Space Plasma Environment and Radio Science Group

Department of Communication Systems

Lancaster University

United Kingdom

Technological Advances in Imaging Riometry

Martin Grill, Dipl.-Ing. (FH)

Submitted in part fulfilment of the requirements

for the degree of Doctor of Philosophy

June 2007

Abstract

Technological Advances in Imaging Riometry

Martin Grill, Dipl.-Ing. (FH)

Submitted for the degree of Doctor of Philosophy

June 2007

Since their inception in 1935, relative ionospheric opacity meters (riometers) have evolved through several generations, from simple, manually operated, widebeam receivers to automated multi-beam imaging riometers. This thesis follows the development of a new type of imaging riometer based on a Mills Cross antenna array: the Advanced Rio-Imaging Experiment in Scandinavia (ARIES). This is the first time that a digital cross-correlation beamforming technique has been used in riometry. The investigations presented cover initial simulations, software specification, design and implementation, hardware prototyping, the working instrument and first data products. Therefore, the work is an interdisciplinary slice through the engineering cycle of ARIES, encompassing a variety of subject areas, including (Space Plasma) Physics, Electronics, Astronomy and (Software) Engineering.

This thesis makes three specific contributions. Firstly, several low- and high-level simulations are presented, culminating in the development of the riometer simulation toolkit (RIOSIM). RIOSIM is not specific to ARIES, but is capable of simulating the behaviour of arbitrary antenna arrays.

Secondly, a flexible software architecture is developed and implemented in form of the Advanced Riometer Components (ARCOM) operating software. ARCOM consists of components participating in processing pipelines through high-speed shared memory interfaces and a flexible streaming data format based around principles similar to those used in digital video broadcasting (DVB). ARCOM software architecture and data formats are not limited to riometry but will readily support a wide range of data acquisition and processing tasks.

Thirdly, a new approach to image interpolation for riometers (GLEAM) is developed and its performance evaluated. GLEAM uses a matrix inversion technique, combined with knowledge about antenna and phased array directivity patterns and predictions obtained from simulations, to fit a parametrised model of the sky brightness distribution to real data. This is envisaged to become the primary data product of a new generation of riometers.

Declaration

The material presented in this thesis is the result of my own work and has not been produced as the result of collaborations, except where specifically indicated. The material has not been submitted in substantially the same form for the award of a higher degree elsewhere.

Some of the ideas in this thesis stem from discussions with my supervisors and colleagues, and these have been referenced as ‘personal communication’ where appropriate.

Some of the material presented in this thesis has been, or is in the process of being, published in journals (Hagfors, Grill, Honary 2003 — [HGH03]; Nielsen, Honary, Grill 2004 — [NHG04]; Honary, Grill, Barratt, Nielsen 2007 — [HGBC07]) and conference presentations (Grill, Honary, Nielsen et al. 2003 — [GHN+03]; Honary, Marple, Kosch, Senior, Makarevitch, Grill 2004 — [HMK+04]; Barratt, Honary, Grill, Chapman 2004 — [BHGC04]; Grill, Senior, Honary 2005 — [GSH05]; Honary, Chapman, Grill, Barratt et al. 2005 — [HCG+05]; Honary, Chapman, Grill, Barratt et al. 2006 — [HCG+06]).

Martin Grill

Dedication

“To him who is able to do immeasurably more than all we ask or imagine, according to his power that is at work within us.” [God]

Acknowledgements

While working on this thesis, numerous people have supported and influenced me in many different ways. Mentioning them all seems even more of a 'Wicked Problem' than designing operating software for ARIES. So I shall ask forgiveness from all those people I have missed out on the following list and from those whose contributions I have misrepresented or misjudged.

Stefani Maier and Haybatolah Khakzar have helped establish the contact with Lancaster University in the first place. I thank my supervisor Farideh Honary for giving me the opportunity to write this thesis and supporting me all the way.

Erling Nielsen and the late Tor Hagfors are the originators of the idea to build a riometer based around a Mills Cross antenna array.

Hisao Yamagishi suggested a valuable practical approach for integration time estimation.

Mike Rietveld and staff at the EISCAT facility near Tromsø have helped out with advice and equipment on various occasions, both during our on-site campaigns in Norway and during remote operation.

