hour rotation of Earth.

Giving the system capability to track satellites and planets have also been considered, but time constraints made this less of a priority and so it was not implemented.

3.7 User Information

The GUI have been split into a few separate areas which deal with the different functions of the program. See Fig. 3.4 when referred to the different areas and use of functions.

3.7.1 Dish Mode

The *Dish Mode* area is where one chooses the running mode to be used; either running the dishes separately or in interferometric mode. The GUI has been adapted for using both antennas even though currently, only one dish is operational. For each dish there is an IP address and a port to be specified. The program have default values in the configurations-file in the program conf-folder, values which can be reset from the *Menu* tab. In this area you also *Connect* and *Disconnect* from the specified dish.

3.7.2 Fix Position

After connecting to a dish one can use the other parts of the program, next is the *Fix Position* area. Here you can set the dish to move to a specific position, either in horizontal or equatorial coordinates. Choose coordinate system, enter the coordinates, and press *Set Fix Point* and the dish will start moving toward that point. When the dish has reached is destination, the *Current Position* label will be updated with the new position, in both coordinate systems. When you have set a position you can continue with tracking that specific point in the sky, e.g. if you use the default equatorial position you can track the Andromeda galaxy. You are also required to set a time limit for how long you wish to track the object, up to 24 hours is possible.

3.7.3 Hemispherical Scan

For mapping purposes there is the *Hemispherical Scan* area, this will have the dish running the scan algorithm discussed before. This part has also got a required time limit with a maximum of 24 hours.

3.7.4 Other Functions

Both scanning and tracking have their current progress shown in the bottom progress bar for easy observation. More extensive information on the system is shown in the *Controller log* window. Here you can see tracking and scanning information and also communication from the Controller-box which is displayed in red to highlight them. There are also two important buttons in the top right of the program; the *Stop* and *Reset Position* buttons. Their implementation are quite straight forward, *Stop* immediately stops the dish in its current movement and also stops any running scan or tracking sequences, while *Reset Position* moves the dish to its default position.

NOTE! It is recommended to start each new run of the system with resetting the dish to azimuth 270 degrees **and elevation** 0 degrees**, using the *Reset Position* button.**

More information on the GUI can be found in the program's help-section, under the *Help* tab.

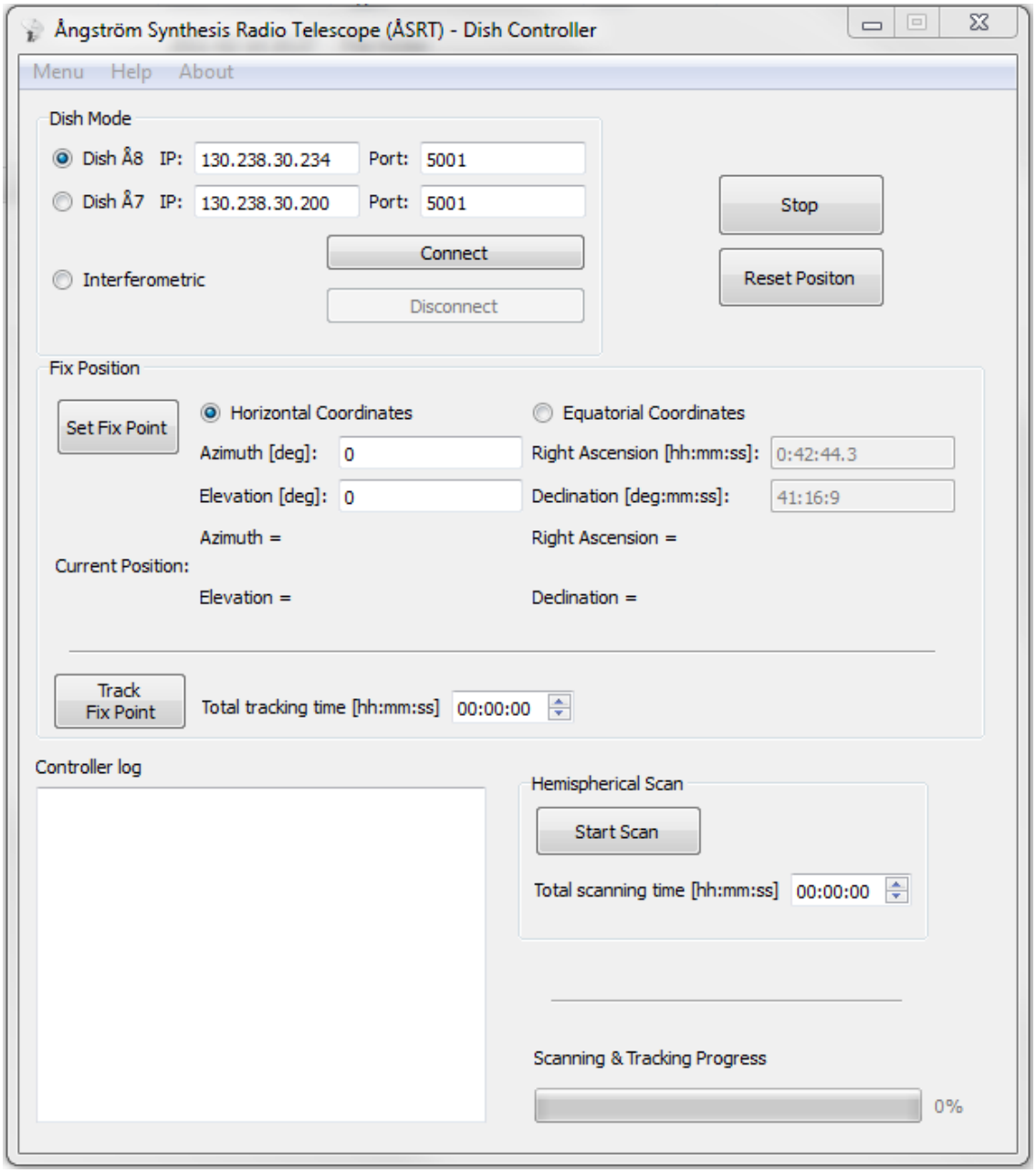

Figure 3.4: The GUI in its final stage, version 0.8.

Chapter 4

Radio System

Compared to sound waves, which are audible below 20 kHz and easily received and sampled by a computer's sound system, microwaves oscillate about a million times faster and requires an external receiver. Basically, there are three different techniques to design a sampling receiver.

Digital direct conversion using Nyquist sampling

The radio frequency (RF) signal is lowpass (LP) filtered and sampled at at least two times the frequency of the highest desirable frequency, so called Nyquist sampling. In our case that would be almost 3 GHz to leave a margin for the LP filter fall-off. Analog-to-Digital-converters (ADC) with this performance are available but they are very expensive and require other very fast, and also expensive, components to digitally mix and bandpass (BP) filter the signal to a manageable data rate.

Analogue baseband receiver using I-Q sampling

The RF signal is converted to baseband by analogue mixing and then LP filtered. This makes the center (mixer) frequency appear at 0 Hz. To correctly handle frequencies below the center frequency, which appears to be negative and would then be folded into the positive area, requires that the signal is analytically continued into the complex domain. In principle this is possible by using the left and right channels of a modern sound card. However, obtaining an analogue baseband signal of high quality is difficult and in practice one would have to employ a superheterodyne receiver, which mixes the signal in two stages: first to an intermediate frequency (IF) and then to I-Q baseband. This is achieved by splitting the signal into two after the initial mixing down to an IF frequency. The two are then mixed down separately to baseband, the in-phase (I) signal use a standard mixer while the quadrature-phase (Q) signal use a mixer which shift the signal by 90 degrees. The I and Q signals correspond to the real and imaginary parts respectively, of the analytic baseband signal. This requires two oscillators, three mixers, as well as BP and LP filters, for each antenna polarization, which would also make the radio system expensive.

Analogue IF receiver with digital direct conversion

This is a hybrid solution where the RF signal is mixed to an IF signal, using an analogue mixer, and direct converted to a baseband I-Q signal using a digital mixer and filter chain. Since a digital high-frequency (HF) receiver was available that could take an IF signal as input and output a digital baseband I-Q signal, this was the preferred solution.

The idea of the receiver is to take the signal of 1420 MHz and digitize it to a bandwidth of about 100 kHz so analysis can be done by processing the data on a computer. To be able to do this one has to perform a series of actions on the signal to adapt it for the digitizing process. Each action is represented by a component in the receiver chain. Since we have two signals with

different polarization, horizontal (H) and vertical (V), we need to have two paths with identical components. The signal will go through the following events when received by the antenna, see Fig. 4.1 for a block diagram over the system.

1. The incoming signal is received by the dish which reflects it towards the two H and V dipoles in the antenna focus.
2. Directly at each dipole, a Low Noise Amplifier (LNA) is connected which amplifies the signal to compensate for the attenuation in the cable on its way to the receiver.
3. A long cable with low attenuation leads the signal to the receiver, which is situated inside the building.
4. At the ends of one of the cables a Bias Tee is connected, it is used to get power to the first amplifiers.
5. Since we do not want any other signals than the one at 1420 MHz a bandpass filter around that frequency is used to block all other signals and noise.
6. Another amplifier is then used, as we want a strong signal for the mixer to work with.
7. The signal enters the mixer which also has another input for a local oscillator (LO). Mixers work in a way which takes two signals as input and then outputs all the sums and differences of the two signals.
   - The oscillator outputs a stable signal, which is somewhat lower than 1420 MHz in our case.
   - It gets amplified since the mixer needs a quite strong signal to work correctly.
   - Since we have two signals from the antenna we must use the same oscillator signal for both mixers so they are synchronized, for this we use a so called splitter to split the LO signal into two.

   The mixer outputs several signals but we only want the one which is made from the frequency difference between the signals, which will be some tens of MHz depending on how the oscillator is set.
8. As a last step we use a bandpass filter around the desired IF output to get rid of all other frequencies from the mixer.
9. The signal have now been adapted so that it can be sampled and digitized. The final output will be decided by the specifications from the sampling equipment and the frequencies it can handle. In our case we used a digital baseband receiver, which accepted up to 25 MHz input frequency. However it was decided to use a higher input frequency and so called undersampling. The reason for this is economical. since the goal was to keep the cost down it was decided to use bandpass filters that where readily available from a previous project, see the discussion on undersampling in Subsection 4.1.7 for more information on the implications of this particular choice.

4.1 Components

As previously mentioned, it was important to assemble the system at a low cost, it was preferable to use as much of the components that were already at hand to limit the amount of purchases needed. A similar receiver chain had been used for another project for 2.4 GHz, but several of the components could be used also for 1.42 GHz.

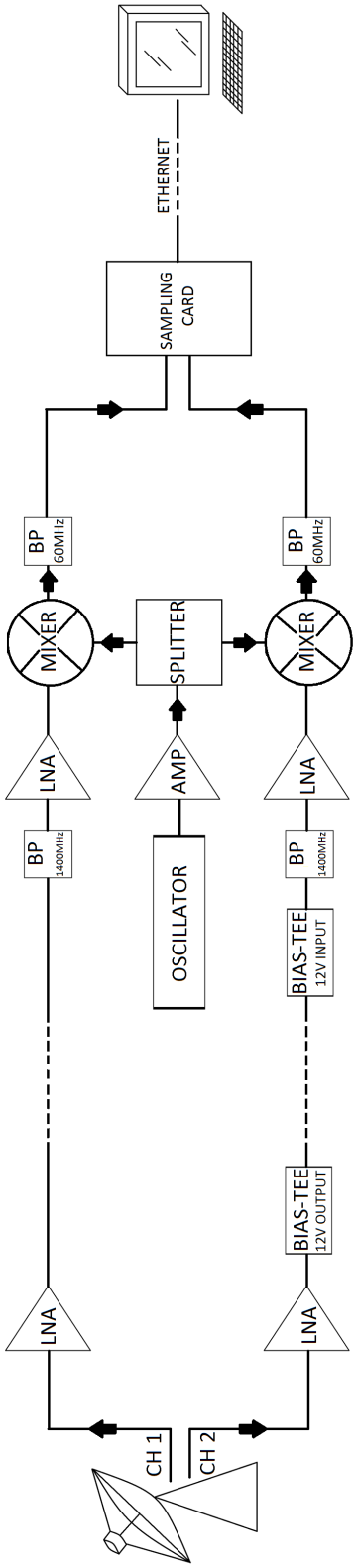

Figure 4.1: Block diagram for the receiver system.

The least expensive way would probably have been to construct all components ourselves, but the amount of time needed for design and assembly would be far too long for it to be feasible within the six months alloted for this project. The quality of the circuits was also a concern as purchasing the items from a reputed manufacturer would in all likelihood lead to a better result than an attempt by us to produce components to the same standard. Much of the work therefore consisted of analyzing data sheets of suitable parts, and based on the findings decide which of the available components could be used and which had to be purchased. See Appendix A for a list of components used for the receiver along with the most important specifications for each.

4.1.1 Amplifiers

For the system, five amplifiers was needed, one pair outdoors by the antenna, one pair inside the receiver box and a single one for amplifying the oscillator signal. The amplifiers in each of the pairs must be of the same model so that the two signals are affected by noise the same manner and have the same frequency characteristics. The same is true for the entire signal chain, the two chains should ideally be mirrors of each other, but there are a few exceptions, which we will address later.

Important when choosing amplifiers are the amount of gain they deliver and also how much noise they produce as a by-product. Since the signals we are searching for are very weak, almost no noise is acceptable. For a good result, we must therefore use low noise amplifiers (LNA). The low noise characteristic comes at the expense of the gain, which is often significantly lower than for amplifiers not made for low noise. The most important part is the first LNA which will set the standard for the rest of the system. Noise from other components in the system will be limited depending on the Gain of the the first LNA [19] [5]. Amplifiers are among the more expensive parts in a system, especially if they have low noise, so it was very fortunate that there were enough amplifiers which matched our specifications already purchased, so that no additional ones were needed.

Amplifier model: LNA 1420 There were two LNA:s available that had been bought especially for this project around the same time as the antennas. They were custom made for the hydrogen line by a company specializing in radio astronomy applications, and they have a very small noise figure. These would be placed outdoors as close as possible to the antennas before the signal cable, they also have weatherproof casing which means no special measures had to be taken to keep them free of moisture.

Amplifier model: ZFL-2500VH+ Inside the receiver box the other amplifier pair is placed after the bandpass filters so that only the frequency we are interested in is amplified. These where also of the low noise type but with not quite as low noise figure as the first pair since these have a much wider bandwidth, almost 1 GHz. The attenuation in the 25 meters long signal cables and the need to get a strong signal for detection, made these amplifiers necessary.

Amplifier model: ZX60-3011+ The task of this amplifier is not as critical and does not have to be of the same low noise characteristics as the others, though it still has to be of good quality. It amplifies the oscillator signal used for mixing down the hydrogen signals. The reason for its use is that we need the same signal for both mixers, meaning we need to split the signal. This results in a drop of about 3 dB in signal strength which we need to compensate for. Also the mixers need a certain power to work properly, and so this amplifier is needed.

4.1.2 Filters

Since we do not want Radio Frequency Interference (RFI) from radio stations or mobile phone signals we need to filter out all unwanted signals. This should be done as early as possible in the chain, preferably just after or even before the first amplification. This would lead to the

filters being placed outdoors which would not be a problem if considering the weather, but the construction of the filters and their required placement makes them quite vulnerable for stress factors and they would quite easily break, so they are instead placed inside the receiver box, filtering after the signal cables. Another filter pair is also needed for the output from the mixers and a few interesting factors have to be considered. The choice of filter frequency is closely tied to the choice of mixer and oscillator, see the section on mixers below for a more detailed explanation.

Filter model: VBFZ-1400+ For the first filter pair we want a bandpass filter around the hydrogen line frequency of 1420 MHz. It turned out to be a little difficult to find good bandpass filters at that specific frequency; most were a little too wide band than one could hope for. Even the filter chosen has a 100 MHz bandwidth ($1350 - 1450$ MHz), however this will not cause large problems as the frequencies in that range are not used very much by other applications than radio astronomy [20]. The problems that might occur from interference with other services will be filtered out by additional filters later in the chain.

Filter model: SBP-60+ This bandpass filter is used to filter out the additional frequencies from the mixer which we do not want. The decision to use a filter at 60 MHz was mostly based on that we already possessed those filters and would then save money, but also the range of products was pretty scarce for frequencies below 100 MHz. The only other viable choice was basically a filter at 10.7 MHz, this might sound like a good choice since our ADC:s operate at 50 Msamp/s. However it turns out to give some additional problems, see the section on mixers.

4.1.3 Mixers

The mixer is one of the more important parts of the receiver as its task is to transform our high frequency signal to a lower frequency which we can sample and analyze. The output frequencies from the mixer consists of all of the different combinations of the hydrogen signal and the local oscillator signal. One need to choose an oscillator frequency which gives an output frequency appropriate for our sampling circuit.

$$\begin{aligned} \text{Output frequency} &= \text{Signal - Oscillator [21]} && (4.1)\\ 60MHz &= 1420MHz - 1360MHz && (4.2)\\ -10MHz &= 1350MHz - 1360MHz && (4.3)\\ 90MHz &= 1450MHz - 1360MHz && (4.4)\\ 55MHz &= 1415MHz - 1360MHz && (4.5)\\ 70MHz &= 1430MHz - 1360MHz && (4.6) \end{aligned}$$

The output frequencies, from various inputs, we get when we choose 60 MHz as the desired output can be seen in the Eqs. (4.1) to (4.69). For a local oscillator signal of 1360 MHz, the first bandpass filter lets through frequencies between 1350 MHz and 1450 MHz, those frequencies result in 90 MHz and $(-)$10 MHz. The minus sign is only for mathematical purposes as in reality this means that it is folded back into 10 MHz. The filter used for 60 MHz lets frequencies between 55 MHz and 70 MHz through, which if we calculate backwards means that only signals between 1415 MHz and 1430 MHz will be detected. Since frequencies at that interval is almost exclusively used for radio astronomy [20], we should have no problems with signals from other sources. We can conclude that the local oscillator (LO) signal needed is 1360 MHz.

$$\begin{aligned} \text{Output frequency} &= \text{Signal - Oscillator} \\ 10MHz &= 1420MHz - 1410MHz \quad (4.7) \\ -60MHz &= 1350MHz - 1410MHz \quad (4.8) \\ 40MHz &= 1450MHz - 1410MHz \quad (4.9) \\ -10MHz &= 1400MHz - 1410MHz \quad (4.10) \end{aligned}$$

If we look at a filter at about 10 MHz, which would be more appropriate since the sampling card only supports frequencies up to 25 MHz, the following relations result; see Eqs. (4.7) to (4.10). We see that it seams to work for the desired frequency, but in the last equation we can note that the result for an unwanted signal at 1400 MHz would be folded and result in a signal mixed down to the same frequency as the hydrogen line. This might lead us to believe that a signal at 1400 MHz is in fact a signal at 1420 MHz, which is totally unacceptable and would lead to false data. One could then conclude that mixing down to 60 MHz is the best choice. There is however a method of double mixing which could improve the end result of our signal.

Two stage mixing

In our case a two stage mixer setup would mean using both of the considered BP filters at 60 MHz and 10.7 MHz. You simply take the signal you have filtered out at 60 MHz and use it as an input for another mixer which gives an output of 10.7 MHz. This will result in even more unwanted signals getting removed, clearly the best option. However as mentioned earlier one must consider the cost of those additional parts. One would need two mixers and filters, and another oscillator with amplifier and splitter, significantly increasing the cost of the system, and for quite little gain compared to the interference from other signals when using the 60 MHz setup only.

Mixer model: ZFM-15+ As with many other components a suitable candidate was already at hand, and it was an easy task of confirming its usability. It is able to mix frequencies up to 3 GHz, see A.3, more than enough for our project. One could perhaps even have wished for the mixer to not have such a broad input range as that would have dampened out some of our unwanted signals further.

4.1.4 Oscillators

When the preferred oscillator frequency has been determined to 1360 MHz one must find a matching oscillator for that frequency. It turns out this was not at all as trivial as one would first think. We had to try three different approaches before finding one that worked, and the result was not quite ideal.

*** First attempt: VCO**

The first suitable product we tried was a Voltage Controlled Oscillator (VCO). VCO:s are set using a certain input voltage which then will generate an output frequency based on that voltage.

Oscillator model: ZX95-1420+ In our case around 14 V gave us the frequency we wanted. It was quite early established that the voltage input had to be very stable or else the frequency would fluctuate greatly. The first measurements showed that the oscillator seamed stable enough, but as the receiver started to get finished and real measurements could be done, a spectrum analysis showed the signal which should have been a sharp spike looked more like a Gaussian curve. This was not at all acceptable and the oscillator had to be stabilized.

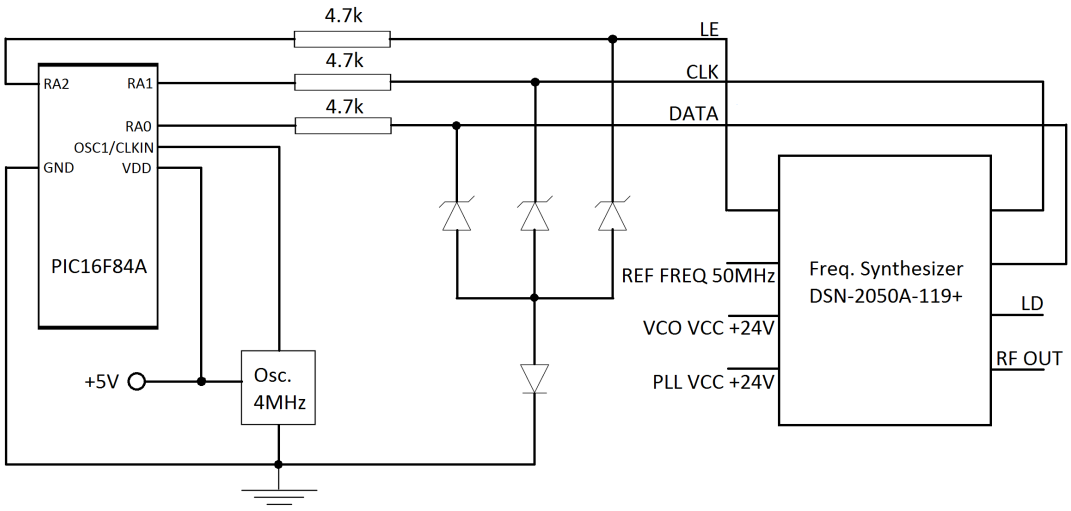

Figure 4.2: Circuit diagram over the frequency synthesizer and the microcontroller. The zener diodes are specified to 3V reverse voltage to limit the input to the synthesizer.

*** Second attempt: Phased Locked Loop**

Stabilizing an oscillator is made using a so called Phase Locked Loop or PLL. It feeds back part of the output frequency and compares its phase to a stable reference frequency, often a crystal-based oscillator [22]. It then changes the input voltage to match the change in frequency and after a few iterations the output frequency is stable. Constructing this setup yourself is quite time consuming, and as this was in the very late stages of the project a new integrated VCO and PLL circuit, called a frequency synthesizer, was purchased, removing the old VCO. The frequency synthesizer also contained a programmable circuit which had to be set every time at power on. For this initialization we had to use some kind of microcontroller, the one deemed the easiest and quickest to get working was a PIC microcontroller.

PIC microcontroller - PIC16F84A

To program the PIC one could use either C or the Assembly language. C would perhaps have been the obvious choice but compilers for C code is not included with the controller and must be purchased separately, while Assembly compilers can be downloaded from the manufacturers homepage [23]. So the choice was made to use Assembly and a few days time was devoted to get it working properly, with the added work of constructing a circuit board for mounting the controller. The Assembly code for the microcontroller can be seen in Appendix B.

Frequency Synthesizer model: DSN-2050A-119+ The synthesizer has the ability to generate and stabilize frequencies from 1100 MHz to 2100 MHz. A drawback is its quite weak output of only 0.5 dBm which makes the subsequent amplifier even more important. We actually purchased two of these synthesizers, and unfortunately managed to burn the first one by accidentally increasing the input voltage above the maximum limit. The problem here was that we never got the synthesizer to work properly (even with the correct voltage), it simply would not accept the programming. Other frequencies were tried, default programming sequences used, but nothing seamed to help. The choice was made to abandon this part of the project because of time constraints, the project had already been extended by one month because of the oscillator and at least one more would be spent finding a new solution.

Table 4.1: Signal strength loss for different cables and frequencies. [24] [25]

Frequency	RG58 [dB/100m]	RG214 [dB/100m]	Aircom Plus [dB/100m]	RG402U [dB/100m]
1000 MHz	53.7	28.6	13.4	37
2000 MHz	83.7	41.9	20.1	-
3000 MHz	107.5	51.7	25.9	-
5000 MHz	-	-	35.9	91
1420 MHz	16.25 dB/25m	8.75 dB/25m	3.75 dB/25m	10 dB/25m

*** Final attempt: External signal generator**

The resulting solution was to use an external signal generator connected to the receiver. A case connector was mounted on the front of the receiver box and connected to the LO-amplifier via a cable inside the case. The original VCO was also left inside the case together with a piece of cable so that one could use this in case there was no external generator available, or just for system testing.

Power Splitter model: ZAPD-1750-S+ If we want to be able to do accurate comparisons between the individual phases, the two signals we get from the antennas must use the same oscillator signal. For this we will need a splitter to simply split the LO-signal into two. The important part here is to know that the mixers requires an input of at least 10 dBm, and some splitters can not handle that much power. The model chosen can handle up to 10 W (40 dBm) which gives a good margin.

4.1.5 Signal cables

The Å8-dish is placed outdoors on the edge of the roof of house 8, the plan was to have the signal cable run from the dish through a nearby wall, and down to the receiver placed in a cabinet on the floor below. According to rough measurements about 25 m cable was needed for this, including a few meters extra in case of difficult cable management at installation. We considered three different cables for use in this project, the regular RG58 cable, the slightly better RG214 and the high quality Aircom Plus® cable. The signals they are to carry are of very high frequency, and the higher frequency one uses the more impact the attenuating properties of the cable have. RG58 works for short distances even at quite high frequencies, but if the cable is very long the signal will eventually become substantially attenuated. It does however have the advantage of being light and flexible. The less attenuation you want the more shielding must be used to prevent leakage through the shield. Also other dielectric material as well as a thicker inner conductor are required. This makes the cable more rigid and a bit harder to handle. In addition, cable routing becomes more difficult. A comparison of the attenuation of the different cables is shown in Table 4.1.

Signal cable model: Aircom Plus We clearly see that the Aircom Plus is the superior cable for our application. There is of course the previously mentioned problem of cable routing when dealing with stiff cables like this one, but after surveying the intended area no eventual problems was found and installation should go smoothly. More information on the Aircom Plus cable in Appendix A.7.

Signal cable model: RG402 It is imperative that the chains of the different channels are of the same length to keep the signals in phase. To make sure this is accomplished we used a semi-rigid coaxial cable, basically an inner conductor surrounded by dielectrica and a solid copper shield, which can be bent and will then keep its shape. The length is determined by how much the channels differ, which was about 3 cm, this is mostly from the use of two Bias Tees in one of the channels.

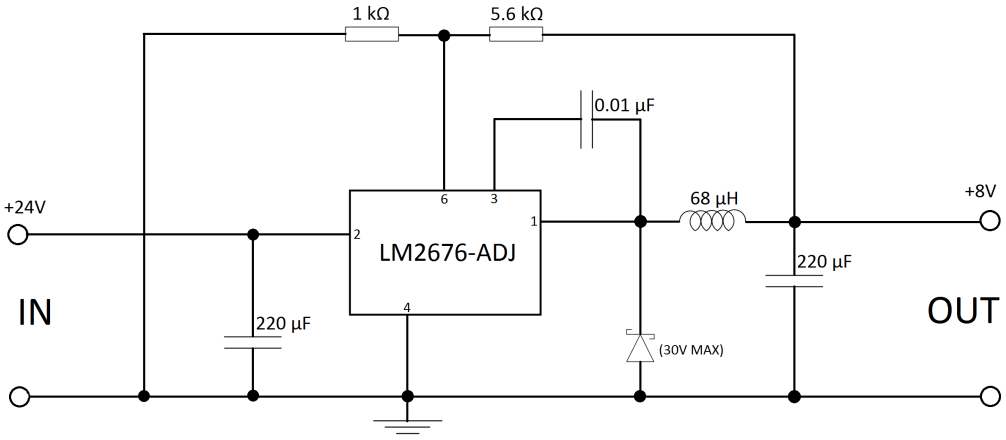

Figure 4.3: Circuit diagram of the step-down circuit. It reduces 24 V to 8 V and limits the heat loss very efficiently since the energy is stored in the capacitor and inductor, instead of heating a resistor.

4.1.6 Power supply

Amplifiers, sampling circuits and oscillators are power hungry parts and require specific voltages to work. A power supply was needed, integrated into the receiver box, which could deliver the necessary voltages and currents for the components. At first the power supply needed to provide up to about 16 V as this was about what the original oscillator needed, and so we purchased a switched power supply with the closest matching voltage which was able to provide 24 V. It could also deliver 1.3 A which at first was thought to be enough and with quite good margin, but later it was realized that the ADC on the sampling card required a lot more current than anticipated. This did not become obvious until the system was ready for the first tests and the current budget was calculated again. The result was that the power supply would be able to almost exactly supply the needed current. A slightly higher startup current however managed to trigger the supply's current limiter, which was not reset until after a power down. A capacitor had to be put in to limit the startup current, after which it all worked fine.

Regulators and step down

The voltage needed from the supply was 12 V for all the five amplifiers, 5 V for the sampling card, the microcontroller and part of the synthesizer, the other part of the synthesizer required 24 V. To achieve these different voltages regulators was used for 12 V and 5 V, before the 5 V regulator a step down from 24 V to 8 V was used to limit the heat loss somewhat. A circuit diagram of the step down is shown in Fig. 4.3.

Because of the heat loss from the power supply a fan was mounted in the top of the case to exhaust the hot air so that the temperature in the receiver was kept low. All the used components are sensitive to heat which is why the excess heat should be expelled. Close to room temperature is ideal but the receiver was measured to just above 30°C, which is acceptable.

Bias Tee

A problem with mounting amplifiers directly on the dipoles are that they must be provided with power. This would normally result in separate cables being put up, and they need to be fastened properly so they do not stretch and break while the dish is moving. To solve this problem one can use two so called Bias Tees. What it does is that it inputs a bias, of in our case 12 V, on the signal cable in one end of the cable, which can then be extracted in the other end by the other device.

Bias Tee model: ZNBT-60-1W+ This device can handle frequencies up to 6 GHz and RF signals of 30 dBm. Another advantage is that it has N-connectors which makes it easy to connect to the Aircom Plus cable as it also utilizes N-connectors. One item of this type was already purchased and only another one was needed. They are quite expensive but the benefits of using a bias tee is well worth the price.

4.1.7 Receiver Circuit Board

When all analog signal processing have been completed it is time to convert the signal to digital form for storage and analysis. This is implemented with a receiver card designed by Walter Puccio at the Institute for Space Physics (IRFU), originally for research in the 100 MHz area. The card has been made in about a dozen copies and is able to receive up to three channels. However the card we used had only been equipped with two 14 bit Analog-to-Digital converters (ADC), resulting in only two channels and making it perfect for our project.

Each channel has a bandwidth of 74 kHz and a sampling frequency of 50 MHz, giving us a maximum input frequency of 25 MHz when the Nyquist sampling theorem is taken into account. The bandpass filters at 60 MHz will limit the bandwidth to 15 MHz, from $(5-20)$ MHz. All other frequencies will be blocked. The card also utilizes a 10 Mbit Ethernet connection for transmitting data to the user.

Undersampling

For sampling we utilize a common technique which might seem strange at first. We will use a sampling frequency of 50 MHz which is less than the maximum frequency of 60 MHz that we receive. This is far from the recommended 120 MHz for sampling such a signal, and this will cause us to have folding frequencies. As noted in the sections on filters and mixers, Sec. 4.1.2 and 4.1.3, we did not want any folding in those parts because it may cause undesired signals to enter our sampling bandwidth. However, here we will use this phenomena to our advantage.

The maximum input frequency of the circuit board is 25 MHz, so frequencies higher than 25 MHz, *i.e.* $(25-50)$ MHz, will be folded down to the $(0-25)$ MHz band and also mirrored, meaning a frequency of 26 MHz would appear to be at 24 MHz. But frequencies in the range $(50-75)$ MHz, will be folded twice and not mirrored [26]. For our frequency at 60 MHz this means that it would appear to be at 10 MHz, and so we can conclude that this frequency is in fact our hydrogen line. One should note that our ability to make these operations is based on filters blocking the unwanted frequencies beforehand, otherwise we could not assume that this signal would be the hydrogen line. Using this kind of undersampling enables us to get the same results using less equipment, e.g. we save one down-mixing step and filtering as mentioned before.

4.2 Evaluating The System

An important property of the system is to know how weak a signal one will be able to detect. In radio astronomy it is often called sensitivity (S_{i}), or otherwise known as minimum input signal power. In order to calculate the sensitivity we calculate the noise figure of the system with the Friis formula for noise, and the expressions below from Pozar [19], where:

$$\begin{aligned} F_{\mathrm{sys}} &= F_{\mathrm{LNA1}} + \frac{F_{\mathrm{cable}}-1}{G_{\mathrm{LNA1}}} + \frac{F_{\mathrm{BP1420}}-1}{G_{\mathrm{LNA1}}\cdot G_{\mathrm{cable}}} + \frac{F_{\mathrm{LNA2}}-1}{G_{\mathrm{LNA1}}\cdot G_{\mathrm{cable}}\cdot G_{\mathrm{BP1420}}} \\ &+ \frac{F_{\mathrm{MIXER}}-1}{G_{\mathrm{LNA1}}\cdot G_{\mathrm{cable}}\cdot G_{\mathrm{BP1420}}\cdot G_{\mathrm{LNA2}}} + \frac{F_{\mathrm{BP60}}-1}{G_{\mathrm{LNA1}}\cdot G_{\mathrm{cable}}\cdot G_{\mathrm{BP1420}}\cdot G_{\mathrm{LNA2}}\cdot G_{\mathrm{MIXER}}} \\ &= 0.46 \text{ dB} \end{aligned} \tag{4.11}$$

F_{sys}	The total Noise Figure of the system.
$B = 60$ MHz	The Bandwidth after the mixer.
$G_{\text{tot}} = 34.12$ dB	The total gain the system after loss from cables and other components.
N_{o}	The output noise power.
T_{sys}	The equivalent noise temperature of the system.
$T_{\text{o}} = 290$ K	Actual system temperature, specified as room temperature.
$SNR = 3$ dB	The minimum required Signal-to-Noise ratio.
$T_{\text{ant}} = 100$ K	The noise temperature of the sky at 1420 MHz [27].
$k = 1.38 \cdot 10^{-23}$ J/K	Boltzmann constant.

$$T_{\text{sys}} = (F_{\text{sys}} - 1)T_0 = (1.112 - 1)290 = 32.48 \text{ K} \tag{4.12}$$

$$N_{\text{o}} = k(T_{\text{ant}} + T_{\text{sys}})G_{\text{tot}}B = -65.5 \text{ dBm} \tag{4.13}$$

$$S_{\text{i}} = SNR\frac{N_{\text{o}}}{G_{\text{tot}}} = -96.6 \text{ dBm} \tag{4.14}$$

The resulting sensitivity from the components is quite high, which largely depends on the good LNA put at the dipoles, and should be enough to detect the hydrogen line. This is of course an estimation and the actual sensitivity might be slightly smaller.

Chapter 5

Results

5.1 The Ångström Small Radio Telescope

After long months the result was finally a steerable telescope able to receive and sample the hydrogen line. At the end of the project the parts that were completed were:

- The implementation of a Graphical User Interface for steering the dish, with improvements made to the existing controller-box firmware.
- The construction of a complete receiver system including digital sampling of the signal.

Both the receiver and the controller utilizes an Ethernet connection, and have been given local IP addresses and are connected to a router, making them accessible from outside.

5.1.1 The Dish

After using the dish for testing the controller program during the development phase it was notable how well preserved the dish was, almost no rust, or other effects from harsh weather. The perforation makes the dish light and easy to work with, as well as limit the effect from strong winds, see Fig. 5.1 and 5.2. The tripod made for the antenna is however not at all light, being made from galvanized steel. This caused some small problems as the tripod and motors had to be brought down from the roof, and placed inside during the development of the controller GUI. There is unfortunately no elevator up to the roof of House 8, instead there is a small stair, with a chain block for bringing up heavy equipment. It took some effort to get the pieces down, and later up, but without incidents. Cables were fastened with cable ties and secured in a way so that they would not be stretched or damaged as the dish turned.

5.1.2 The Motors

The motors were in very good condition despite not being used for a few years, only some basic maintenance with grease on the gears was needed. A slight improvement have been the calibration so that the motors move 180 degrees in each direction. It should be noted that since the motors were not originally intended for use in both elevation and azimuth, the construction implies that the center pivot point will change somewhat when moving the dish in azimuth, it is however largely negligible.

5.1.3 The Controller System

Much of the work made on the controller box itself was basically debugging in conjunction with the development of new functions in the controller GUI. Parallel to the calibration of the motors some values had to be edited in the controller box firmware as well, to get them synchronized.

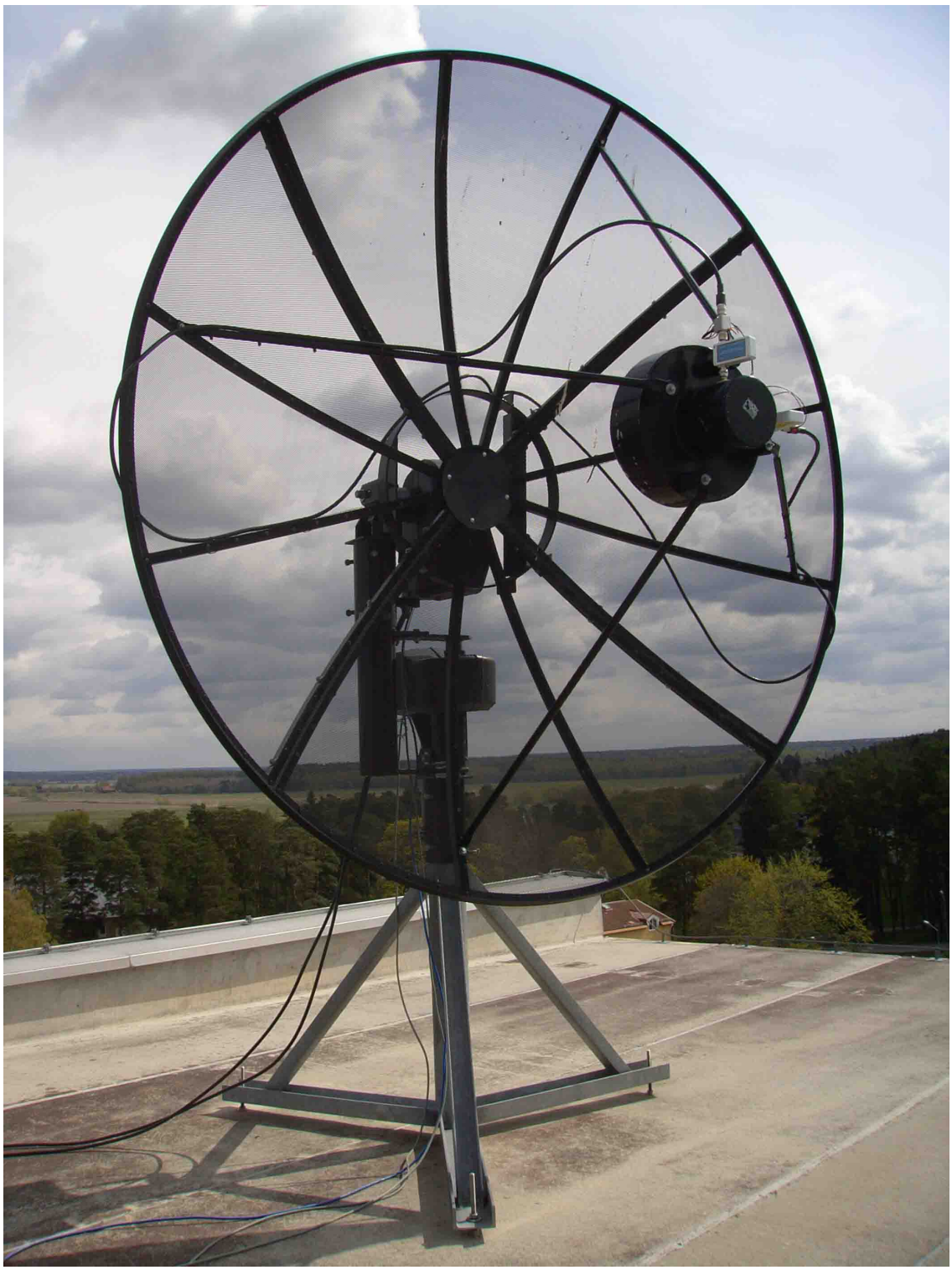

Figure 5.1: The dish in its final stage with all cables and amplifiers connected, pointing at the reset point, Azimuth 270 degrees and elevation 0 degrees.