As well as being a brilliant role model of a scientist and the originator of the ideas behind chapter 10, Andrew Senior has often supported me with advice, comments and uniquely insightful, ethical and diplomatic points of view. He has also provided me with much-needed extra time by taking over some of my duties at various times.

Throughout several projects, including ARIES, Keith Barratt has always been a pleasure to work with and an invaluable source of advice. He also shared my passion for well-designed systems and interfaces.

Peter Chapman was the source of many interesting insights into RF design and applied electronics.

The support and friendship of all members of the Space Plasma Environment and Radio Science (SPEARS) group has always been a great encouragement and especially Roman Makarevitch has become a good friend.

The original ARIES project was funded by the Particle Physics and Astronomy Research Council (PPARC), which has since been merged into the Science and Technology Facilities Council (STFC). Without funding for the ARIES project, this thesis would not have been possible.

I am deeply grateful that my parents Wolfgang and Sibylle Grill and my sister Claudia Grill have been a never ending source of love and support.

Martin and Claudia Strohbach have become dear friends and have provided valuable advice, support and my first godchild!

My church family at Lancaster Baptist Church and beyond, and particularly our housegroup, have been a tremendous source of strength, support and advice.

Finally, the biggest ‘thank you’ of all goes to my wife Ruth and my son Ken Lukas. The very thought of them has the power to fill every moment of my life with joy and happiness, also and especially whenever work and research are not as joyful as they should be.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

New technologies enable us to probe as-yet unknown areas of creation. Today's complex systems are made up of a multiplicity of mechanical, electrical and electronic components, with the intangible 'ether' of software causing dead matter to rise beyond its pure existence and assist us in our quest of exploration. It is this fascination with making things work, interact and fulfil a higher purpose that drives an engineer.

This thesis follows the development of a new type of imaging riometer based around the principle of a Mills Cross cross-correlating antenna array: the Advanced Rio-Imaging Experiment in Scandinavia (ARIES). This is the first time that a digital cross-correlation beamforming technique has been used in riometry.

The author's background in Mechatronics results in this thesis being an interdisciplinary slice through the engineering cycle of ARIES, encompassing a variety of subject areas including (Space Plasma) Physics, Electronics, Astronomy and (Software) Engineering.

Several of the concepts presented in this thesis have since been applied to, or used during the development of, other scientific instruments. These include the Advanced Imaging Riometer for Ionospheric Studies (AIRIS), which uses the ARCOM operating software, and the new high-speed photometer for optical emission measurements (SPARKLE), which employs a packet-based streaming data format very similar to the one presented in this thesis.

1.1 Main Contributions

This thesis makes three specific contributions. *Firstly*, several low- and high-level simulations are presented, culminating in the development of the riometer simulation toolkit (RIOSIM).

RIOSIM is not specific to ARIES, but is capable of simulating the behaviour of arbitrary antenna arrays.

Secondly, a flexible software architecture is developed and implemented in form of the Advanced Riometer Components (ARCOM) operating software. ARCOM consists of components participating in processing pipelines through high-speed shared memory interfaces and a flexible streaming data format based around principles similar to those used in digital video broadcasting (DVB). ARCOM software architecture and data formats are not limited to riometry but will readily support a wide range of data acquisition and processing tasks.

Thirdly, a new approach to image interpolation for riometers (GLEAM) is developed and its performance evaluated. GLEAM uses a matrix inversion technique, combined with knowledge about antenna and phased array directivity patterns and predictions obtained from simulations, to fit a parametrised model of the sky brightness distribution to real data. This is envisaged to become the primary data product of a new generation of riometers.

1.2 Brief Description of All Chapters

The sequence of chapters follows the logical development cycle of the riometer, from inital concepts through low- and high-level simulations, software specification, design and implementation, hardware prototyping and testing to a sophisticated piece of scientific measuring equipment and analyses of real data recorded by this brand-new instrument. Each chapter builds on the preceding ones. Chapters 2 and 3 form the background part of the thesis, the subsequent chapters address the author's specific contributions. Additional information to complement the material presented in the main body of the thesis is provided in several appendices, including technical documentation of certain tools as well as information collated from third-party documentation that is not the author's own work, or only distantly related to the main topic of this thesis.