Figure 5.2: The motors, with the bottom one mounted perpendicular to the top motor.

Also some of the cables from the controller box had to be extended to reach all the way out to the dish, see the controller box in Fig. 5.3.

Figure 5.3: The controller-box.

5.1.4 The Receiver System

The receiver can be split into the outer and inner parts. The outer parts are the amplifiers mounted on the dipoles, see Fig. 5.4. The implementation of these are fairly straight forward and the mounting was easy thanks to the use of the bias-tee. Note that one of the signal cables have a red colored connector to distinguish them.

As for the inner part, it is comprised by the case of the receiver and its components, see Fig. 5.5. The case has; input connectors for the two channels and the external oscillator for the mixer, and an output Ethernet connector, on the front. It also has a power input for 240 V in the back with a power-ON LED on the front. The case itself is perforated to allow for airflow, with the fan mounted in the top. Also note the red colored connector on the same cable as can be seen outdoors, see Fig. 5.4.

The inside of the receiver is even more interesting, all the components are neatly mounted on a board with a thin layer of copper to give them a common ground, see Fig. 5.6. Cable management has not been a priority since it would take valuable time away from the actual project. Note the little semicircle cable of channel two, that is the effect of the two channels being of different length because of the Bias Tees. To keep the same phase for the channels it has to be somewhat longer than what would otherwise be considered practical. In the bottom right corner you can also see a potentiometer, this is used for setting the correct voltage for the internal oscillator, adjusting it would result in a different output frequency.

In Fig. 5.7 the step-down circuit can be seen, working as intended and resulting in a much lower temperature in the 5 V regulator. The other regulator however, gets quite hot when in use, hence the necessity of a fan and perforated case.

Figure 5.4: 1. LNA and 2. Bias-Tee, mounted on the dipoles.

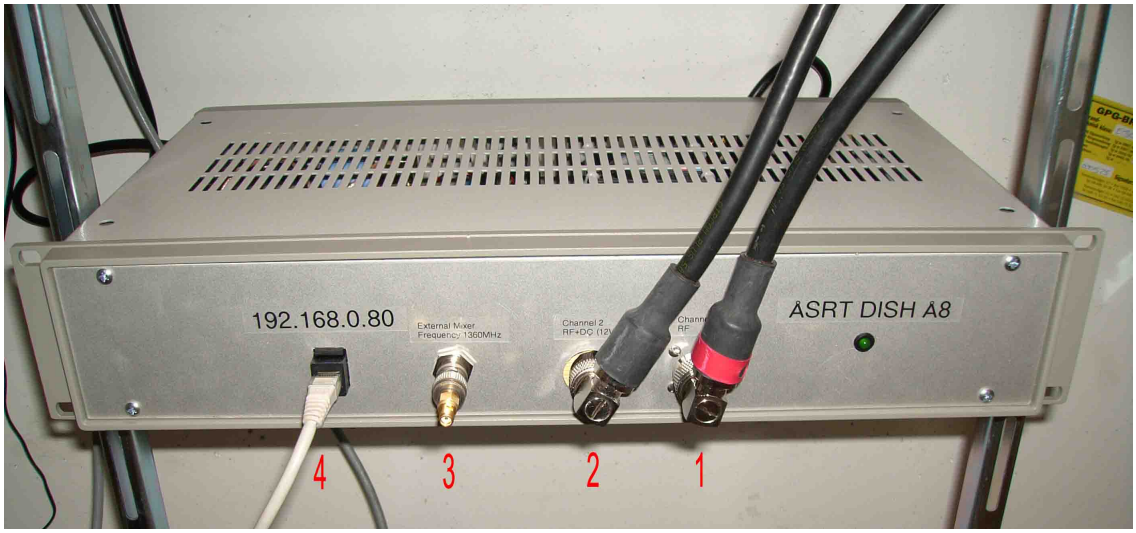

Figure 5.5: The front of the receiver case. 1. Channel 1, 2. Channel 2, 3. External Oscillator input, 4. Ethernet output.

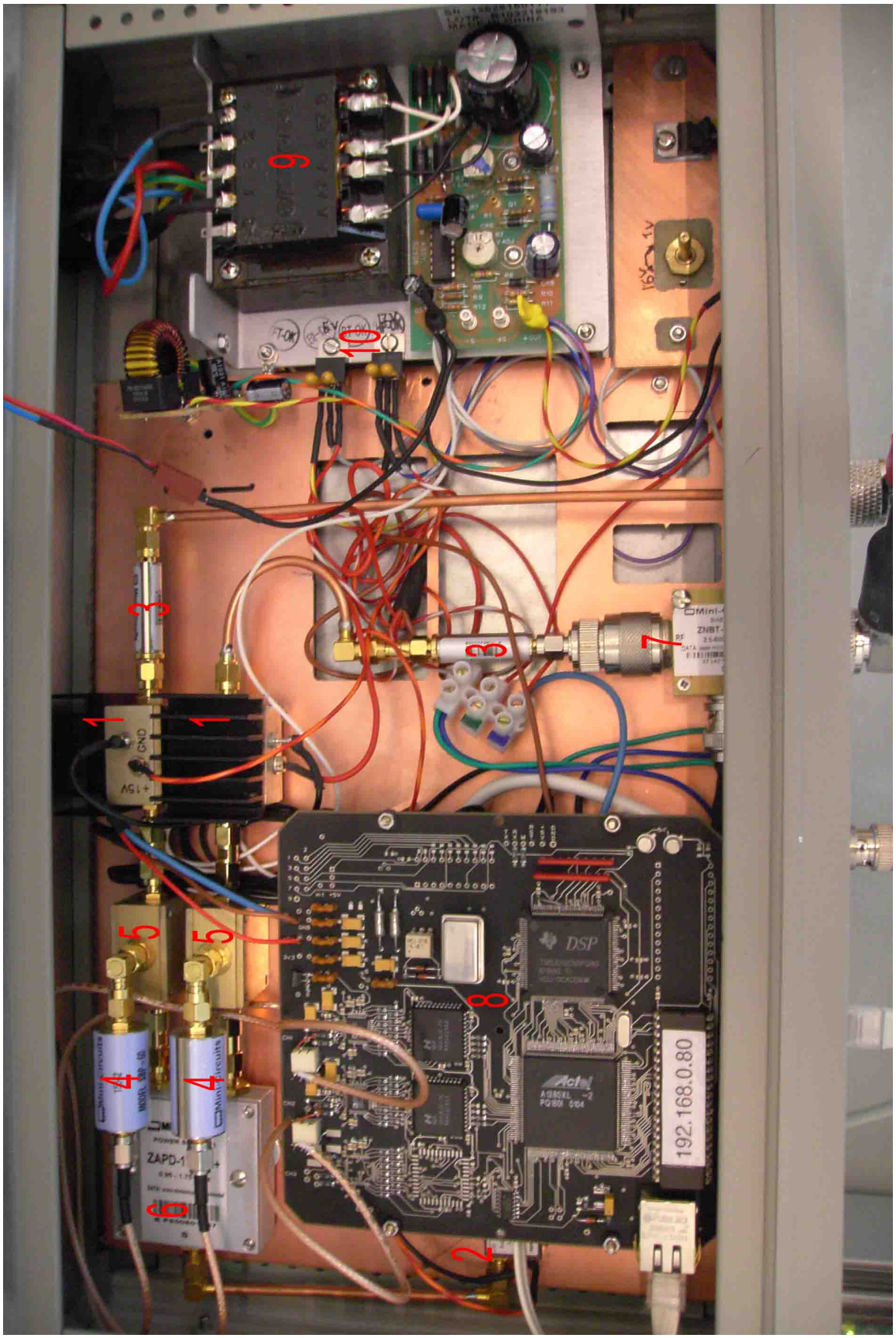

Figure 5.6: The inside of the receiver case. 1. LNA amplifier, 2. LO Amplifier, 3. 1400 MHz filters, 4. 60 MHz filters, 5. Mixers, 6. Splitter, 7. Bias-Tee, 8. Sampling card, 9. Power Supply, 10. Regulators. The internal oscillator is placed underneath the sampling card, and is not visible.

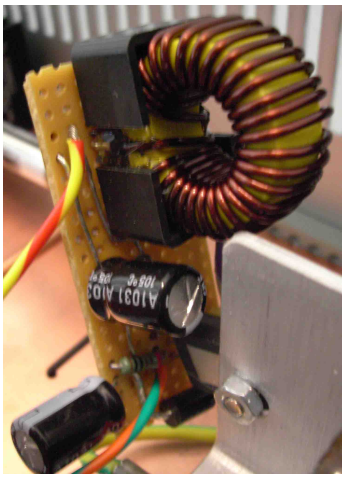

Figure 5.7: A close up of the step down circuit.

5.2 The signal from outer space.

The concluding test was to analyze the sampled signal and see how strong the signal was. We used a signal generator as the local oscillator and set it to the prescribed 1360 MHz. We also used the Sensor GUI [28] program which is a light analysis program made for the same sampling card type as ours. It can among other information, show a spectrum of the received signal, see Fig. 5.8. As one can see we have two signals, red and green, the green is a little weaker than the other. After exchanging cables and components back and forth it was concluded that the weaker signal was not an effect of the equipment, simply that one polarization was weaker than the other.

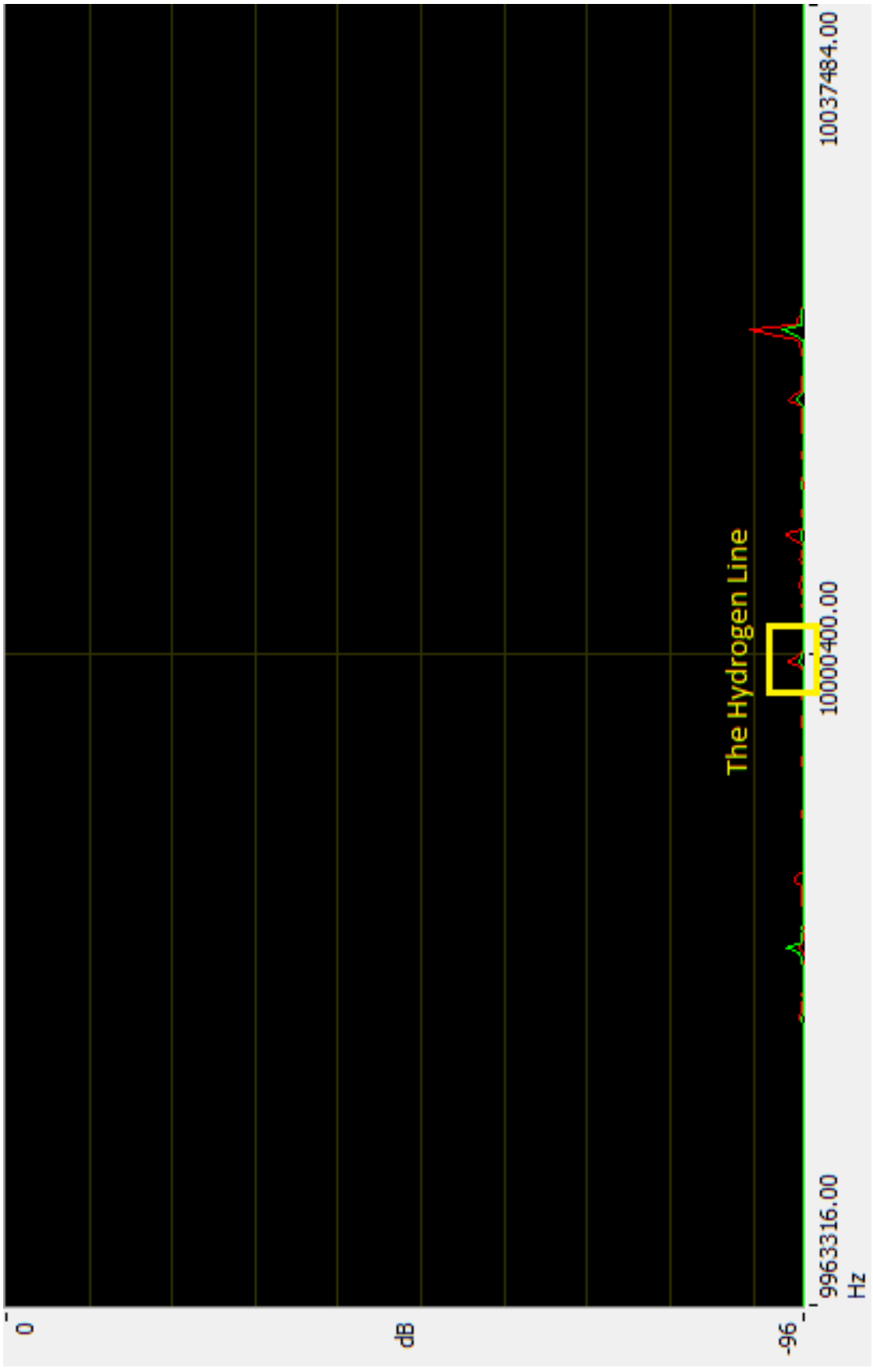

Figure 5.8: The spectrum of the 21 cm line received by the ÅSRT. The signal is as expected very weak, red is from the horizontal dipole and green from the vertical. The data is taken in the direction of Az 160°, *El* 60° *at* 13 : 06 *on May 11, 2011.*

Chapter 6

Conclusions

The project has been filled with interesting challenges, and has touched upon more or less all kinds of electronics that one could come in contact with: analog electronics for power supply, regulators and step-down circuits; digital electronics in programming the PIC controller and the synthesizer; high frequency electronics for the entire system; and graphical interface and hardware-near programming for the controller. As the project progressed it become evident that not all of the goals set up in the beginning would be accomplished. Some parts where left to be implemented at a later stage, such as: combining the controller software with a program to store data onto a hard drive, and making drift-scans. This was a direct result of the complexity of the project, each part took more time than was originally estimated, and the unfortunate problems with the local oscillator delayed it even more. All in all, the project extended 3 months over time, and it would probably have taken another few months if the skipped parts would have been implemented in this project.

6.1 Possible Improvements

The telescope is not a perfect machine and there are of course improvements to be made. While these have been considered during the project they have not been implemented, a few because of time constraints while others require more or less a complete reconstruction of parts of the system.

- Calibration - One thing that should be considered is that the telescope should have a thermal diode mounted on the dish to be used for calibrating the system.
- Hall elements - A big concern is the fact that the telescope only can move in one degree increments, this is because there is a mechanical counter keeping track of how long the motor have moved. There are about twelve teeth counts on that gear for each degree of the dish, the fact that it is not an exactly known number of teeth makes it impossible to have a more exact movement of the dish. There is also the risk of the counter jumping one tooth for some reason since it is mechanical. To improve on this one could use another form of counting mechanism, *e.g.* Hall elements, small magnets distributed evenly along the gear and a detector which register each time a magnet crosses in front of it, the more magnets the higher the accuracy of movement. This must be made in conjunction with rewriting parts of the controller-box firmware to enable it to receive commands in decimal form.
- Instant sampling - As mentioned before there are possibilities to sample frequencies as high as the hydrogen line without the help of analogue mixing stages. This would mean large parts of the receiver system could be exchanged for a card with those capabilities, but they are as stated very expensive.

- Step-Down to 12 V - The step-down circuit for the 5 V regulator was very successful and resulted in much less heating. The same could or perhaps should be done for the 12 V regulator as well. This would decrease the heat in the case and make other components work better in the lowered temperature.
- The oscillator - An unfinished part is of course the local oscillator for the mixer. The current implementation works for now, but should be changed for a more permanent solution. It is probably recommended to scrap the previous attempts and start from scratch with a new one. Another possibility is to run either an external or the internal source. This is possible as is, but only after tedious removal of components to gain access to the oscillator. A solution would be to connect a splitter before the LO amplifier and connect both sources to it, and mount a power switch on the front to turn the internal oscillator on or off.
- The GUI - After the GUI was complete there has become evident that a few functions would make the use of the program a little easier. This would first be an option to choose which of the scanning patterns should be used, by a simple radio button, and secondly an option to turn off the required scanning and tracking time limiter. There are also extensions possible for the program which would increase the versatility of the program. Mostly it is in the ability to track different objects, planets, and satellites. NORAD has for example information on movement of most satellites in a so called two-line element set (TLE) [29], a text format for easy use. Planets have their orbits well documented and algorithms for tracking their movement should not be hard to find.

Appendix A

List of Components

This list is comprised of the RF components used for this project, this includes cables and adapters. Each product has its most important properties listed, as specified by the manufacturer.

A.1 RF Circuits

Table A.1: Low Noise Amplifiers

Manufacturer	Model	Amount	Frequency
Radio Astronomy Supplies [30]	LNA 1420	2	1420 MHz
Mini-Circuits [31]	ZFL-2500VH+	2	(10-2500) MHz
Mini-Circuits	ZX60-3011+	1	(400-3000) MHz

Gain	Noise Figure	Bias Voltage	Max. Current
28 dB	0.35 dB	+(12-15) V	100 mA
20 dB	5.5 dB	+(12-15) V	300 mA
13.5 dB	1.5 dB	+12 V	120 mA

Table A.2: Filters

Manufacturer	Model	Amount	Frequency	Insertion Loss
Mini-Circuits	SBP-60+	2	(55-67) MHz	1.14 dB
Mini-Circuits	VBFZ-1400+	2	(1350-1450) MHz	1.97 dB

Table A.3: Mixer

Manufacturer	Model	Amount	Frequency	Conversion Loss
Mini-Circuits	ZFM-15+	2	(10-3000) MHz	7 dB

Table A.4: Oscillator & Synthesizer

Manufacturer	Model	Amount	Frequency
Mini-Circuits	ZX95-1420+	1	(1230-1420) MHz
Mini-Circuits	DSN-250A-119+	1	(1130-2100) MHz

Power Output	Bias Voltage	Max. Current	Tuning Voltage
+6 dBm	5 V	35 mA	+(0-16) V
+0.5 dBm	VCO +5 V PLL +24 V	VCO 31 mA PLL 27 mA	-

Table A.5: Splitter

Manufacturer	Model	Amount	Frequency	Insertion Loss	Max. Power Input
Mini-Circuits	ZAPD-1750-S+	1	(950-1750) MHz	3.2 dB	10 W

Table A.6: Bias-Tee

Manufacturer	Model	Amount	Frequency
Mini-Circuits	ZNBT-60-1W++	2	(2.5-6000) MHz

Conversion Loss	Max. Voltage Input	Max. Current Input
0.6 dB	30 V	500 mA

A.2 Cables & Adapters

Table A.7: Cable

Manufacturer	Model	Amount	Frequency
SSB-Electronic [24]	Aircom Plus	25 m	(0-10) GHz
Hangzhou Hongsen Cable Co. [25]	RG402U	0.5 m	(0-20) GHz

Impedance	Attenuation at 1500 MHz	Shielding	Dielectric
50 Ω	17 dB/100m	Copper foil and braid	Semi airspaced
50 Ω	0.5 dB/1m	Seamless Copper Tube	PTFE

Table A.8: Adapters

Model	Connector	Amount
Straight SMA-SMA	Plug-Plug	4
Right Angled SMA-SMA	Plug-Jack	2
RG402-(Case N)	Cable-Jack	1
RG58-(Case BNC)	Cable-Jack	1
Straight N-SMA	Jack-Plug	1
Straight SMA-SMA	Jack-Jack	1

Appendix B

Source Code - PIC microcontroller

B.1 PLL_prog.asm

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;       PLL setup program for local oscillator in the Ångström Small Radio Telescope Receiver
;
;       Author: Henrik Lindén            Last changed: 2011-04-07
;
;       Send data sequences to the PLL to set frequency to lock on.
;       Data recepition is enabled at power on in PLL.
;       Uses "Initialization Latch Method" for ADF4113 chip programming
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

INCLUDE "p16f84a.inc"

;*****Register Shortcut*****

STATUS  equ 03h                         ; Address of STATUS register
TRISA   equ 85h                         ;Address of TRISA register for Port A
PORTA   equ 05h                         ;Address of Port A



;*****Define Bit Shortcut*****
#DEFINE Data PORTA,0            ;Set "Data" as shortcut to the Bit controlling RA0
#DEFINE Clock PORTA,1
#DEFINE LatchEnabled PORTA,2
#DEFINE Toggle PORTA,3

PAGE
__CONFIG _CP_OFF & _XT_OSC & _PWRTE_ON  & _WDT_OFF

;*****Macros*****
 ClockStrobe MACRO                      ;Toggle Clock to detect "Data" Bit
 call Dlay5                                     ;delay to give zenerdiod time to react if voltage is high
        bsf     Clock
 call Dlay5
        bcf     Clock
 ENDM

 LatchStrobe MACRO                      ;Toggle LatchEanbled to load 24bit sequence into memory on oscillator
 call Dlay5
        bsf     LatchEnabled
 call Dlay5
        bcf     LatchEnabled
ENDM
```

```
44
45  ;*****Allocate Data*****
46  CBLOCK 0Ch
47          bitNr
48          REG
49          Dlay
50  ENDC
51
52
53  ;*****  *****
54  ORG 0
55
56
57  ;*****Setup Ports*****
58  bsf STATUS,5                        ;Switch to Bank 1
59  movlw b'00000'                      ;Set RA0, RA1 and RA2 to outputmode (and
        others also to outputmode)
60  movwf TRISA                                 ;Move value from w to TRISA
        register to set outputmode
61  bcf STATUS,5                        ;Switch to Bank 0
62
63
64  ;*****MAIN Program*****
65  START:
66                  bcf LatchEnabled
67                  bcf Clock
68                  bcf Data
69
70                  movlw b'00000011'       ;Register Sequence L, LSB 11
71                  call Setbyte
72
73                  movlw b'10000000'
74                  call Setbyte
75
76                  movlw b'10010011'
77                  call Setbyte
78
79                  LatchStrobe                 ;Load sequence
80
81          ;       movlw b'00000011'       ;Register Sequence F, LSB 10
82          ;       call Setbyte
83
84          ;       movlw b'10000000'
85          ;       call Setbyte
86
87          ;       movlw b'10010010'
88          ;       call Setbyte
89
90          ;       LatchStrobe                 ;Load sequence
91
92                  movlw b'00000000'       ;Register Sequence R, LSB 00
93                  call Setbyte
94
95                  movlw b'00000000'
96                  call Setbyte
97
98                  movlw b'00010100'
99                  call Setbyte
100
101                 LatchStrobe                 ;Load sequence
102
103                 movlw b'00000000'       ;Register Sequence N, LSB 01
104                 call Setbyte
105
106                 movlw b'00010001'
107                 call Setbyte
108
```

```
109                 movlw b'00000001'
110                 call Setbyte
111
112                 LatchStrobe                         ;Load sequence
113
114                 bcf LatchEnabled
115                 bcf Clock
116                 bcf Data
117
118 Endlessloop:                                ;Loop Forever when Done
119                 ;clrwdt
120                 ;bsf    Toggle
121                 ;bcf Toggle
122                 goto   Endlessloop
123
124
125 ;*****Subroutines*****
126 Setbyte:
127                 clrwdt
128                 movwf REG
129                 movlw 8                             ;Number of bits in each register to
                        send, loop that many times
130                 movwf bitNr
131 Sendreg:
132                 rlf REG,1                           ;Rotate and put bit into the Carry
133                 bcf Data                            ;Clear the Data bit
134                 rlf PORTA,1                         ;Shift the Carry into the Data bit
135                 ClockStrobe                         ;Send bit
136                 decfsz bitNr,1              ;Check if all bits in byte have been sent
137                 goto Sendreg                ;Repeat to send next bit
138
139         Return
140
141 ;*****New Sub
142 Dlay5:                              ;  Delay 5 msecs
143                 movlw  4                          ;  Set up the Delay
144                 movwf  Dlay
145                 movlw  256 - 0x0E8
146                 addlw  1
147                 btfsc  STATUS,2
148                 decfsz Dlay
149                 goto $-3
150
151         Return
152
153
154
155
156 ;L      b'00000011' b'10000000' b'10010011'
157
158 ;F      b'00000011' b'10000000' b'10010010'
159
160 ;R      b'00000000' b'00000000' b'00010100'
161
162 ;N      b'00000000' b'00010001' b'00000001'
163
164
165
166
167
168 ;*****End of program*****
169 end
```

Appendix C

Source Code - GUI Controller

C.1 main.cpp

```
/*************************************************************
|Ångström Synthetic Radio Telescope (ASRT) - Dish Controller GUI|
|                                                             |
|  Author: Henrik Lindén                                      |
|                                                             |
|  Last Changes: 2010-11-03                                   |
*************************************************************/

#include <QtGui/QApplication>

#include "dishwindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    DishWindow w;
    w.show();

    return a.exec();
}
```

C.2 dishwindow.h

```
#ifndef DISHWINDOW_H
#define DISHWINDOW_H

#include <QMainWindow>
#include <QMessageBox>
#include <QTcpSocket>
#include <QCloseEvent>

namespace Ui {
    class DishWindow;
}

class DishWindow : public QMainWindow
{
    Q_OBJECT

protected:
    void closeEvent(QCloseEvent *event);

```

```
20 public:
21     explicit DishWindow(QWidget *parent = 0);
22     ~DishWindow();
23
24 private slots:
25     void scan();
26     void stop();
27     void reset();
28     void calcTrack();
29     void calcScan();
30     bool yesToScan();
31     void about();
32     void help();
33     void version();
34
35     void checkDisconnect();
36     int checkConnection();
37     void checkMode();
38     void sendPosition();
39
40     void errorCon_0();
41     void errorCon_1();
42     void readReturnMsg_0();
43     void readReturnMsg_1();
44     void msgConnected_0();
45     void msgConnected_1();
46     void connectionClosedByServer_0();
47     void connectionClosedByServer_1();
48     void shift_coord();
49     void writeDefaultINI();
50
51 private:
52     Ui::DishWindow *ui;
53
54     QTcpSocket tcpSocket[2];  //create several sockets in array to be able to
           connect to both Dishes at once
55     void closeConnection();
56     void savelog(QString logMsg);
57     void connectToServer(QString ipAdress, unsigned short port, int i);
58     void msgConnected(int i);
59     void errorCon(int i);
60     void readReturnMsg(int i);
61     void connectionClosedByServer(int i);
62     bool notConnected();
63     bool stopcloseMsg();
64     void sendCommand(QString command);
65     bool noRuntime();
66     void azelToRAdec(float horizontal[], float equatorial[]);
67     void RAdecToazel(float equatorial[], float horizontal[]);
68     int checkCoord();
69     void convertTofloat(QString coord[], float equatorial[]);
70     float LST();
71     void readINI();
72
73 };
74
75 #endif // DISHWINDOW_H
```

C.3 dishwindow.cpp

```
1 #include <QtNetwork>
2 #include <QString>
3 #include <QTextCursor>
4 #include <QDateTime>
5 #include <QFile>
```

```
6  #include <QCoreApplication>
7  #include <QTextStream>
8  #include <QIntValidator>
9  #include <QtCore/qmath.h>
10 #include <QDesktopServices>
11 #include <QColor>
12 #include <QSettings>
13
14 #include "dishwindow.h"
15 #include "ui_dishwindow.h"
16 #include "helpbrowser.h"
17
18 bool scanning = false;       //global variables to set if a scan i running
19 bool nextScan = false;
20 bool tracking = false;
21
22 DishWindow::DishWindow(QWidget *parent) :
23     QMainWindow(parent),
24     ui(new Ui::DishWindow)
25 {
26     ui->setupUi(this);
27
28     //when a SIGNAL from the respective function is emitted then the function in
           SLOT is called
29     connect(&tcpSocket[0], SIGNAL(connected()), this, SLOT(msgConnected_0()));
30     connect(&tcpSocket[0], SIGNAL(disconnected()), this, SLOT(
           connectionClosedByServer_0()));
31     connect(&tcpSocket[0], SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(
           errorCon_0()));
32     connect(&tcpSocket[0], SIGNAL(readyRead()), this, SLOT(readReturnMsg_0()));
33
34     connect(&tcpSocket[1], SIGNAL(connected()), this, SLOT(msgConnected_1()));
35     connect(&tcpSocket[1], SIGNAL(disconnected()), this, SLOT(
           connectionClosedByServer_1()));
36     connect(&tcpSocket[1], SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(
           errorCon_1()));
37     connect(&tcpSocket[1], SIGNAL(readyRead()), this, SLOT(readReturnMsg_1()));
38
39     connect(ui->setfixButton, SIGNAL(clicked()), this, SLOT(sendPosition()));
40
41     //limit which numbers can be written into the "Fix Position" boxes
42     QValidator *azValidator = new QIntValidator(0, 359, this);
43     QValidator *elValidator = new QIntValidator(0, 90, this);
44     ui->azEdit->setValidator(azValidator);
45     ui->elEdit->setValidator(elValidator);
46
47     connect(ui->degreesRadioButton, SIGNAL(clicked()), this, SLOT(shift_coord()));
48     connect(ui->coordRadioButton, SIGNAL(clicked()), this, SLOT(shift_coord()));
49
50     readINI(); //read the ini file and set the values
51 }
52
53 DishWindow::~DishWindow()
54 {
55     delete ui;
56 }
57
58 //*************************************************************
59 /*functions in SLOT can't carry variables with them to other functions, to work
       around this
60 these functions just forwards the correct value based on how we are connected*/
61 void DishWindow::errorCon_0(){
62     errorCon(0);}
63 void DishWindow::errorCon_1(){
64     errorCon(1);}
65 void DishWindow::readReturnMsg_0(){
66     readReturnMsg(0);}
```

```
67  void DishWindow::readReturnMsg_1(){
68      readReturnMsg(1);}
69  void DishWindow::msgConnected_0(){
70      msgConnected(0);}
71  void DishWindow::msgConnected_1(){
72      msgConnected(1);}
73  void DishWindow::connectionClosedByServer_0(){
74      connectionClosedByServer(0);
75  }
76  void DishWindow::connectionClosedByServer_1(){
77      connectionClosedByServer(1);
78  }
79  void DishWindow::shift_coord() //Enable and disable coordinate format boxes so the
        values in them do not change during execution
80  {
81      if(ui->degreesRadioButton->isChecked())
82      {
83          ui->rightEdit->setEnabled(false);
84          ui->decEdit->setEnabled(false);
85          ui->azEdit->setEnabled(true);
86          ui->elEdit->setEnabled(true);
87      }
88      else if(ui->coordRadioButton->isChecked())
89      {
90          ui->rightEdit->setEnabled(true);
91          ui->decEdit->setEnabled(true);
92          ui->azEdit->setEnabled(false);
93          ui->elEdit->setEnabled(false);
94      }
95  }
96
97  //Connecting...
98  void DishWindow::checkMode() //Check mode and connect to correct dish
99  {
100 savelog("Checking mode...");
101
102 //Check which mode is chosen
103 if(ui->D8RadioButton->isChecked())  //if dish 8 chosen, connect
104     connectToServer(ui->ip8LineEdit->text(),ui->port8LineEdit->text().toUShort(
            0,10),0);
105 else if(ui->D7RadioButton->isChecked()) //if dish 7 chosen, connect
106     connectToServer(ui->ip7LineEdit->text(),ui->port7LineEdit->text().toUShort(
            0,10),1);
107 else if(ui->interRadioButton->isChecked())  //if interferometry chosen, connect to
        both
108 {
109     connectToServer(ui->ip8LineEdit->text(),ui->port8LineEdit->text().toUShort(
            0,10),0);
110     connectToServer(ui->ip7LineEdit->text(),ui->port7LineEdit->text().toUShort(
            0,10),1);
111 }
112 else     //if no mode
113     savelog("No mode chosen, choose mode!");
114 }
115
116 void DishWindow::connectToServer(QString ipAdress, unsigned short port, int i) //
        connect to specified dish
117 {
118 tcpSocket[i].connectToHost(ipAdress, port); //QHostAddress::LocalHost or IP
119 ui->connectButton->setEnabled(false);
120 ui->disconnectButton->setEnabled(true);
121 ui->D7RadioButton->setEnabled(false);
122 ui->D8RadioButton->setEnabled(false);
123 ui->interRadioButton->setEnabled(false);
124 savelog("Connecting to dish...");
125 }
126
```

```
127 void DishWindow::msgConnected(int i) //show connection message
128 {
129     if(i==0)
130     savelog("Connected to Dish 8!");
131     else if(i==1)
132     savelog("Connected to Dish 7!");
133 }
134
135 //Disconnecting...
136 void DishWindow::checkDisconnect() //send disconnection message and disconnect
137 {
138     if(stopcloseMsg()){      //ask if we want to disconnect
139         savelog("Disconnecting...");
140         closeConnection();
141     }
142 }
143
144 void DishWindow::closeConnection() //close connection to current dish
145 {
146 stop(); //send stop command before closing connection
147 if(ui->D8RadioButton->isChecked())  //if dish 8 chosen, disconnect...
148 {
149     tcpSocket[0].disconnectFromHost();
150     savelog("Disconnected from D8!");
151 }
152 else if(ui->D7RadioButton->isChecked()) //if dish 7 chosen, disconnect...
153 {
154     tcpSocket[1].disconnectFromHost();
155     savelog("Disconnected from D7!");
156 }
157 else if(ui->interRadioButton->isChecked())  //if interferometry chosen, disconnect
        from both
158 {
159     tcpSocket[0].disconnectFromHost();
160     savelog("Disconnected from D8!");
161     tcpSocket[1].disconnectFromHost();
162     savelog("Disconnected from D7!");
163 }
164
165 ui->connectButton->setEnabled(true);
166 ui->disconnectButton->setEnabled(false);
167 ui->D7RadioButton->setEnabled(true);
168 ui->D8RadioButton->setEnabled(true);
169 ui->interRadioButton->setEnabled(true);
170 }
171
172 //Possible Errors
173 void DishWindow::connectionClosedByServer(int i) //display error if connection lost
174 {
175     if(i==0)
176     savelog("Error: Connection closed by Dish 8!");
177     else if(i==1)
178     savelog("Error: Connection closed by Dish 7!");
179     else
180     savelog("Error: Connection closed by Dish!");
181 //closeConnection();
182 }
183
184 void DishWindow::errorCon(int i) //display network error and close connection
185 {
186 savelog(tcpSocket[i].errorString());
187 closeConnection();
188 }
189
190 int DishWindow::checkConnection() //check which connections are used and return
        corresponding integer
191 {
```

```
192     int a=tcpSocket[0].state();
193     int b=tcpSocket[1].state();
194     if ((a==3) & (b==0))
195     {
196         //savelog("0");
197         return 0;
198     }
199     else if ((b==3) & (a==0))
200     {
201         //savelog("1");
202         return 1;
203     }
204     else if ((a==3) & (b==3))
205     {
206         //savelog("2");
207         return 2;
208     }
209     else
210     {
211         //savelog("3");
212         notConnected();
213         if(scanning)
214             scanning = false;
215         return 3;
216     }
217 }
218
219 void DishWindow::readReturnMsg(int i) //read and display messages from controller
220 {
221     float AZEL[2],RADEC[2];
222     AZEL[0]=-1;
223
224     while(tcpSocket[i].canReadLine ()) //read as long as there are new messages to
            read
225     {
226         QString text;
227         text = tcpSocket[i].readLine();
228         savelog("Return msg: " + text);
229
230         if(text.contains("Azimuth", Qt::CaseSensitive))      //print current
                position to lables
231         {
232             ui->azposLabel->setText(text);
233             QStringList RADecpos = text.split("=", QString::SkipEmptyParts); //
                    extract the numberpart of the position
234             AZEL [0]= RADecpos[1].toFloat();
235
236         }
237         else if(text.contains("Elevation", Qt::CaseSensitive))
238         {
239             ui->elposLabel->setText(text);
240             QStringList RADecpos = text.split("=", QString::SkipEmptyParts); //
                    extract the numberpart of the position
241             AZEL [1]= RADecpos[1].toFloat();
242
243             if(scanning)
244                 nextScan=true;
245             else if(tracking)
246                 nextScan=true;
247         }
248         //this code should be removed when endstop bug is removed
249         else if(text.contains("End stop!", Qt::CaseSensitive))
250             nextScan=true;
251     }
252
253     if(AZEL[0]>0) //convert the coordinates sent from the controller as AZEL and
            print them as RADEC on labels
```

```
254     {
255         QString coord[2];
256         azelToRAdec(AZEL,RADEC); //converting to RADEC
257 
258         int hh = int(RADEC[0]/15); //split up right ascension to the correct format
259         int mm = int((RADEC[0]/15 - hh)*60);
260         float ss = ((RADEC[0]/15 - hh)*60 - mm)*60;
261 
262         QString RAhour,RAminute,RAsecond;
263         RAhour.setNum(hh); //number to string conversion
264         RAminute.setNum(mm);
265         RAsecond.setNum(ss);
266 
267         float dec = RADEC[1]; //split up declination to the correct format
268         int deg = int(dec);
269         mm = int((dec - deg)*60);
270         ss = ((dec - deg)*60 - mm)*60;
271 
272         QString Ddeg,Dminute,Dsecond;
273         Ddeg.setNum(deg);  //number to string conversion
274         Dminute.setNum(mm);
275         Dsecond.setNum(ss);
276 
277         coord[0] = RAhour + ":" + RAminute + ":" + RAsecond;
278         coord[1] = Ddeg + ":" + Dminute + ":" + Dsecond;
279         ui->raposLabel->setText("Right Ascension = " + coord[0] + "\n"); //print to
                labels
280         ui->decposLabel->setText("Declination = " + coord[1] + "\n");
281     }
282 }
283 
284 void DishWindow::sendPosition() //send new position to controller
285 {
286     int coord = checkCoord();
287 
288     if(coord==0) //Absolute degrees
289     {
290         QString position;
291         position = "Az," + ui->azEdit->text() + ";" + "El," + ui->elEdit->text();
292         sendCommand(position);
293     }
294     else if(coord==1) //Equatorial coordinates
295     {
296         float horizontal[2], equatorial[2];
297         QString coordinates[2];
298         coordinates[0] = ui->rightEdit->text();
299         coordinates[1] = ui->decEdit->text();
300 
301         convertTofloat(coordinates, equatorial);
302 
303         RAdecToazel(equatorial, horizontal);
304 
305         QString position,AZ,EL;
306         AZ.setNum(qRound(horizontal[0]));
307         EL.setNum(qRound(horizontal[1]));
308         position = "Az," + AZ  + ";" + "El," + EL;
309         savelog("Calulated position" + position);
310 
311         if(horizontal[1]>=0) //check if the elevation is below zero, meaning the
              object is below the horizon, and if not send the command
312         {
313             sendCommand(position); //send new position
314         }
315         else
316             savelog("The object is currently below the horizon and can not be
                  targeted!");
317     }
```