Antennas are essential in any device that receives or transmits radio waves. **Chapter 2** looks at properties of individual antennas as well as of arrays of antennas. This general description forms the basis for the more specific discussion of our area of interest, riometry, and how antennas are used in riometers, in chapter 3. A large part of chapter 2 is dedicated to the discussion of various aspects of the Mills Cross as used by the Advanced Rio-Imaging Experiment in Scandinavia (ARIES) riometer.

Chapter 3 explains what riometers are, introduces the various types of riometers and describes what they are used for. It builds on the knowledge about antennas as introduced in chapter 2. Sky maps and radio stars are also introduced and basic concepts relating to these are discussed. Knowledge about the working principles of riometers, sky maps and radio stars is important for the chapters to follow.

In **chapter 4** we introduce a set of programs to simulate the data flow through a Mills Cross type system from source to the final beam output. These programs (and the results they produce) help to deepen the understanding of the working principle of a Mills Cross type system such as the one used for ARIES. The simulations discussed in this chapter will also enable us to examine the signals inside the system at various stages, providing test data even before any hardware has been built.

The fact that this simulation is done at signal level implies that it is not possible to simulate long periods of time due to the amount of processing power and storage space required. For the same reasons, the simulation cannot be carried out in real-time, and there is a practical limit to the number of sources that can be simulated.

Chapter 5 will introduce a different simulation that is geared towards determining estimates for the required integration time in a realistic situation, but these simulations will no longer simulate the whole reception process (as done in chapter 4) but only the final cross-correlation stage. We explore how different factors contribute to the required integration time, these results are then used to extrapolate a realistic estimate of the required integration time.

Having looked at the basic working principles of antennas and riometers in chapters 2 and 3, and having simulated the low-level reception processes in chapters 4 and 5, **chapter 6** operates on a higher level of abstraction, focusing on radiation patterns and on how the concept of radiation patterns can help in the evaluation and deployment of real system designs.

The toolbox developed in this chapter will enable us to apply all findings to arbitrary riometers or, in fact, antenna systems.

Chapter 7 describes applications of the RIOSIM toolkit developed in chapter 6. RIOSIM was designed with many of these applications in mind, and this chapter aims to prove that RIOSIM fulfils these expectations and has in fact contributed to the successful deployment of ARIES in various ways, as many of the applications presented have been used during initial investigations and deployment of the ARIES riometer. Some have also proved useful for existing riometers, for example IRIS. We show that due to the object-oriented nature of the developed toolkit, the power of many of the applications presented can easily be harnessed for other riometers, usually by simply changing the location, time and beam pattern parameters appropriately.

Chapter 8 defines the ARCOM framework, a generic operating software for advanced riometer systems. The chapter is roughly structured along the process activities of the Software Engineering cycle. It does not go into implementation details of every single function call, aiming instead at providing a general (although more abstract) description of the working principles involved.

The design of operating software for a 'Wicked System' such as ARIES presents unique challenges to the software engineer. The chapter looks at what these challenges are, how they influenced the design of the ARIES operating software, and how the implemented software architecture solves the 'Wicked Problem.'

Chapter 9 contains discussions of the different results obtained from the 2002 ARIES experiment. During the 2002 experiment, a variety of datasets has been recorded for different configurations of a preliminary ARIES system. The data comprises several hundred gigabytes of raw input data as recorded from the A/D converters connected to the beamforming network, as well as integrated data derived from the raw data in real-time.

The first full ARIES system started recording first long-term datasets in March 2006, and data from ARIES in its final configuration is available from March 2007. **Chapter 10** describes the GLEAM algorithm developed to compensate for the correlation-related issues discovered during the initial experiments. This algorithm also has the potential to provide higher-quality image interpolation compared to the currently used approach. We describe the GLEAM algorithm and apply it to various test- and real datasets to show its effectiveness.

Chapter 11 summarises the thesis, and draws some conclusions for future development of riometers. The results presented in this thesis open up many new research opportunities, and some ideas for future work are presented.

The **appendix** contains a **glossary** of terms and abbreviations used in this thesis as well as various additional materials of interest. A **bibliography** concludes this thesis.