```
318
319 }
320
321 void DishWindow::scan() //start scanning the hemisphere
322 {
323  if(checkConnection()==3) //check for connection before running scan
324      return;
325
326  if(yesToScan())     //check if we want to run the scan
327  {
328
329   //Get scan duration from spinbox
330   QString t;
331   int q=0;
332   t = ui->runtimeEdit->sectionText(QDateTimeEdit::HourSection); //get # of hours as
        text
333   q += 3600*t.toInt();                                          //convert text-
      hours to int as seconds
334   t = ui->runtimeEdit->sectionText(QDateTimeEdit::MinuteSection);
335   q += 60*t.toInt();
336   t = ui->runtimeEdit->sectionText(QDateTimeEdit::SecondSection);
337   q += t.toInt();
338
339   int total_scan_time = q; //total duration in seconds that tha scan will be run
      (86400 = one whole day)
340
341   if(total_scan_time ==0) //make sure a runtime is set adn display dialog and
      logmessage if not
342   {
343       noRuntime();
344       return;
345   }
346
347   ui->resetButton->setEnabled(false);
348   ui->setfixButton->setEnabled(false);
349   ui->scanButton->setEnabled(false);
350   savelog("Scanning...");
351
352   savelog("Duration of scan is " + QString::number(total_scan_time) + " seconds.");
353
354
355   QFile posFile("conf/default.pos");      //open position file
356        if (!posFile.open(QIODevice::ReadOnly | QIODevice::Text)) //check if it is
          open
357            return;
358
359        QTextStream getPos(&posFile);     //create stream of text from file
360        QString line;
361
362        //line = getPos.readLine();     //read first line which contains number of
          sweeps to run
363        //int sweeps = line.toInt(); //make the first string line into an integer
364        //savelog(line + " sweeps per hour");
365
366        line = getPos.readLine();     //read second line which contains time between
          movements
367        int count_down = line.toInt(); //make the second string line into an integer
368        savelog(line + " seconds between each new positon in scan pattern");
369        posFile.close();
370
371        ui->progressBar->setEnabled(true);
372        QTime progressTime; //create a timer for how long the scann has ben runnning
373        ui->progressBar->setRange(0, total_scan_time); //set the range of the
          progressbar based on how long the scan will be running
374
375        QTime scanTime = QTime::currentTime().addSecs(total_scan_time); //wait for
          the amount of seconds specified in the file before allowing next sweep
```

```
            to be run
376
377         progressTime.start(); //start calculating the elapsed time for display in
                the progressbar
378
379         while( (QTime::currentTime() < scanTime) && scanning)   //while waiting and
                the scanning has not been stopped...
380         {
381          //int a=1; //counter to increment every time a sweep has been run upto the
                 correct amounts of sweeps has been run
382          //while (scanning && (QTime::currentTime() < scanTime))
383          //{
384             QFile posFile("conf/default.pos");      //open position file
385                  if (!posFile.open(QIODevice::ReadOnly | QIODevice::Text)) //check
                        if it is open
386                      return;
387
388                  QTextStream getPos(&posFile);
389                  QString line;
390
391                  //line = getPos.readLine();     //can't move directly to third line
                        to get command,...
392                  line = getPos.readLine();     //...so must read the first two lines
                        before reading the first position
393
394              while (!getPos.atEnd() && scanning && (QTime::currentTime() < scanTime)
                    )       //run as long as you have not reached the end of the file,
                    and while scanning varible is still true and scanning duration has
                    not ended
395              {
396                  if(nextScan) //run if the previous command have been executed
397                   {
398                      line = getPos.readLine();     //get the next position from the
                            file
399                      sendCommand(line);              //send position command for scan
400                      savelog("Moving to: " + line); //log current position
401                      nextScan=false;
402                   }
403
404                  ui->progressBar->setValue(progressTime.elapsed()/1000); //set
                        progressbar to elapsed time since scan begun
405                  QTime waitTime = QTime::currentTime().addSecs(count_down); //wait
                        for the amount of seconds specified in the file before allowing
                         next position to be run
406                  while( (QTime::currentTime() < waitTime) && scanning){      //while
                        waiting and the scanning has not been stopped...
407                  QCoreApplication::processEvents(QEventLoop::AllEvents);}  //...
                        check if other commands want to run
408              }
409
410              //a++;
411              posFile.close();       //close the position file
412          //}
413         }
414
415     if(posFile.open(QIODevice::ReadOnly | QIODevice::Text))
416           posFile.close();       //close the position file
417
418     if(scanning)  //if still scanning then the scan has been completed
419       {
420           savelog("Scan Completed!");
421           scanning = false;      //mark that scanning is not running
422       }
423     else if(!scanning)
424         savelog("Scan Interrupted!");
425
426     ui->progressBar->reset();
```

```
427     ui->progressBar->setEnabled(false);
428     ui->resetButton->setEnabled(true);
429     ui->setfixButton->setEnabled(true);
430     ui->scanButton->setEnabled(true);
431   }
432 }
433
434 void DishWindow::sendCommand(QString command) //send command to the correct dish
435 {
436     if(command.contains("Az,", Qt::CaseSensitive)) //check if azimuth is set to 360
            degrees and change to 0 if it is
437     {
438         QStringList newCom = command.split(";", QString::SkipEmptyParts); //split
                the command into parts so that the degrees of...
439         QStringList newPos = newCom[0].split(",", QString::SkipEmptyParts);//...
                azimuth can be evaluated
440
441         if(newPos[1]=="360")
442         {
443             command = "Az,0;" + newCom[1];
444             savelog("Azimuth 360 was changed to 0!");
445         }
446     }
447
448     int i = checkConnection();
449
450     if (i == 0) //if connected to dish 8
451     {
452         QTextStream stream_0(&tcpSocket[0]);
453         stream_0 << command <<"\n";
454     }
455     else if (i == 1) //if connected to dish 7
456     {
457         QTextStream stream_1(&tcpSocket[1]);
458         stream_1 << command <<"\n";
459     }
460     else if (i == 2) //if connected to both dishes
461     {
462         QTextStream stream_0(&tcpSocket[0]);
463         stream_0 << command <<"\n";
464         QTextStream stream_1(&tcpSocket[1]);
465         stream_1 << command <<"\n";
466     }
467     else if (i == 3)
468     {
469         savelog("Command could not be sent!");
470     }
471 }
472
473 void DishWindow::stop() //Stops the dish from completing its move action
474 {
475     sendCommand("stop");
476     if(scanning)
477         scanning = false;
478     else if(tracking)
479         tracking = false;
480 }
481
482 void DishWindow::reset() //reset dish to default position
483 {
484     sendCommand("reset");
485 }
486
487 void DishWindow::version() //check version of controllerchip firmware
488 {
489     sendCommand("ver");
490 }
```

```
491
492 void DishWindow::calcScan() /*calculate scan of the hemisphere (could include
        depending on start azimuth position), save to file which can be read from
        during scan*/
493 {
494     QString path = HelpBrowser::directoryOf("conf").absolutePath(); //get path of
          ini file
495     QSettings settings(path + "/config.ini", QSettings::IniFormat); //read settings
          from .ini file
496
497     float timeunit = 3600; //1=hours, 60=minutes, 3600=seconds etc.
498     float earth_rotspeed = (float)360/(24*timeunit); //in degrees per "timeunit"
          86164
499     float beam_width = settings.value("settings/beam_width", 7.0).toFloat(); //beam
          width in degrees (see specifications for dish)
500     //float max_dish_movement = 180/195; //in degrees per second (~0.92)
501
502     //formula to calculate which positions to move to during the scan
503     //float x = float(qPow(timeunit,2));
504     //float y = 360*beam_width;
505     //float time_per_degree = (x*earth_rotspeed)/y;
506     //float ideal_dish_movement = (float)180/time_per_degree; //how many degrees
          the dish should move per timeunit
507
508     QString azPos = "0"; //ui->azEdit->text(); //starting azimut position
509     int move_degrees = (int)beam_width; //qRound(ideal_dish_movement); //how many
          degrees between each position in a sweep
510
511     int number_of_positions = qRound(180/move_degrees); //Number of positions the
          dish has to move to during one sweep
512     int number_of_sweeps = qRound(earth_rotspeed*3600/beam_width); //number of
          sweeps needed to cover one hour of earth rotation
513     int time_between_move = qRound(3600/(number_of_positions*number_of_sweeps)); //
          how much idle time needed between moving to next position
514
515     /*open and save scan positions to list to file*/
516     QFile posFile("conf/default.pos");
517     if (!posFile.open(QIODevice::WriteOnly | QIODevice::Text)) //check if file is
          open
518         return;
519
520     QTextStream position(&posFile);
521     //position << number_of_sweeps << "\n";
522     //posFile.close();
523
524     posFile.open(QIODevice::Append | QIODevice::Text); //check if file is open
525     position << time_between_move << "\n";
526     position << "reset" << "\n"; //set reset as the first position to make sure
          that no calculation errors during movement are left
527
528     int el=0;
529     for(int elPos=move_degrees;elPos<=180;elPos=elPos+move_degrees)
530     {
531         position << "Az," << azPos << ";" << "El," << elPos << "\n";
532         el=elPos;
533     }
534     if(el!=180)
535          position << "Az," << azPos << ";El,180\n";
536     else if(el==180)
537          el -= move_degrees;
538
539     for(int elPos=el;elPos>0;elPos=elPos-move_degrees)
540     {
541         position << "Az," << azPos << ";" << "El," << elPos << "\n";
542     }
543     posFile.close();
544 }
```

```
545
546 void DishWindow::calcTrack() //start tracking the specified point based upon the
        coordinate type choosen
547 {
548     if(checkConnection()==3) //check for connection before running tracking
549         return;
550
551     int coord = checkCoord(); //check which coordinate system is used
552
553     QString t;
554     int q=0;
555     t = ui->tracktimeEdit->sectionText(QDateTimeEdit::HourSection); //get # of
            hours as text
556     q += 3600*t.toInt();                                          //convert text-
            hours to int as seconds
557     t = ui->tracktimeEdit->sectionText(QDateTimeEdit::MinuteSection);
558     q += 60*t.toInt();
559     t = ui->tracktimeEdit->sectionText(QDateTimeEdit::SecondSection);
560     q += t.toInt();
561
562     int total_tracking_time = q; //total duration in seconds that tha scan will be
            run (86400 = one whole day)
563
564     if(total_tracking_time == 0) //make sure a runtime is set and display dialog
            and logmessage if not
565     {
566         noRuntime();
567         return;
568     }
569
570     ui->degreesRadioButton->setEnabled(false);
571     ui->coordRadioButton->setEnabled(false);
572     ui->trackButton->setEnabled(false);
573     ui->resetButton->setEnabled(false);
574     ui->setfixButton->setEnabled(false);
575     ui->scanButton->setEnabled(false);
576
577     savelog("Tracking...");
578
579     float horizontal[2], equatorial[2];
580
581     tracking = true;
582     nextScan = true;
583
584     savelog("Duration␣of␣tracking␣is␣" + QString::number(total_tracking_time) + "␣
            seconds.");
585
586     if(coord==0) //Horizontal coordinates degrees
587     {
588         //Get Az/El from user
589         QString coordinates[2];
590         coordinates[0] = ui->azEdit->text();
591         coordinates[1] = ui->elEdit->text();
592
593         horizontal[0] = coordinates[0].toFloat();
594         horizontal[1] = coordinates[1].toFloat();
595         savelog("AZ->RA");
596         //Convert Az/El coordinates to RA/Dec and save them in equatorial[]
597         azelToRAdec(horizontal, equatorial);
598     }
599     else if(coord==1) //Equatorial coordinates
600     {
601         //Get RA/Dec from user
602         QString coordinates[2];
603         coordinates[0] = ui->rightEdit->text();
604         coordinates[1] = ui->decEdit->text();
605         //Convert text format flaots and send back as degrees saved in equatorial[]
```

```
606         convertTofloat(coordinates, equatorial);
607     }
608
609     QString RA,DEC; //print the position in RA/Dec coordinates, in degrees
610             RA.setNum(equatorial[0]);
611             DEC.setNum(equatorial[1]);
612             savelog("before loop [deg] RA " + RA + " DEC " + DEC);
613
614     QString position,AZ,EL,lastPosition="default";
615     ui->progressBar->setEnabled(true);
616     QTime progressTime; //create a timer for how long the scan has ben runnning
617     ui->progressBar->setRange(0, total_tracking_time); //set the range of the
            progressbar based on how long the scan will be running
618
619     QTime trackingTime = QTime::currentTime().addSecs(total_tracking_time); //wait
            for the amount of seconds specified in the file before allowing next sweep
            to be run
620
621     progressTime.start(); //start calculating the elapsed time for display in the
            progressbar
622
623     while ( (QTime::currentTime() < trackingTime) && tracking)       //run while
            tracking varible is still true and tracking duration has not ended
624     {
625         if(nextScan && tracking) //run if the previous command have been executed
626          {
627             savelog("RA->AZ");
628             //Send the RA/Dec coordinates for conversion back to Az/El, but since
                    som time has passed they will have changed some
629             RAdecToazel(equatorial, horizontal);
630             //convert Az/El to strings and format them accoringly so they can be
                    sent to the controllerchip
631             AZ.setNum(qRound(horizontal[0])); //horizontal[] must be rounded to
                    integers as the controller only can receive integer degrees
632             EL.setNum(qRound(horizontal[1]));
633             position = "Az," + AZ  + ";" + "El," + EL;
634
635             if((position.contains(lastPosition, Qt::CaseSensitive))) //check if the
                     new position is the same as the last one, if it is print it
636             {
637                 savelog("Same position: " + position + " = " + lastPosition);
638             }
639             else
640             {
641                 if(horizontal[1]>=0) //check if the elevation is below zero,
                        meaning the object is below the horizon, and if not send the
                        command
642                 {
643                     sendCommand(position); //send new position
644                     nextScan=false;
645                     lastPosition=position; //save new position as last position
646                     savelog("Last position " + lastPosition);
647
648                     savelog("calctrack horizontal[deg] " + position);
649                 }
650                 else
651                     savelog("The object is currently below the horizon and can not
                            be tracked!\n Tracking will start as soon as it is visible!
                            ");
652             }
653          }
654
655         ui->progressBar->setValue(progressTime.elapsed()/1000); //set progressbar
                to elapsed time since scan begun
656         QTime waitTime = QTime::currentTime().addSecs(60); //wait for the amount of
                seconds specified in the file before allowing next position to be run
```

```
657         while( (QTime::currentTime() < waitTime) && tracking){      //while waiting
                and the tracking has not been stopped...
658         QCoreApplication::processEvents(QEventLoop::AllEvents);}  //...check if
                other commands want to run
659
660      //Continue to send the saved RA/Dec coordinates to keep tracking the object.
661     }
662
663     if(tracking)  //if still scanning then the scan has been completed
664       {
665           savelog("Tracking Completed!");
666           tracking = false;      //mark that scanning is not running
667       }
668     else if(!tracking)
669         savelog("Tracking Interrupted!");
670
671     ui->progressBar->reset();
672     ui->progressBar->setEnabled(false);
673     ui->degreesRadioButton->setEnabled(true);
674     ui->coordRadioButton->setEnabled(true);
675     ui->resetButton->setEnabled(true);
676     ui->setfixButton->setEnabled(true);
677     ui->scanButton->setEnabled(true);
678     ui->trackButton->setEnabled(true);
679     tracking = false;
680     nextScan = false;
681 }
682
683 int DishWindow::checkCoord() //check which coordinate format is used and return the
       corresponding integer
684 {
685     if(ui->degreesRadioButton->isChecked())
686     {
687         return 0;
688     }
689     else if(ui->coordRadioButton->isChecked())
690     {
691         return 1;
692     }
693     else
694         return 2;
695 }
696
697 float DishWindow::LST() //calculates local siderial time and returns it as float
698 {
699     const float PI = 3.14159265358979;
700     float dish_geographical_longitude = 17.648733333333332*PI/180; //longitude of
            dish 8
701
702     QDateTime time_date = QDateTime::currentDateTime().toUTC(); //get current date
            and time and convert to UTC
703     QString timedateString = time_date.toString("yyyy:M:d:h:m:s"); //put
            date and time in a string using using ":" to split up teh values
704     QStringList timedateList = timedateString.split(":", QString::SkipEmptyParts);
            //make a list of the date/time value using ":" to separte them
705
706     float year = timedateList[0].toFloat(); //convert each element in the list to a
             float for later computation
707     float month = timedateList[1].toFloat();
708     float day = timedateList[2].toFloat();
709     float hour = timedateList[3].toFloat();
710     float minute = timedateList[4].toFloat();
711     float second = timedateList[5].toFloat();
712
713     float current_hour = hour + minute/60 + second/3600; //convert current time
            into fractions of hours
714
```

```
715     int z = (month+9)/12;
716     int x = (7*(year + z))/4;
717     int y = 275*month/9;
718     float days_since_J2000 = 367*year - x + y + day - 730530 + current_hour/24;
719
720     float LST = 98.9818 + 0.985647352 * days_since_J2000 +
            dish_geographical_longitude*180/PI + 15*current_hour; //compute the local
            siderial time in degrees
721     int c = (LST/360); //value of LST must be within 0-360 deggrees
722     float local_sidereal_time = LST - 360*c;
723
724     return local_sidereal_time;
725 }
726
727 void DishWindow::azelToRAdec(float horizontal[], float equatorial[]) //convert
        Azimuth/Elevation coordinates to Right Ascension/Declination
728 {
729     QString log;
730     const float PI = 3.14159265358979;
731     float dish_geographical_latitude = 59.83773333333333*PI/180; //latitude of dish
            8
732
733     float local_sidereal_time = LST();
734
735     //savelog("LST " + log.setNum(local_sidereal_time)); //get the local siderial
            time
736
737     float azimuth = horizontal[0]*PI/180; //convert to flaot and radians
738     float elevation = horizontal[1]*PI/180;
739     //savelog("az[rad] " + log.setNum(azimuth));
740     //savelog("el[rad] " + log.setNum(elevation));
741
742     float declination = qAsin(qSin(dish_geographical_latitude)*qSin(elevation) +
            qCos(dish_geographical_latitude)*qCos(elevation)*qCos(azimuth));
743     //savelog("dec[rad] " + log.setNum(declination));
744     float h = (qAtan2(-qSin(azimuth)*qCos(elevation),(qCos(
            dish_geographical_latitude)*qSin(elevation) - qSin(
            dish_geographical_latitude)*qCos(elevation)*qCos(azimuth))))*180/PI;
745     //savelog("h[deg] " + log.setNum(h));
746
747     float hour_angle; //value of hour angle must be positive 0-360 degrees
748     if(h<0)
749         hour_angle = (360+h);
750     else
751         hour_angle = h;
752
753     //savelog("hour_angle[deg] " + log.setNum(hour_angle));
754
755     float right_ascension = local_sidereal_time - hour_angle;
756
757     equatorial[0] = right_ascension;
758     //savelog("right_ascension[deg] " + log.setNum(equatorial[0]));
759     equatorial[1] = declination*180/PI;
760     //savelog("declination[deg] " + log.setNum(equatorial[1]));
761 }
762
763 void DishWindow::RAdecToazel(float equatorial[], float horizontal[]) //convert
        Right Ascension/Declination to Azimuth/Elevation coordinates
764 {
765     QString log;
766     const float PI = 3.14159265358979;
767     float dish_geographical_latitude = 59.83773333333333*PI/180; //latitude of dish
            8
768
769     float local_sidereal_time = LST(); //get the local siderial time
770
771     //savelog("LST " + log.setNum(local_sidereal_time));
```