1.3 Typographical Conventions

The layout of this thesis adheres to the official guidelines of the University of Lancaster as published in [Par07]. In the electronic (PDF) version of this document, all (cross-)references have been hyperlinked for reading convenience. The following typographical conventions are used throughout this thesis:

- Variable parameters in equations are depicted by lowercase italic characters, e.g. n.
- **Bold** C-style notation is used for program functions (methods), i.e. **run()**.
- Classes, objects, components and variables are referred to in **bold text**, but without trailing parentheses, as in **CNoiseSource**.
- **Bold** and *italic* text is also used for general emphasis.
- Where deduction from the context is potentially ambiguous, the C++-style scope operator (::) is used to indicate which class a function belongs to, for example **CNoiseSource::-getSample()**.
- A `monospaced font` is employed for
  - File names (e.g. `small_circle.source`).
  - File contents.
  - Output of commands.
  - (Excerpts of) source code.
- A **`bold monospaced`** font is used for commands and names of programs entered on a command line (e.g. **`./sendcmd`**).
- Diagrams relating to software architecture adhere to the Unified Modelling Language (UML) notation.

Chapter 2

Antennas

Antennas are essential in any device that receives or transmits radio waves. This chapter summarises some properties of individual antennas as well as of arrays of antennas in general. This general description forms the basis for the more specific discussion of our area of interest, riometry, and how antennas are used in riometers in chapters 3 onwards.

2.1 Radiation Properties of Antennas

Antennas radiate or receive energy in form of electromagnetic waves. The radiation pattern of an antenna describes how the antenna in question responds to radiation received from any given direction. The radiation pattern of an antenna usually depends on the frequency of the radiation in question. Antennas can be designed to maintain a nearly constant radiation pattern over a wide range of frequencies. However, most simple antennas show strong variations in their radiation pattern as the operating frequency is varied.

From the description above it is clear that at least two parameters are needed to describe the radiation pattern of any given antenna, namely the spherical coordinates (azimuth and elevation) that specify the direction in question.

However, electromagnetic radiation can be polarised. To fully describe a radio wave propagating in any given direction, the polarisation state of this wave needs to be known. This can be specified through the Stokes parameters [Kra88], or, equivalently, through two complex phasors in space quadrature describing the two principal components of the electric field (or equivalently the magnetic field which is always perpendicular to the electric field) and the relative phase offset between them.

A general radiation pattern is therefore a function of direction and polarisation. From now on, we will refer to such a radiation pattern as complex radiation pattern, because two (complex) phasors are used to describe the radiation properties for each direction.

Most of the time, however, one is not interested in the instantaneous amplitude of the received signal. The property of interest is the average power received from any given direction. This is called the power gain of an antenna (often specified in relative terms relative to an 'isotropic radiator' and assuming unpolarised radiation). Another term that is often used to describe the properties of an antenna is the 'antenna directivity.' Often, this refers to the power gain as discussed above. Sometimes, the term directivity is also used to describe the *maximum* power gain.

2.2 A Brief History of Crossed Dipole Antennas

The individual antenna elements of today's riometers are crossed dipole antennas, also known as 'turnstile' antennas. This kind of antenna was first described by Brown in [Bro36]. Brown's goals were to build an antenna with circularly symmetrical radiation pattern that should concentrate the energy in the vertical plane so that the signal strength toward the horizon for a given power input will be considerably greater than that obtained from a single half-wave vertical antenna with the same input power. In fact, he built a vertical array of what we now know as crossed dipole antennas. In [Bro36], he also derives the horizontal radiation pattern of such an antenna, and how it changes as a function of the amplitude and phase of the signal that is fed into each of the two arms of the crossed dipole. He experimentally verified the uniformness of the radiation pattern if the current in the two arms is in time quadrature.

Brown built a model of such an antenna to operate at a wavelength of 3m (100MHz). Having verified the working principle, he then built a full scale antenna for operation at 6.7m (45MHz). Interestingly, Brown also mentions the superior fluctuation behaviour of 'his' antenna compared to a single half-wave dipole. He states that observers reported that, in districts where signals from a single half-wave antenna had fluctuated as much as ten to one due to changes in field distribution due to moving automobiles and possibly elevator cables, the signal from the turnstile only shifted between limits whose ratio was two to one. He states that this effect is probably due to the fact that the transmitting antenna is spread through a space two and one-half wave lengths long, thus giving 'diversity' effect. This is the same reason that led to the development of today's MIMO (multiple input multiple output systems) which are used in digital communication systems [SCT03].

Wells [Wel44] describes a 'quadrant aerial' consisting of two linear dipoles in space quadrature. He describes the radiation patterns of such an antenna for different lengths of dipoles. He also derives radiation patterns for such antennas above perfect and imperfect earth. The two dipoles of his 'quadrant aerial' are, however, not centred on each other. Their common point is located at the end of the dipoles.

Today, crossed dipoles are widely used in riometry. The main reasons are

- Omnidirectional radiation pattern.
- Circular polarisation matches predominant polarisation of incoming 'signal' from cosmic noise background.
- Simple and inexpensive to build [Nie01].
- Use of guy ropes (IRIS) or guide wires (ARIES) makes them highly resistant to environmental influences such as snow and wind.

In [Nie01], Nielsen approximates the power gain ψ of such a crossed dipole to be

$$\psi = 2 \cdot \sin(\frac{2\pi}{\lambda} h \cdot \sin(\frac{\pi}{2} - \theta)) \tag{2.1}$$

where λ is the wave-length, h the height of the antenna in wave-lengths above a perfect ground plane and θ the zenith angle. We can visualise this and other radiation patterns with the RIOSIM toolkit described in chapter 6.

2.3 Phased Array Antennas

2.3.1 Working Principle of an Additive Phased Array

Figure 2.1 shows a simple example of an additive phased array. The antenna elements are positioned in a Cartesian coordinate system, the location of each element i is described by its position vector $\overrightarrow{p_i}$. All lengths are specified in multiples of the operating wavelength. In figure 2.1, for simplicity's sake, there are four antenna elements, equally spaced 0.5 wavelengths apart on the x-axis and located symmetrically to the centre of the coordinate system. The following discussion is true for arbitrary aerial positions, however.

2.3.2 Reception

To understand the response of such an array to an arbitrary incoming radio wave, we logically divide the process up into the reception (this section) and beamforming (section 2.3.3 below) stages.

Let the direction of the incoming wave be specified by the unit vector $\overrightarrow{w}$. If we want to find the directivity pattern of the array, we simply have to determine the response of the array to all possible incoming wave directions $\overrightarrow{w}$. That is why, in the following discussion, we stick to one single direction $\overrightarrow{w} = \overrightarrow{w_0}$.

A wave received from a certain direction $\overrightarrow{w}$ is received by each aerial with a certain phase shift relative to the centre of the array (or, in fact, any arbitrary fixed point in space — known as the phase centre[1]). This individual phase shift depends on the placement $\overrightarrow{p_i}$ of the aerial element i in question and on the direction of the incoming wave as specified by $\overrightarrow{w}$.

Note that all phase information is relative to a certain fixed point and time, for convenience we specify that point to be the origin of the coordinate system. Thus, an aerial at $O(0 \mid 0 \mid 0)$ receives the wave from direction $\overrightarrow{w}$ with phase delay 0rad=0°. We will specify all angles in radians from now on.

The amount of phase delay for any given aerial position $\overrightarrow{p_i}$ is simply related to the projection of $\overrightarrow{p_i}$ on the direction vector $\overrightarrow{w}$ of the incoming wave:

$$\Delta\phi_i = -2\pi \cdot \overrightarrow{w} \cdot \overrightarrow{p_i} \tag{2.2}$$

This relation is also shown in figure 2.1, again for a simple two-dimensional case.

2.3.3 Additive Beamforming

In order to form a beam n pointing into a specific direction, the input signals are added together (thus additive array) with certain additional phase shifts $\beta_{i,n}$. These phase shifts determine the direction of the formed beam. Figure 2.2 shows this process for one specific beam. In addition, the signal from each individual antenna element can be attenuated by a certain amount, again affecting the shape and direction of the resulting beam. This step is known as tapering, and a variety of tapering functions $a(\overrightarrow{p_i})$ exist, leading to different sidelobe levels and beam widths. Theoretically, one could specify an individual tapering function for each beam. In practice, one

[1]To obtain correct results when adding the response from several (sub-)arrays, it is important that their phase centres coincide.