```
772
773     float right_ascension = equatorial[0];
774     float declination = equatorial[1]*PI/180; //convert to radians
775
776     //savelog("convReturn RA[deg]: " + log.setNum(right_ascension));
777     //savelog("convReturn Dec[rad]: " + log.setNum(declination));
778
779     float h = (local_sidereal_time - right_ascension); //calculate hour_angle in
            degrees
780
781     //savelog("h[deg] " + log.setNum(h));
782
783     float hour_angle; //value of hour angle must be positive 0-360 degrees
784     if(h<0)
785         hour_angle = (360+h)*PI/180;
786     else
787         hour_angle = h*PI/180;
788
789     //savelog("hour angle[rad] " + log.setNum(hour_angle));
790
791     float elevation = qAsin(qSin(dish_geographical_latitude)*qSin (declination) +
            qCos(dish_geographical_latitude)*qCos(declination)*qCos(hour_angle)); //
            calculate the elevation in radians
792     float a = (qAtan2(-qCos(declination)*qSin(hour_angle),(qSin(declination)*qCos(
            dish_geographical_latitude)-qCos(declination)*qCos(hour_angle)*qSin(
            dish_geographical_latitude))))*180/PI;
793
794     //savelog("a " + log.setNum(a)); //value of azimuth must be within 0-360
            degrees
795     float azimuth;
796     if(a<0)
797         azimuth=360+a;
798     else
799         azimuth=a;
800
801     horizontal[0] = azimuth;
802     horizontal[1] = elevation*180/PI;
803 }
804
805 void DishWindow::convertTofloat(QString coord[], float equatorial[]) //takes right
        ascension/declination from the string array and converts them to floats in
        degrees
806 {
807     QStringList coordList1 = coord[0].split(":", QString::SkipEmptyParts); //split
            up the strings containting RA/Dec into separate arrays depending on where
            ':' is placed
808     QStringList coordList2 = coord[1].split(":", QString::SkipEmptyParts);
809     equatorial[0] = (coordList1[0].toFloat() + coordList1[1].toFloat()/60 +
            coordList1[2].toFloat()/3600)*15; //convert RA values to flaots and convert
             into degrees
810     if (coordList2[0].toFloat() < 0) //check if value is negative and add
            accoringly
811         equatorial[1] = coordList2[0].toFloat() - qAbs((coordList2[1].toFloat() +
                coordList1[2].toFloat()/60)/60); //convert Dec values to flaots and
                convert into degrees
812     else
813         equatorial[1] = coordList2[0].toFloat() + (coordList2[1].toFloat() +
                coordList1[2].toFloat()/60)/60; //convert Dec values to flaots and
                convert into degrees
814
815     QString log;
816     //savelog("conversion RA[deg]: " + log.setNum(equatorial[0]));
817     //savelog("conversion Dec[deg]: " + log.setNum(equatorial[1]));
818 }
819
820 void DishWindow::savelog(QString logMsg) //save event to logfile and show in
        logwindow
```

```
821 {
822     if(logMsg.contains("Return msg:", Qt::CaseSensitive)) //print messages from the
          controllerchip in dark red to make them easier to distinguish
823     {
824         ui->logTextEdit->setTextColor(QColor("darkRed"));
825         ui->logTextEdit->append(logMsg);
826         ui->logTextEdit->setTextColor(QColor("black"));
827     }
828     else
829     ui->logTextEdit->append(logMsg); //print msg to logwindow
830
831     //get current date
832     QDate date = QDate::currentDate();
833     QString dateString = date.toString("yyyy.MM.dd");
834
835     //get current time
836     QTime time = QTime::currentTime();
837     QString timeString = time.toString();
838
839     //open and save log message to log file
840     QFile logFile("log/" + dateString + ".log");
841     if (!logFile.open(QIODevice::Append | QIODevice::Text)) //check if file is open
842         return;
843     QTextStream out(&logFile);
844     out << timeString << " " << logMsg << "\n";
845     logFile.close();
846 }
847
848 void DishWindow::about()     //show about dialog
849 {
850     QMessageBox::about(this, tr("About ÅSRT - Dish Controller"),
851     tr("<h2>Ångström Synthesis Radio Telescope (ÅSRT) - Dish Controller 0.8</h2>"
852         "<p>Author: Henrik Lindén"
853         "<p>Last changed on: 2010-11-17"
854         "<p>Dish Controller connects to the controllerchip on the dishes "
855         "of the Ångström Synthesis Radio Telescope (ÅSRT))."));
856 }
857
858 void DishWindow::help() //open helpfile browser
859 {
860     HelpBrowser::showPage("index.html");
861 }
862
863 bool DishWindow::yesToScan()  //check if the person really want to run the scan
864 {
865     int r = QMessageBox::warning(this, tr("ÅSRT - Dish Controller"),
866                                  tr("This will start scan of the hemisphere. It might
                                       take a several hours."
867                                     "<p>Are your sure you want to run the scan?"),
868                                  QMessageBox::Yes | QMessageBox::No);
869      if (r == QMessageBox::Yes)
870     {
871         nextScan = true;
872         scanning = true;
873         return true;
874     }
875
876     return false;
877
878 }
879
880 bool DishWindow::notConnected()  //display not connected dialog
881 {
882     int r = QMessageBox::information(this, tr("ÅSRT - Dish Controller"),
883                                  tr("You are not connected to any dish!"
884                                     "<p>Connect and try again."),
885                                  QMessageBox::Ok);
```

```
886     if (r == QMessageBox::Ok){
887         savelog("No Connection!");
888                 return true;
889             }
890     else{
891     savelog("No Connection!");
892     return true;
893     }
894
895 }
896
897 bool DishWindow::noRuntime() //display no runtime set dialog
898 {
899     int r = QMessageBox::information(this, tr("ÅSRT - Dish Controller"),
900                                 tr("No Runtime has been set!"
901                                    "<p>Set Runtime and try again."),
902                                 QMessageBox::Ok);
903     if (r == QMessageBox::Ok){
904         savelog("No Runtime has been set!");
905                 return true;
906             }
907     else{
908         savelog("No Runtime has been set!");
909     return true;
910     }
911 }
912
913 bool DishWindow::stopcloseMsg() //Check if want to diconnect and stop
914 {
915     int r = QMessageBox::warning(this, tr("ÅSRT - Dish Controller"),
916                                 tr("You are connected to a dish! This will stop any
                                        running scans and positon settings!"
917                                    "<p>Are your sure you want to proceed?"),
918                                 QMessageBox::Yes | QMessageBox::No);
919             if (r == QMessageBox::Yes)
920                 return true;
921
922     return false;
923 }
924
925 void DishWindow::readINI()
926 {
927     QString text;
928     QString path = HelpBrowser::directoryOf("conf").absolutePath();
929     QSettings settings(path + "/config.ini", QSettings::IniFormat);
930
931     text = settings.value("communication/ipadress8", "error").toString();
932     ui->ip8LineEdit->setText(text);
933     text = settings.value("communication/port8", "error").toString();
934     ui->port8LineEdit->setText(text);
935     text = settings.value("communication/ipadress7", "error").toString();
936     ui->ip7LineEdit->setText(text);
937     text = settings.value("communication/port7", "error").toString();
938     ui->port7LineEdit->setText(text);
939
940     text = settings.value("position/azimuth", "error").toString();
941     ui->azEdit->setText(text);
942     text = settings.value("position/elevation", "error").toString();
943     ui->elEdit->setText(text);
944     text = settings.value("position/right_ascension", "error").toString();
945     ui->rightEdit->setText(text);
946     text = settings.value("position/declination", "error").toString();
947     ui->decEdit->setText(text);
948 }
949
950 void DishWindow::writeDefaultINI()
951 {
```

```
952     QString text;
953     QString path = HelpBrowser::directoryOf("conf").absolutePath();
954     QSettings settings(path + "/config.ini", QSettings::IniFormat);
955 
956     //Communications
957     settings.beginGroup("communication");
958     settings.setValue("ipadress8", "130.238.30.234");
959     text = settings.value("ipadress8", "error").toString();
960     ui->ip8LineEdit->setText(text);
961 
962     settings.setValue("port8", 5001);
963     text = settings.value("port8", "error").toString();
964     ui->port8LineEdit->setText(text);
965 
966     settings.setValue("ipadress7", "130.238.30.200");
967     text = settings.value("ipadress7", "error").toString();
968     ui->ip7LineEdit->setText(text);
969 
970     settings.setValue("port7", 5001);
971     text = settings.value("port7", "error").toString();
972     ui->port7LineEdit->setText(text);
973     settings.endGroup();
974 
975     //Position
976     settings.beginGroup("position");
977     settings.setValue("azimuth", 0);
978     text = settings.value("azimuth", "error").toString();
979     ui->azEdit->setText(text);
980 
981     settings.setValue("elevation", 0);
982     text = settings.value("elevation", "error").toString();
983     ui->elEdit->setText(text);
984 
985     settings.setValue("right_ascension", "0:42:44.3");
986     text = settings.value("right_ascension", "error").toString();
987     ui->rightEdit->setText(text);
988 
989     settings.setValue("declination", "41:16:9");
990     text = settings.value("declination", "error").toString();
991     ui->decEdit->setText(text);
992     settings.endGroup();
993 
994     //Settings
995     settings.beginGroup("settings");
996     settings.setValue("beam_width", 7.0);
997     settings.endGroup();
998 }
999 
1000 void DishWindow::closeEvent(QCloseEvent *event) //check if there are open TCPIP
        connections, close them and write log to file before exiting program
1001 {
1002     if ((tcpSocket[0].state() !=0) | (tcpSocket[1].state() !=0)) //check if there
            is a connection, ask to close it and quit program
1003     {
1004         if(stopcloseMsg())
1005         {
1006             closeConnection();
1007             savelog("Exiting program!\n");
1008             event->accept();
1009         }
1010         else
1011             event->ignore();
1012     }
1013     else
1014     {
1015         savelog("Exiting program!\n");
1016         event->accept();
```

```
1017     }
1018 }
```

C.4 helpbrowser.h

```
1  #ifndef HELPBROWSER_H
2  #define HELPBROWSER_H
3
4  #include <QString>
5  #include <QWidget>
6  #include <QDir>
7
8  namespace Ui {
9      class Helpwindow;
10 }
11
12 class HelpBrowser : public QWidget
13 {
14     Q_OBJECT
15
16 private:
17     Ui::Helpwindow *hb;
18
19 public:
20     HelpBrowser(const QString &path, const QString &page, QWidget *parent = 0); //
           constructor
21     ~HelpBrowser();
22
23     static void showPage(const QString &page);
24     static QDir directoryOf(const QString &subdir);
25
26 private slots:
27     void updateWindowTitle();
28     void home();
29     void backward();
30 };
31
32 #endif // HELPBROWSER_H
```

C.5 helpbrowser.cpp

```
1  #include <QString>
2  #include <QDir>
3
4  #include "ui_helpbrowser.h"
5  #include "helpbrowser.h"
6
7
8  HelpBrowser::HelpBrowser(const QString &path, const QString &page, QWidget *parent)
       : QWidget(parent), hb(new Ui::Helpwindow)
9  {
10     hb->setupUi(this);
11
12     //setAttribute(Qt::WA_DeleteOnClose);
13     setAttribute(Qt::WA_GroupLeader);
14     hb->closeButton->setShortcut(tr("Esc"));
15
16     hb->textBrowser->setSearchPaths(QStringList() << path << ":/img");
17     hb->textBrowser->setSource(page);
18 }
19
```

```
20 HelpBrowser::~HelpBrowser()
21 {
22     delete hb;
23 }
24
25 void HelpBrowser::updateWindowTitle()
26 {
27     setWindowTitle(tr("ÅSRT - Help: %1").arg(hb->textBrowser->documentTitle()));
28 }
29
30 QDir HelpBrowser::directoryOf(const QString &subdir)
31 {
32     QDir dir(QApplication::applicationDirPath());
33         dir.cd(subdir);
34
35     return dir;
36 }
37
38 void HelpBrowser::showPage(const QString &page)
39 {
40     QString path = directoryOf("doc").absolutePath();
41     HelpBrowser *browser = new HelpBrowser(path, page);
42     browser->show();
43 }
44
45 void HelpBrowser::home()
46 {
47     hb->textBrowser->home();
48 }
49
50 void HelpBrowser::backward()
51 {
52     hb->textBrowser->backward();
53 }
```

C.6 ui_dishwindow.h

```
1  /********************************************************************************
2  ** Form generated from reading UI file 'dishwindow.ui'
3  **
4  ** Created: Thu 4. Nov 16:10:20 2010
5  **      by: Qt User Interface Compiler version 4.6.3
6  **
7  ** WARNING! All changes made in this file will be lost when recompiling UI file!
8  ********************************************************************************/
9
10 #ifndef UI_DISHWINDOW_H
11 #define UI_DISHWINDOW_H
12
13 #include <QtCore/QVariant>
14 #include <QtGui/QAction>
15 #include <QtGui/QApplication>
16 #include <QtGui/QButtonGroup>
17 #include <QtGui/QFrame>
18 #include <QtGui/QGroupBox>
19 #include <QtGui/QHBoxLayout>
20 #include <QtGui/QHeaderView>
21 #include <QtGui/QLabel>
22 #include <QtGui/QLineEdit>
23 #include <QtGui/QMainWindow>
24 #include <QtGui/QMenu>
25 #include <QtGui/QMenuBar>
26 #include <QtGui/QProgressBar>
27 #include <QtGui/QPushButton>
```

```
28 #include <QtGui/QRadioButton>
29 #include <QtGui/QSpacerItem>
30 #include <QtGui/QTextEdit>
31 #include <QtGui/QTimeEdit>
32 #include <QtGui/QVBoxLayout>
33 #include <QtGui/QWidget>
34
35 QT_BEGIN_NAMESPACE
36
37 class Ui_DishWindow
38 {
39 public:
40     QAction *actionAbout;
41     QAction *actionExit;
42     QAction *actionHelp;
43     QAction *actionControllerchip_version;
44     QAction *actionCalculate_default_scan_pattern;
45     QAction *action_Set_default_settings;
46     QWidget *centralwidget;
47     QGroupBox *positionBox;
48     QWidget *layoutWidget;
49     QHBoxLayout *horizontalLayout_3;
50     QVBoxLayout *verticalLayout_12;
51     QPushButton *setfixButton;
52     QSpacerItem *verticalSpacer_3;
53     QLabel *posLabel;
54     QSpacerItem *verticalSpacer_4;
55     QVBoxLayout *verticalLayout_7;
56     QRadioButton *degreesRadioButton;
57     QHBoxLayout *horizontalLayout_2;
58     QVBoxLayout *verticalLayout_6;
59     QLabel *azLabel;
60     QLabel *elLabel;
61     QVBoxLayout *verticalLayout_5;
62     QLineEdit *azEdit;
63     QLineEdit *elEdit;
64     QLabel *azposLabel;
65     QLabel *elposLabel;
66     QVBoxLayout *verticalLayout_8;
67     QRadioButton *coordRadioButton;
68     QHBoxLayout *horizontalLayout;
69     QVBoxLayout *verticalLayout_2;
70     QLabel *label_3;
71     QLabel *label_4;
72     QVBoxLayout *verticalLayout;
73     QLineEdit *rightEdit;
74     QLineEdit *decEdit;
75     QLabel *raposLabel;
76     QLabel *decposLabel;
77     QWidget *layoutWidget_2;
78     QHBoxLayout *horizontalLayout_9;
79     QLabel *tracktimeLabel;
80     QTimeEdit *tracktimeEdit;
81     QPushButton *trackButton;
82     QFrame *line;
83     QGroupBox *dishBox;
84     QWidget *layoutWidget1;
85     QVBoxLayout *verticalLayout_11;
86     QHBoxLayout *horizontalLayout_6;
87     QRadioButton *D8RadioButton;
88     QLabel *ip8Label;
89     QLineEdit *ip8LineEdit;
90     QLabel *label;
91     QLineEdit *port8LineEdit;
92     QHBoxLayout *horizontalLayout_7;
93     QRadioButton *D7RadioButton;
94     QLabel *ip7Label;
```

```
95      QLineEdit *ip7LineEdit;
96      QLabel *label_2;
97      QLineEdit *port7LineEdit;
98      QHBoxLayout *horizontalLayout_8;
99      QVBoxLayout *verticalLayout_3;
100     QSpacerItem *verticalSpacer_2;
101     QRadioButton *interRadioButton;
102     QSpacerItem *verticalSpacer;
103     QVBoxLayout *verticalLayout_10;
104     QPushButton *connectButton;
105     QPushButton *disconnectButton;
106     QGroupBox *scanBox;
107     QWidget *layoutWidget2;
108     QHBoxLayout *horizontalLayout_4;
109     QLabel *runtimeLabel;
110     QTimeEdit *runtimeEdit;
111     QPushButton *scanButton;
112     QWidget *layoutWidget3;
113     QVBoxLayout *verticalLayout_9;
114     QLabel *logLabel;
115     QTextEdit *logTextEdit;
116     QFrame *line_2;
117     QWidget *layoutWidget4;
118     QVBoxLayout *verticalLayout_4;
119     QHBoxLayout *horizontalLayout_10;
120     QLabel *label_5;
121     QSpacerItem *horizontalSpacer;
122     QProgressBar *progressBar;
123     QWidget *layoutWidget5;
124     QVBoxLayout *verticalLayout_13;
125     QPushButton *stopButton;
126     QPushButton *resetButton;
127     QMenuBar *menubar;
128     QMenu *menuMenu;
129     QMenu *menuAbout;
130     QMenu *menuHelp;
131
132     void setupUi(QMainWindow *DishWindow)
133     {
134         if (DishWindow->objectName().isEmpty())
135             DishWindow->setObjectName(QString::fromUtf8("DishWindow"));
136         DishWindow->setEnabled(true);
137         DishWindow->resize(600, 660);
138         QSizePolicy sizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
139         sizePolicy.setHorizontalStretch(0);
140         sizePolicy.setVerticalStretch(0);
141         sizePolicy.setHeightForWidth(DishWindow->sizePolicy().hasHeightForWidth());
142         DishWindow->setSizePolicy(sizePolicy);
143         DishWindow->setMinimumSize(QSize(600, 660));
144         DishWindow->setMaximumSize(QSize(600, 660));
145         QIcon icon;
146         icon.addFile(QString::fromUtf8(":/img/img/satellite_256.png"), QSize(),
                QIcon::Normal, QIcon::Off);
147         DishWindow->setWindowIcon(icon);
148         actionAbout = new QAction(DishWindow);
149         actionAbout->setObjectName(QString::fromUtf8("actionAbout"));
150         actionExit = new QAction(DishWindow);
151         actionExit->setObjectName(QString::fromUtf8("actionExit"));
152         actionHelp = new QAction(DishWindow);
153         actionHelp->setObjectName(QString::fromUtf8("actionHelp"));
154         actionControllerchip_version = new QAction(DishWindow);
155         actionControllerchip_version->setObjectName(QString::fromUtf8("
                actionControllerchip_version"));
156         actionCalculate_default_scan_pattern = new QAction(DishWindow);
157         actionCalculate_default_scan_pattern->setObjectName(QString::fromUtf8("
                actionCalculate_default_scan_pattern"));
158         action_Set_default_settings = new QAction(DishWindow);
```

```
159         action_Set_default_settings->setObjectName(QString::fromUtf8("
                action_Set_default_settings"));
160         centralwidget = new QWidget(DishWindow);
161         centralwidget->setObjectName(QString::fromUtf8("centralwidget"));
162         positionBox = new QGroupBox(centralwidget);
163         positionBox->setObjectName(QString::fromUtf8("positionBox"));
164         positionBox->setGeometry(QRect(10, 160, 571, 231));
165         layoutWidget = new QWidget(positionBox);
166         layoutWidget->setObjectName(QString::fromUtf8("layoutWidget"));
167         layoutWidget->setGeometry(QRect(12, 25, 541, 141));
168         horizontalLayout_3 = new QHBoxLayout(layoutWidget);
169         horizontalLayout_3->setObjectName(QString::fromUtf8("horizontalLayout_3"));
170         horizontalLayout_3->setContentsMargins(0, 0, 0, 0);
171         verticalLayout_12 = new QVBoxLayout();
172         verticalLayout_12->setObjectName(QString::fromUtf8("verticalLayout_12"));
173         setfixButton = new QPushButton(layoutWidget);
174         setfixButton->setObjectName(QString::fromUtf8("setfixButton"));
175         sizePolicy.setHeightForWidth(setfixButton->sizePolicy().hasHeightForWidth()
                );
176         setfixButton->setSizePolicy(sizePolicy);
177         setfixButton->setMinimumSize(QSize(0, 35));
178
179         verticalLayout_12->addWidget(setfixButton);
180
181         verticalSpacer_3 = new QSpacerItem(20, 25, QSizePolicy::Minimum,
                QSizePolicy::Fixed);
182
183         verticalLayout_12->addItem(verticalSpacer_3);
184
185         posLabel = new QLabel(layoutWidget);
186         posLabel->setObjectName(QString::fromUtf8("posLabel"));
187
188         verticalLayout_12->addWidget(posLabel);
189
190         verticalSpacer_4 = new QSpacerItem(20, 5, QSizePolicy::Minimum, QSizePolicy
                ::Fixed);
191
192         verticalLayout_12->addItem(verticalSpacer_4);
193
194
195         horizontalLayout_3->addLayout(verticalLayout_12);
196
197         verticalLayout_7 = new QVBoxLayout();
198         verticalLayout_7->setObjectName(QString::fromUtf8("verticalLayout_7"));
199         degreesRadioButton = new QRadioButton(layoutWidget);
200         degreesRadioButton->setObjectName(QString::fromUtf8("degreesRadioButton"));
201         degreesRadioButton->setChecked(true);
202
203         verticalLayout_7->addWidget(degreesRadioButton);
204
205         horizontalLayout_2 = new QHBoxLayout();
206         horizontalLayout_2->setObjectName(QString::fromUtf8("horizontalLayout_2"));
207         verticalLayout_6 = new QVBoxLayout();
208         verticalLayout_6->setObjectName(QString::fromUtf8("verticalLayout_6"));
209         azLabel = new QLabel(layoutWidget);
210         azLabel->setObjectName(QString::fromUtf8("azLabel"));
211
212         verticalLayout_6->addWidget(azLabel);
213
214         elLabel = new QLabel(layoutWidget);
215         elLabel->setObjectName(QString::fromUtf8("elLabel"));
216
217         verticalLayout_6->addWidget(elLabel);
218
219
220         horizontalLayout_2->addLayout(verticalLayout_6);
221
```

```
222         verticalLayout_5 = new QVBoxLayout();
223         verticalLayout_5->setObjectName(QString::fromUtf8("verticalLayout_5"));
224         azEdit = new QLineEdit(layoutWidget);
225         azEdit->setObjectName(QString::fromUtf8("azEdit"));
226         azEdit->setEnabled(true);
227         azEdit->setInputMethodHints(Qt::ImhNone);
228
229         verticalLayout_5->addWidget(azEdit);
230
231         elEdit = new QLineEdit(layoutWidget);
232         elEdit->setObjectName(QString::fromUtf8("elEdit"));
233         elEdit->setEnabled(true);
234
235         verticalLayout_5->addWidget(elEdit);
236
237
238         horizontalLayout_2->addLayout(verticalLayout_5);
239
240
241         verticalLayout_7->addLayout(horizontalLayout_2);
242
243         azposLabel = new QLabel(layoutWidget);
244         azposLabel->setObjectName(QString::fromUtf8("azposLabel"));
245
246         verticalLayout_7->addWidget(azposLabel);
247
248         elposLabel = new QLabel(layoutWidget);
249         elposLabel->setObjectName(QString::fromUtf8("elposLabel"));
250
251         verticalLayout_7->addWidget(elposLabel);
252
253
254         horizontalLayout_3->addLayout(verticalLayout_7);
255
256         verticalLayout_8 = new QVBoxLayout();
257         verticalLayout_8->setObjectName(QString::fromUtf8("verticalLayout_8"));
258         coordRadioButton = new QRadioButton(layoutWidget);
259         coordRadioButton->setObjectName(QString::fromUtf8("coordRadioButton"));
260
261         verticalLayout_8->addWidget(coordRadioButton);
262
263         horizontalLayout = new QHBoxLayout();
264         horizontalLayout->setObjectName(QString::fromUtf8("horizontalLayout"));
265         verticalLayout_2 = new QVBoxLayout();
266         verticalLayout_2->setObjectName(QString::fromUtf8("verticalLayout_2"));
267         label_3 = new QLabel(layoutWidget);
268         label_3->setObjectName(QString::fromUtf8("label_3"));
269
270         verticalLayout_2->addWidget(label_3);
271
272         label_4 = new QLabel(layoutWidget);
273         label_4->setObjectName(QString::fromUtf8("label_4"));
274
275         verticalLayout_2->addWidget(label_4);
276
277
278         horizontalLayout->addLayout(verticalLayout_2);
279
280         verticalLayout = new QVBoxLayout();
281         verticalLayout->setObjectName(QString::fromUtf8("verticalLayout"));
282         rightEdit = new QLineEdit(layoutWidget);
283         rightEdit->setObjectName(QString::fromUtf8("rightEdit"));
284         rightEdit->setEnabled(false);
285
286         verticalLayout->addWidget(rightEdit);
287
288         decEdit = new QLineEdit(layoutWidget);
```

```
289         decEdit->setObjectName(QString::fromUtf8("decEdit"));
290         decEdit->setEnabled(false);
291
292         verticalLayout->addWidget(decEdit);
293
294
295         horizontalLayout->addLayout(verticalLayout);
296
297
298         verticalLayout_8->addLayout(horizontalLayout);
299
300         raposLabel = new QLabel(layoutWidget);
301         raposLabel->setObjectName(QString::fromUtf8("raposLabel"));
302
303         verticalLayout_8->addWidget(raposLabel);
304
305         decposLabel = new QLabel(layoutWidget);
306         decposLabel->setObjectName(QString::fromUtf8("decposLabel"));
307
308         verticalLayout_8->addWidget(decposLabel);
309
310
311         horizontalLayout_3->addLayout(verticalLayout_8);
312
313         layoutWidget_2 = new QWidget(positionBox);
314         layoutWidget_2->setObjectName(QString::fromUtf8("layoutWidget_2"));
315         layoutWidget_2->setGeometry(QRect(100, 200, 224, 22));
316         horizontalLayout_9 = new QHBoxLayout(layoutWidget_2);
317         horizontalLayout_9->setObjectName(QString::fromUtf8("horizontalLayout_9"));
318         horizontalLayout_9->setContentsMargins(0, 0, 0, 0);
319         tracktimeLabel = new QLabel(layoutWidget_2);
320         tracktimeLabel->setObjectName(QString::fromUtf8("tracktimeLabel"));
321
322         horizontalLayout_9->addWidget(tracktimeLabel);
323
324         tracktimeEdit = new QTimeEdit(layoutWidget_2);
325         tracktimeEdit->setObjectName(QString::fromUtf8("tracktimeEdit"));
326         tracktimeEdit->setMinimumDateTime(QDateTime(QDate(2000, 1, 1), QTime(0, 0,
                0)));
327         tracktimeEdit->setMaximumTime(QTime(23, 59, 59));
328         tracktimeEdit->setCurrentSection(QDateTimeEdit::HourSection);
329         tracktimeEdit->setCalendarPopup(false);
330         tracktimeEdit->setCurrentSectionIndex(0);
331         tracktimeEdit->setTimeSpec(Qt::LocalTime);
332
333         horizontalLayout_9->addWidget(tracktimeEdit);
334
335         trackButton = new QPushButton(positionBox);
336         trackButton->setObjectName(QString::fromUtf8("trackButton"));
337         trackButton->setGeometry(QRect(10, 190, 81, 35));
338         sizePolicy.setHeightForWidth(trackButton->sizePolicy().hasHeightForWidth())
                ;
339         trackButton->setSizePolicy(sizePolicy);
340         trackButton->setMinimumSize(QSize(0, 35));
341         line = new QFrame(positionBox);
342         line->setObjectName(QString::fromUtf8("line"));
343         line->setGeometry(QRect(20, 170, 521, 16));
344         line->setFrameShape(QFrame::HLine);
345         line->setFrameShadow(QFrame::Sunken);
346         layoutWidget_2->raise();
347         trackButton->raise();
348         line->raise();
349         layoutWidget->raise();
350         dishBox = new QGroupBox(centralwidget);
351         dishBox->setObjectName(QString::fromUtf8("dishBox"));
352         dishBox->setGeometry(QRect(10, 10, 341, 151));
353         layoutWidget1 = new QWidget(dishBox);
```

```
354         layoutWidget1->setObjectName(QString::fromUtf8("layoutWidget1"));
355         layoutWidget1->setGeometry(QRect(10, 21, 321, 114));
356         verticalLayout_11 = new QVBoxLayout(layoutWidget1);
357         verticalLayout_11->setObjectName(QString::fromUtf8("verticalLayout_11"));
358         verticalLayout_11->setContentsMargins(0, 0, 0, 0);
359         horizontalLayout_6 = new QHBoxLayout();
360         horizontalLayout_6->setObjectName(QString::fromUtf8("horizontalLayout_6"));
361         D8RadioButton = new QRadioButton(layoutWidget1);
362         D8RadioButton->setObjectName(QString::fromUtf8("D8RadioButton"));
363         D8RadioButton->setChecked(true);
364
365         horizontalLayout_6->addWidget(D8RadioButton);
366
367         ip8Label = new QLabel(layoutWidget1);
368         ip8Label->setObjectName(QString::fromUtf8("ip8Label"));
369
370         horizontalLayout_6->addWidget(ip8Label);
371
372         ip8LineEdit = new QLineEdit(layoutWidget1);
373         ip8LineEdit->setObjectName(QString::fromUtf8("ip8LineEdit"));
374
375         horizontalLayout_6->addWidget(ip8LineEdit);
376
377         label = new QLabel(layoutWidget1);
378         label->setObjectName(QString::fromUtf8("label"));
379
380         horizontalLayout_6->addWidget(label);
381
382         port8LineEdit = new QLineEdit(layoutWidget1);
383         port8LineEdit->setObjectName(QString::fromUtf8("port8LineEdit"));
384
385         horizontalLayout_6->addWidget(port8LineEdit);
386
387
388         verticalLayout_11->addLayout(horizontalLayout_6);
389
390         horizontalLayout_7 = new QHBoxLayout();
391         horizontalLayout_7->setObjectName(QString::fromUtf8("horizontalLayout_7"));
392         D7RadioButton = new QRadioButton(layoutWidget1);
393         D7RadioButton->setObjectName(QString::fromUtf8("D7RadioButton"));
394
395         horizontalLayout_7->addWidget(D7RadioButton);
396
397         ip7Label = new QLabel(layoutWidget1);
398         ip7Label->setObjectName(QString::fromUtf8("ip7Label"));
399
400         horizontalLayout_7->addWidget(ip7Label);
401
402         ip7LineEdit = new QLineEdit(layoutWidget1);
403         ip7LineEdit->setObjectName(QString::fromUtf8("ip7LineEdit"));
404
405         horizontalLayout_7->addWidget(ip7LineEdit);
406
407         label_2 = new QLabel(layoutWidget1);
408         label_2->setObjectName(QString::fromUtf8("label_2"));
409
410         horizontalLayout_7->addWidget(label_2);
411
412         port7LineEdit = new QLineEdit(layoutWidget1);
413         port7LineEdit->setObjectName(QString::fromUtf8("port7LineEdit"));
414
415         horizontalLayout_7->addWidget(port7LineEdit);
416
417
418         verticalLayout_11->addLayout(horizontalLayout_7);
419
420         horizontalLayout_8 = new QHBoxLayout();
```

```
421         horizontalLayout_8->setObjectName(QString::fromUtf8("horizontalLayout_8"));
422         verticalLayout_3 = new QVBoxLayout();
423         verticalLayout_3->setObjectName(QString::fromUtf8("verticalLayout_3"));
424         verticalSpacer_2 = new QSpacerItem(20, 28, QSizePolicy::Minimum,
                QSizePolicy::Expanding);
425
426         verticalLayout_3->addItem(verticalSpacer_2);
427
428         interRadioButton = new QRadioButton(layoutWidget1);
429         interRadioButton->setObjectName(QString::fromUtf8("interRadioButton"));
430         interRadioButton->setEnabled(true);
431
432         verticalLayout_3->addWidget(interRadioButton);
433
434         verticalSpacer = new QSpacerItem(20, 28, QSizePolicy::Minimum, QSizePolicy
                ::Expanding);
435
436         verticalLayout_3->addItem(verticalSpacer);
437
438
439         horizontalLayout_8->addLayout(verticalLayout_3);
440
441         verticalLayout_10 = new QVBoxLayout();
442         verticalLayout_10->setObjectName(QString::fromUtf8("verticalLayout_10"));
443         connectButton = new QPushButton(layoutWidget1);
444         connectButton->setObjectName(QString::fromUtf8("connectButton"));
445
446         verticalLayout_10->addWidget(connectButton);
447
448         disconnectButton = new QPushButton(layoutWidget1);
449         disconnectButton->setObjectName(QString::fromUtf8("disconnectButton"));
450         disconnectButton->setEnabled(false);
451
452         verticalLayout_10->addWidget(disconnectButton);
453
454
455         horizontalLayout_8->addLayout(verticalLayout_10);
456
457
458         verticalLayout_11->addLayout(horizontalLayout_8);
459
460         scanBox = new QGroupBox(centralwidget);
461         scanBox->setObjectName(QString::fromUtf8("scanBox"));
462         scanBox->setGeometry(QRect(300, 410, 247, 102));
463         layoutWidget2 = new QWidget(scanBox);
464         layoutWidget2->setObjectName(QString::fromUtf8("layoutWidget2"));
465         layoutWidget2->setGeometry(QRect(10, 60, 227, 22));
466         horizontalLayout_4 = new QHBoxLayout(layoutWidget2);
467         horizontalLayout_4->setObjectName(QString::fromUtf8("horizontalLayout_4"));
468         horizontalLayout_4->setContentsMargins(0, 0, 0, 0);
469         runtimeLabel = new QLabel(layoutWidget2);
470         runtimeLabel->setObjectName(QString::fromUtf8("runtimeLabel"));
471
472         horizontalLayout_4->addWidget(runtimeLabel);
473
474         runtimeEdit = new QTimeEdit(layoutWidget2);
475         runtimeEdit->setObjectName(QString::fromUtf8("runtimeEdit"));
476         runtimeEdit->setMinimumDateTime(QDateTime(QDate(2000, 1, 1), QTime(0, 0, 0)
                ));
477         runtimeEdit->setMaximumTime(QTime(23, 59, 59));
478         runtimeEdit->setCurrentSection(QDateTimeEdit::HourSection);
479         runtimeEdit->setCalendarPopup(false);
480         runtimeEdit->setCurrentSectionIndex(0);
481         runtimeEdit->setTimeSpec(Qt::LocalTime);
482
483         horizontalLayout_4->addWidget(runtimeEdit);
484
```

```
485         scanButton = new QPushButton(scanBox);
486         scanButton->setObjectName(QString::fromUtf8("scanButton"));
487         scanButton->setGeometry(QRect(10, 20, 101, 31));
488         sizePolicy.setHeightForWidth(scanButton->sizePolicy().hasHeightForWidth());
489         scanButton->setSizePolicy(sizePolicy);
490         layoutWidget3 = new QWidget(centralwidget);
491         layoutWidget3->setObjectName(QString::fromUtf8("layoutWidget3"));
492         layoutWidget3->setGeometry(QRect(10, 400, 271, 221));
493         verticalLayout_9 = new QVBoxLayout(layoutWidget3);
494         verticalLayout_9->setObjectName(QString::fromUtf8("verticalLayout_9"));
495         verticalLayout_9->setContentsMargins(0, 0, 0, 0);
496         logLabel = new QLabel(layoutWidget3);
497         logLabel->setObjectName(QString::fromUtf8("logLabel"));
498
499         verticalLayout_9->addWidget(logLabel);
500
501         logTextEdit = new QTextEdit(layoutWidget3);
502         logTextEdit->setObjectName(QString::fromUtf8("logTextEdit"));
503         logTextEdit->setAutoFillBackground(false);
504         logTextEdit->setUndoRedoEnabled(false);
505         logTextEdit->setAcceptRichText(false);
506         logTextEdit->setTextInteractionFlags(Qt::TextSelectableByMouse);
507
508         verticalLayout_9->addWidget(logTextEdit);
509
510         line_2 = new QFrame(centralwidget);
511         line_2->setObjectName(QString::fromUtf8("line_2"));
512         line_2->setGeometry(QRect(320, 540, 211, 16));
513         line_2->setFrameShape(QFrame::HLine);
514         line_2->setFrameShadow(QFrame::Sunken);
515         layoutWidget4 = new QWidget(centralwidget);
516         layoutWidget4->setObjectName(QString::fromUtf8("layoutWidget4"));
517         layoutWidget4->setGeometry(QRect(310, 570, 268, 51));
518         verticalLayout_4 = new QVBoxLayout(layoutWidget4);
519         verticalLayout_4->setObjectName(QString::fromUtf8("verticalLayout_4"));
520         verticalLayout_4->setContentsMargins(0, 0, 0, 0);
521         horizontalLayout_10 = new QHBoxLayout();
522         horizontalLayout_10->setObjectName(QString::fromUtf8("horizontalLayout_10")
                );
523         label_5 = new QLabel(layoutWidget4);
524         label_5->setObjectName(QString::fromUtf8("label_5"));
525
526         horizontalLayout_10->addWidget(label_5);
527
528         horizontalSpacer = new QSpacerItem(118, 20, QSizePolicy::Expanding,
                QSizePolicy::Minimum);
529
530         horizontalLayout_10->addItem(horizontalSpacer);
531
532
533         verticalLayout_4->addLayout(horizontalLayout_10);
534
535         progressBar = new QProgressBar(layoutWidget4);
536         progressBar->setObjectName(QString::fromUtf8("progressBar"));
537         progressBar->setEnabled(false);
538         progressBar->setValue(0);
539
540         verticalLayout_4->addWidget(progressBar);
541
542         layoutWidget5 = new QWidget(centralwidget);
543         layoutWidget5->setObjectName(QString::fromUtf8("layoutWidget5"));
544         layoutWidget5->setGeometry(QRect(420, 50, 101, 81));
545         verticalLayout_13 = new QVBoxLayout(layoutWidget5);
546         verticalLayout_13->setObjectName(QString::fromUtf8("verticalLayout_13"));
547         verticalLayout_13->setContentsMargins(0, 0, 0, 0);
548         stopButton = new QPushButton(layoutWidget5);
549         stopButton->setObjectName(QString::fromUtf8("stopButton"));
```