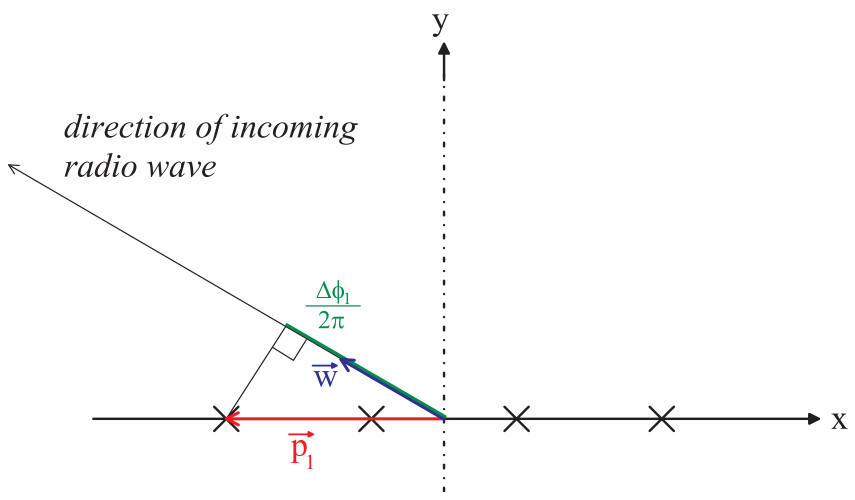


Figure 2.1: A simple phased array

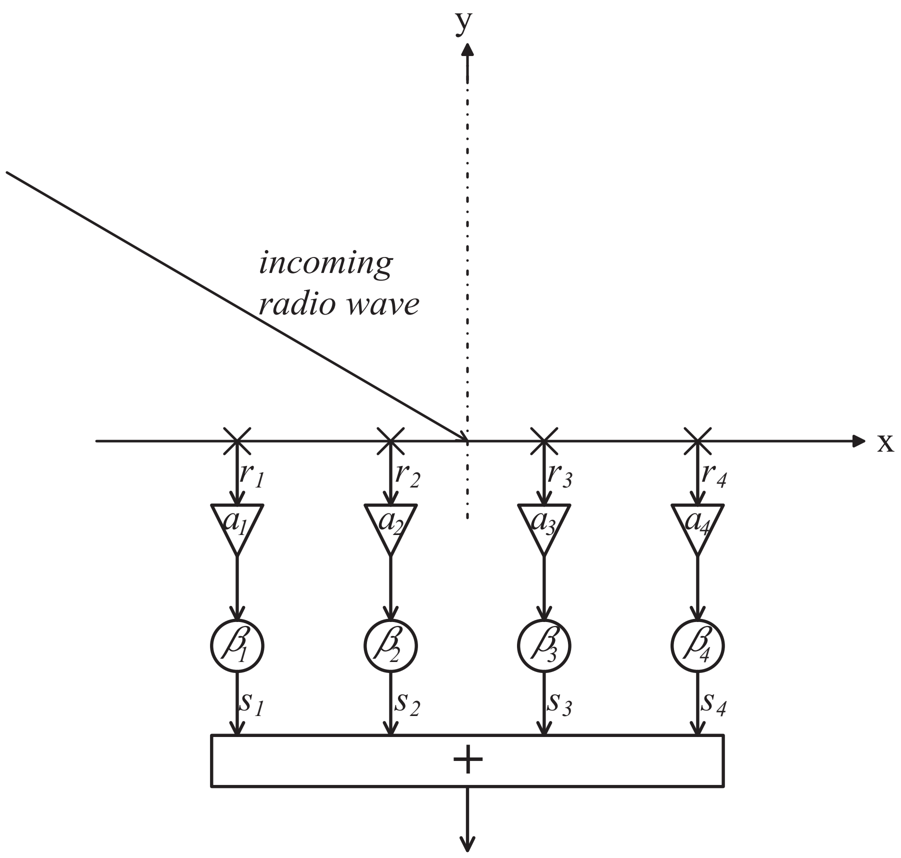


Figure 2.2: Beamforming summation

tapering function for all beams is normally used. All processing steps are shown in figure 2.2.