```
550         QSizePolicy sizePolicy1(QSizePolicy::Minimum, QSizePolicy::Minimum);
551         sizePolicy1.setHorizontalStretch(0);
552         sizePolicy1.setVerticalStretch(0);
553         sizePolicy1.setHeightForWidth(stopButton->sizePolicy().hasHeightForWidth())
                ;
554         stopButton->setSizePolicy(sizePolicy1);
555
556         verticalLayout_13->addWidget(stopButton);
557
558         resetButton = new QPushButton(layoutWidget5);
559         resetButton->setObjectName(QString::fromUtf8("resetButton"));
560         sizePolicy1.setHeightForWidth(resetButton->sizePolicy().hasHeightForWidth()
                );
561         resetButton->setSizePolicy(sizePolicy1);
562
563         verticalLayout_13->addWidget(resetButton);
564
565         DishWindow->setCentralWidget(centralwidget);
566         menubar = new QMenuBar(DishWindow);
567         menubar->setObjectName(QString::fromUtf8("menubar"));
568         menubar->setGeometry(QRect(0, 0, 600, 21));
569         menuMenu = new QMenu(menubar);
570         menuMenu->setObjectName(QString::fromUtf8("menuMenu"));
571         menuAbout = new QMenu(menubar);
572         menuAbout->setObjectName(QString::fromUtf8("menuAbout"));
573         menuHelp = new QMenu(menubar);
574         menuHelp->setObjectName(QString::fromUtf8("menuHelp"));
575         DishWindow->setMenuBar(menubar);
576         QWidget::setTabOrder(degreesRadioButton, azEdit);
577         QWidget::setTabOrder(azEdit, elEdit);
578         QWidget::setTabOrder(elEdit, coordRadioButton);
579         QWidget::setTabOrder(coordRadioButton, rightEdit);
580         QWidget::setTabOrder(rightEdit, decEdit);
581         QWidget::setTabOrder(decEdit, D8RadioButton);
582         QWidget::setTabOrder(D8RadioButton, D7RadioButton);
583         QWidget::setTabOrder(D7RadioButton, connectButton);
584         QWidget::setTabOrder(connectButton, scanButton);
585
586         menubar->addAction(menuMenu->menuAction());
587         menubar->addAction(menuHelp->menuAction());
588         menubar->addAction(menuAbout->menuAction());
589         menuMenu->addAction(actionCalculate_default_scan_pattern);
590         menuMenu->addAction(action_Set_default_settings);
591         menuMenu->addSeparator();
592         menuMenu->addAction(actionExit);
593         menuAbout->addAction(actionAbout);
594         menuAbout->addAction(actionControllerchip_version);
595         menuHelp->addAction(actionHelp);
596
597         retranslateUi(DishWindow);
598         QObject::connect(actionExit, SIGNAL(triggered()), DishWindow, SLOT(close())
                );
599         QObject::connect(actionAbout, SIGNAL(triggered()), DishWindow, SLOT(about()
                ));
600         QObject::connect(scanButton, SIGNAL(clicked()), DishWindow, SLOT(scan()));
601         QObject::connect(stopButton, SIGNAL(clicked()), DishWindow, SLOT(stop()));
602         QObject::connect(resetButton, SIGNAL(clicked()), DishWindow, SLOT(reset()))
                ;
603         QObject::connect(connectButton, SIGNAL(clicked()), DishWindow, SLOT(
                checkMode()));
604         QObject::connect(disconnectButton, SIGNAL(clicked()), DishWindow, SLOT(
                checkDisconnect()));
605         QObject::connect(trackButton, SIGNAL(clicked()), DishWindow, SLOT(calcTrack
                ()));
606         QObject::connect(actionCalculate_default_scan_pattern, SIGNAL(triggered()),
                 DishWindow, SLOT(calcScan()));
```

```
607         QObject::connect(actionHelp, SIGNAL(triggered()), DishWindow, SLOT(help()))
                ;
608         QObject::connect(actionControllerchip_version, SIGNAL(triggered()),
                DishWindow, SLOT(version()));
609         QObject::connect(action_Set_default_settings, SIGNAL(triggered()),
                DishWindow, SLOT(writeDefaultINI()));
610
611         QMetaObject::connectSlotsByName(DishWindow);
612     } // setupUi
613
614     void retranslateUi(QMainWindow *DishWindow)
615     {
616         DishWindow->setWindowTitle(QApplication::translate("DishWindow", "\303\205
                ngstr\303\266m Synthesis Radio Telescope (\303\205SRT) - Dish
                Controller", 0, QApplication::UnicodeUTF8));
617         actionAbout->setText(QApplication::translate("DishWindow", "&About \303\205
                SRT...", 0, QApplication::UnicodeUTF8));
618         actionExit->setText(QApplication::translate("DishWindow", "E&xit", 0,
                QApplication::UnicodeUTF8));
619         actionHelp->setText(QApplication::translate("DishWindow", "&Help", 0,
                QApplication::UnicodeUTF8));
620         actionHelp->setShortcut(QApplication::translate("DishWindow", "F1", 0,
                QApplication::UnicodeUTF8));
621         actionControllerchip_version->setText(QApplication::translate("DishWindow",
                "Controllerchip &version", 0, QApplication::UnicodeUTF8));
622         actionCalculate_default_scan_pattern->setText(QApplication::translate("
                DishWindow", "&Set default scanning pattern", 0, QApplication::
                UnicodeUTF8));
623         action_Set_default_settings->setText(QApplication::translate("DishWindow",
                "&Set default settings to ini file", 0, QApplication::UnicodeUTF8));
624         positionBox->setTitle(QApplication::translate("DishWindow", "Fix Position",
                 0, QApplication::UnicodeUTF8));
625         setfixButton->setText(QApplication::translate("DishWindow", "Set Fix Point"
                , 0, QApplication::UnicodeUTF8));
626         posLabel->setText(QApplication::translate("DishWindow", "Current Position:"
                , 0, QApplication::UnicodeUTF8));
627         degreesRadioButton->setText(QApplication::translate("DishWindow", "
                Horizontal Coordinates", 0, QApplication::UnicodeUTF8));
628         azLabel->setText(QApplication::translate("DishWindow", "Azimuth [deg]:", 0,
                 QApplication::UnicodeUTF8));
629         elLabel->setText(QApplication::translate("DishWindow", "Elevation [deg]:",
                0, QApplication::UnicodeUTF8));
630         azEdit->setInputMask(QString());
631         azEdit->setText(QApplication::translate("DishWindow", "0", 0, QApplication
                ::UnicodeUTF8));
632         elEdit->setInputMask(QString());
633         elEdit->setText(QApplication::translate("DishWindow", "0", 0, QApplication
                ::UnicodeUTF8));
634         azposLabel->setText(QApplication::translate("DishWindow", "Azimuth =\n"
635 "", 0, QApplication::UnicodeUTF8));
636         elposLabel->setText(QApplication::translate("DishWindow", "Elevation =\n"
637 "", 0, QApplication::UnicodeUTF8));
638         coordRadioButton->setText(QApplication::translate("DishWindow", "Equatorial
                Coordinates", 0, QApplication::UnicodeUTF8));
639         label_3->setText(QApplication::translate("DishWindow", "Right Ascension [hh
                :mm:ss]:", 0, QApplication::UnicodeUTF8));
640         label_4->setText(QApplication::translate("DishWindow", "Declination [deg:mm
                :ss]:", 0, QApplication::UnicodeUTF8));
641         rightEdit->setText(QApplication::translate("DishWindow", "0:42:44.3", 0,
                QApplication::UnicodeUTF8));
642         decEdit->setText(QApplication::translate("DishWindow", "41:16:9", 0,
                QApplication::UnicodeUTF8));
643         raposLabel->setText(QApplication::translate("DishWindow", "Right Ascension
                =\n"
644 "", 0, QApplication::UnicodeUTF8));
645         decposLabel->setText(QApplication::translate("DishWindow", "Declination =\
                n"
```

```
646 "", 0, QApplication::UnicodeUTF8));
647         tracktimeLabel->setText(QApplication::translate("DishWindow", "Total␣
                tracking␣time␣[hh:mm:ss]", 0, QApplication::UnicodeUTF8));
648         trackButton->setText(QApplication::translate("DishWindow", "Track\n"
649 "␣Fix␣Point", 0, QApplication::UnicodeUTF8));
650         dishBox->setTitle(QApplication::translate("DishWindow", "Dish␣Mode", 0,
                QApplication::UnicodeUTF8));
651         D8RadioButton->setText(QApplication::translate("DishWindow", "Dish␣
                \303\2058", 0, QApplication::UnicodeUTF8));
652         ip8Label->setText(QApplication::translate("DishWindow", "IP:", 0,
                QApplication::UnicodeUTF8));
653         ip8LineEdit->setText(QApplication::translate("DishWindow", "130.238.30.234"
                , 0, QApplication::UnicodeUTF8));
654         label->setText(QApplication::translate("DishWindow", "Port:", 0,
                QApplication::UnicodeUTF8));
655         port8LineEdit->setText(QApplication::translate("DishWindow", "5001", 0,
                QApplication::UnicodeUTF8));
656         D7RadioButton->setText(QApplication::translate("DishWindow", "Dish␣
                \303\2057", 0, QApplication::UnicodeUTF8));
657         ip7Label->setText(QApplication::translate("DishWindow", "IP:", 0,
                QApplication::UnicodeUTF8));
658         ip7LineEdit->setText(QApplication::translate("DishWindow", "127.0.0.1", 0,
                QApplication::UnicodeUTF8));
659         label_2->setText(QApplication::translate("DishWindow", "Port:", 0,
                QApplication::UnicodeUTF8));
660         port7LineEdit->setText(QApplication::translate("DishWindow", "6001", 0,
                QApplication::UnicodeUTF8));
661         interRadioButton->setText(QApplication::translate("DishWindow", "
                Interferometric", 0, QApplication::UnicodeUTF8));
662         connectButton->setText(QApplication::translate("DishWindow", "Connect", 0,
                QApplication::UnicodeUTF8));
663         disconnectButton->setText(QApplication::translate("DishWindow", "Disconnect
                ", 0, QApplication::UnicodeUTF8));
664         scanBox->setTitle(QApplication::translate("DishWindow", "Hemispherical␣Scan
                ", 0, QApplication::UnicodeUTF8));
665         runtimeLabel->setText(QApplication::translate("DishWindow", "Total␣scanning
                ␣time␣[hh:mm:ss]", 0, QApplication::UnicodeUTF8));
666         scanButton->setText(QApplication::translate("DishWindow", "Start␣Scan", 0,
                QApplication::UnicodeUTF8));
667         logLabel->setText(QApplication::translate("DishWindow", "Controller␣log",
                0, QApplication::UnicodeUTF8));
668         label_5->setText(QApplication::translate("DishWindow", "Scanning␣&␣Tracking
                ␣Progress", 0, QApplication::UnicodeUTF8));
669         progressBar->setFormat(QApplication::translate("DishWindow", "%p%", 0,
                QApplication::UnicodeUTF8));
670         stopButton->setText(QApplication::translate("DishWindow", "Stop", 0,
                QApplication::UnicodeUTF8));
671         resetButton->setText(QApplication::translate("DishWindow", "Reset␣Positon",
                 0, QApplication::UnicodeUTF8));
672         menuMenu->setTitle(QApplication::translate("DishWindow", "&Menu", 0,
                QApplication::UnicodeUTF8));
673         menuAbout->setTitle(QApplication::translate("DishWindow", "&About", 0,
                QApplication::UnicodeUTF8));
674         menuHelp->setTitle(QApplication::translate("DishWindow", "&Help", 0,
                QApplication::UnicodeUTF8));
675     } // retranslateUi
676
677 };
678
679 namespace Ui {
680     class DishWindow: public Ui_DishWindow {};
681 } // namespace Ui
682
683 QT_END_NAMESPACE
684
685 #endif // UI_DISHWINDOW_H
```

C.7 ui_helpbrowser.h

```
1  /********************************************************************************
2  ** Form generated from reading UI file 'helpbrowser.ui'
3  **
4  ** Created: Wed 3. Nov 14:32:01 2010
5  **      by: Qt User Interface Compiler version 4.6.3
6  **
7  ** WARNING! All changes made in this file will be lost when recompiling UI file!
8  ********************************************************************************/
9
10 #ifndef UI_HELPBROWSER_H
11 #define UI_HELPBROWSER_H
12
13 #include <QtCore/QVariant>
14 #include <QtGui/QAction>
15 #include <QtGui/QApplication>
16 #include <QtGui/QButtonGroup>
17 #include <QtGui/QHBoxLayout>
18 #include <QtGui/QHeaderView>
19 #include <QtGui/QPushButton>
20 #include <QtGui/QSpacerItem>
21 #include <QtGui/QTextBrowser>
22 #include <QtGui/QVBoxLayout>
23 #include <QtGui/QWidget>
24
25 QT_BEGIN_NAMESPACE
26
27 class Ui_Helpwindow
28 {
29 public:
30     QWidget *layoutWidget;
31     QVBoxLayout *verticalLayout;
32     QHBoxLayout *horizontalLayout;
33     QPushButton *homeButton;
34     QPushButton *backButton;
35     QSpacerItem *horizontalSpacer;
36     QPushButton *closeButton;
37     QTextBrowser *textBrowser;
38
39     void setupUi(QWidget *Helpwindow)
40     {
41         if (Helpwindow->objectName().isEmpty())
42             Helpwindow->setObjectName(QString::fromUtf8("Helpwindow"));
43         Helpwindow->resize(450, 500);
44         QSizePolicy sizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
45         sizePolicy.setHorizontalStretch(0);
46         sizePolicy.setVerticalStretch(0);
47         sizePolicy.setHeightForWidth(Helpwindow->sizePolicy().hasHeightForWidth());
48         Helpwindow->setSizePolicy(sizePolicy);
49         Helpwindow->setMinimumSize(QSize(450, 500));
50         Helpwindow->setMaximumSize(QSize(450, 500));
51         layoutWidget = new QWidget(Helpwindow);
52         layoutWidget->setObjectName(QString::fromUtf8("layoutWidget"));
53         layoutWidget->setGeometry(QRect(10, 11, 431, 481));
54         verticalLayout = new QVBoxLayout(layoutWidget);
55         verticalLayout->setObjectName(QString::fromUtf8("verticalLayout"));
56         verticalLayout->setContentsMargins(0, 0, 0, 0);
57         horizontalLayout = new QHBoxLayout();
58         horizontalLayout->setObjectName(QString::fromUtf8("horizontalLayout"));
59         homeButton = new QPushButton(layoutWidget);
60         homeButton->setObjectName(QString::fromUtf8("homeButton"));
61
62         horizontalLayout->addWidget(homeButton);
63
64         backButton = new QPushButton(layoutWidget);
65         backButton->setObjectName(QString::fromUtf8("backButton"));
```

```
66
67          horizontalLayout->addWidget(backButton);
68
69          horizontalSpacer = new QSpacerItem(170, 20, QSizePolicy::Fixed, QSizePolicy
                ::Minimum);
70
71          horizontalLayout->addItem(horizontalSpacer);
72
73          closeButton = new QPushButton(layoutWidget);
74          closeButton->setObjectName(QString::fromUtf8("closeButton"));
75
76          horizontalLayout->addWidget(closeButton);
77
78
79          verticalLayout->addLayout(horizontalLayout);
80
81          textBrowser = new QTextBrowser(layoutWidget);
82          textBrowser->setObjectName(QString::fromUtf8("textBrowser"));
83          sizePolicy.setHeightForWidth(textBrowser->sizePolicy().hasHeightForWidth())
                ;
84          textBrowser->setSizePolicy(sizePolicy);
85
86          verticalLayout->addWidget(textBrowser);
87
88
89          retranslateUi(Helpwindow);
90          QObject::connect(homeButton, SIGNAL(clicked()), Helpwindow, SLOT(home()));
91          QObject::connect(backButton, SIGNAL(clicked()), Helpwindow, SLOT(backward()
                ));
92          QObject::connect(closeButton, SIGNAL(clicked()), Helpwindow, SLOT(close()))
                ;
93
94          QMetaObject::connectSlotsByName(Helpwindow);
95      } // setupUi
96
97      void retranslateUi(QWidget *Helpwindow)
98      {
99          Helpwindow->setWindowTitle(QApplication::translate("Helpwindow", "\303\205
                SRT - Help", 0, QApplication::UnicodeUTF8));
100         homeButton->setText(QApplication::translate("Helpwindow", "&Home", 0,
                QApplication::UnicodeUTF8));
101         backButton->setText(QApplication::translate("Helpwindow", "&Back", 0,
                QApplication::UnicodeUTF8));
102         closeButton->setText(QApplication::translate("Helpwindow", "&Close", 0,
                QApplication::UnicodeUTF8));
103     } // retranslateUi
104
105 };
106
107 namespace Ui {
108     class Helpwindow: public Ui_Helpwindow {};
109 } // namespace Ui
110
111 QT_END_NAMESPACE
112
113 #endif // UI_HELPBROWSER_H
```

C.8 config.ini

```
1 [communication]
2 ipadress8=130.238.30.234
3 port8=5001
4 ipadress7=130.238.30.200
5 port7=5001
6
```

```
7  [position]
8  azimuth=0
9  elevation=0
10 right_ascension=0:42:44.3
11 declination=41:16:9
12
13 [settings]
14 beam_width=7
```

Bibliography

[1] Onsala Space Observatory, *http://www.chalmers.se/rss/oso-en/*

[2] MIT Haystack Observatory, *http://www.haystack.mit.edu/edu/undergrad/srt/*

[3] MIT Haystack Observatory - Antenna specifications, *http://www.haystack.mit.edu/edu/undergrad/srt/antenna/antenna_info.html/*

[4] J.R. Wertz & W.J. Larson, *Space Mission Analysis and Design 3rd Edition*, p.571, Space Technology Library, New York, 1999.

[5] John D.Kraus, *Radio Astronomy, 2nd Edition*, Cygnus-Quasar Books, 1986.

[6] SALSA Onsala – Such A Lovely Small Antenna, *http://www.chalmers.se/rss/oso-en/observations/2-3-m-lab-antenna-salsa/*

[7] Arecibo Radio Telescope, *http://www.naic.edu/*

[8] SETI Institute, *http://www.seti.org/*

[9] Max Planck Institute for Radio Astronomy - Radio Telescope Effelsberg, *http://www.mpifr.de/english/radiotelescope/index.html/*

[10] SpaceDaily - China To Build World's Largest Radio Telescope, *http://www.spacedaily.com/reports/China_To_Build_World_Largest_Radio_Telescope_999.html/*

[11] R. Nan, B. Peng (2002) Kilometer-square Area Radio Synthesis Telescope-KARST, *http://www.skatelescope.org/uploaded/8481_17_memo_Nan.pdf*

[12] LOFAR, *http://www.lofar.org/astronomy/eor-ksp/redshifted-21cm-hydrogen-line/redshifted-21cm-hydrogen-line/*

[13] C.L. Carilli, D.E. Harris, *Cygnus A - Study of a radio galaxy*, p. 88, University Press, Cambridge 1996.

[14] National Centre for Radio Astrophysics - Pune University, India, *http://gmrt.ncra.tifr.res.in/gmrt_hpage/Users/doc/WEBLF/LFRA/node20.html/*

[15] GTK+, *http://www.gtk.org/*

[16] Qt Development Frameworks, *http://qt.nokia.com/*

[17] Jasmin Blanchette, Mark Summerfield, *C++ GUI Programming with Qt 4, Second Edition*, Prentice Hall, February 04, 2008.

[18] Astronomy Eduction at the University of Nebraska-Lincoln - Celestial Equatorial Coordinate System, *http://astro.unl.edu/naap/motion1/cec_units.html/*

[19] David M. Pozar, *Microwave Engineering 3rd Edition*, p. 496, John Wiley & Sons, 2005.

[20] Committee on Radio Astronomy Frequencies - Status of interference problems in Sweden, *http://www.craf.eu/swe.htm#21c/*

[21] David M. Pozar, *Microwave Engineering 3rd Edition*, p. 617, John Wiley & Sons, 2005.

[22] Mike Curtin and Paul O'Brien, *Phase Locked Loops for High-Frequency Receivers and Transmitters-3*, Analog Dialogue – Volume 33, Number 7, July/August, 1999 *http://www.analog.com/library/analogDialogue/archives/33-07/phase3/index.html/*

[23] Microchip Technology Inc., *http://www.microchip.com/*

[24] SSB-Electronics, *http://www.ssb.de/pdfs/6060_ Aircom%20Plus_ en.pdf*

[25] Hangzhou Hongsen Cable Co. - RG402U Data Sheet, *http://www.hongsencable.com/pdf%5C20154256.pdf*

[26] Paul McCormack, National Semiconductor Corporation, *Effects and Benefits of Undersampling in High-Speed ADC Applications*, Design & Elektronik, Germany, May 24, 2004.

[27] Jim Lesurf, Radio & Coherent Techniques, Part 8 - Sky Noise, Fig. 8.5 *http://www.st-andrews.ac.uk/ www_ pa/Scots_ Guide/RadCom/part8/page3.html/*

[28] LOIS receiver software - Sensor GUI, *http://www.lois-space.net/software.html/*

[29] CelesTrak - NORAD Two-Line Element Sets, *http://celestrak.com/NORAD/elements/*

[30] Radio Astronomy Supplies, *http://www.nitehawk.com/rasmit/jml0.html/*

[31] Mini-Circuits, *http://www.minicircuits.com/*