Mathematically, this reception and beamforming process can be described as follows:

Reception of signal by all aerials i:

$$r_i = e^{-j \cdot 2\pi \cdot \overrightarrow{w} \cdot \overrightarrow{p_i}} \tag{2.3}$$

Attenuate signal and apply phasing:

$$s_i = r_i \cdot a_i \cdot e^{-j \cdot \beta_{i,n}} \tag{2.4}$$

Add up all components to form the beam:

$$S = \sum s_i \tag{2.5}$$

2.3.4 Phase Angles for Butler Matrices

Butler Matrices use an optimised network of phase shifters to form multiple simultaneous beams from a phased array antenna [BL61, Mue72]. Systematic methods are available to design Butler Matrices of any size 2^N [Moo64]. A 'standard Butler Matrix' with N_{in} inputs has an equal number of outputs $N_{out} = N_{in} = N$, thus produces N beams. A Butler Matrix is normally used with linear arrays with equally spaced elements connected to its inputs $\{1, \ldots, N\}$ and applies the following phase factors:

$$\beta_{i,n} = \left(i - \frac{N+1}{2}\right) \cdot \frac{(n-1) - \frac{N-1}{2}}{\frac{N}{2}} \cdot \pi \tag{2.6}$$

where the term $\frac{(n-1)-\frac{N-1}{2}}{\frac{N}{2}} \cdot \pi$ is the phase progression from element to element for a given beam number, and $\left(i - \frac{N+1}{2}\right)$ is a factor that ensures symmetry to the origin (centre of the array).

With formulas 2.3 to 2.6 we can now calculate and plot the beam patterns of a 32 element linear array by simply calculating the response of all outputs of the corresponding 32 port Butler Matrix while looping through all possible signal directions. The result for a vertical slice through the resulting pattern along the axis of the array is shown in figure 2.3. The green numbers are the beam numbers, corresponding to the output ports of the Butler Matrix. Beam 9 is highlighted to show the form of a single beam more clearly. The individual beam patterns exhibit the -13dB sidelobe level typical for untapered linear arrays.

2.3.5 Cosine Tapering

Tapering (i.e. attenuating) the signals from the individual antenna elements leads to differently shaped beams. In general, the aim of tapering is to reduce the sidelobes. The drawback is that the main beam is broadened in doing so.

If, for example, we taper the individual signals according to a cosine function as shown in figure 2.4, the resulting beam pattern for the same Butler Matrix as in 2.3.4 will look like in figure 2.5. Again, beam 9 is highlighted, and it is clearly wider than the original beam 9 in figure 2.3. At the same time, however, the level of the first sidelobes is reduced to -32dB.

Other forms of cosine tapering are possible, for example according to the dotted line in figure 2.4. This will lead to the beam pattern shown in figure 2.6.

The formula that was used to calculate the attenuation factors for the continuous line in figure 2.4 is

$$a_i = \frac{1}{2}\left(\cos\left(\frac{i - \frac{N+1}{2}}{\frac{N-1}{2}\cdot\pi}\right) + 1\right) \tag{2.7}$$

Formula 2.8 below was used for the dotted line in figure 2.4.

$$a_i = \frac{1}{4}\left(\cos\left(\frac{i - \frac{N+1}{2}}{\frac{N-1}{2}\cdot\pi}\right) + 3\right) \tag{2.8}$$

2.3.6 Pairwise Addition

As Nielsen suggests in [Nie02b], with reference to [Mue72], a beam pattern similar to the one in figure 2.6 can be achieved by pairwise addition of the output ports of a 32 port Butler Matrix using 16 signal combiners connected to the 32 output ports of the Butler Matrix, as shown in figure 2.7. This leaves us with 16 beams, the beam pattern of which looks like in figure 2.8. Clearly, there is some resemblance between figures 2.8 and 2.5. Pairwise addition of output ports therefore leads to similar results to cosine-shaped tapering, but without the need of additional attenuators in the input signal lines, therefore reducing overall signal loss.

In figure 2.8, beam 5 is highlighted. Beam 5 is the addition of beams 9 and 10 of the original constellation in figure 2.3.

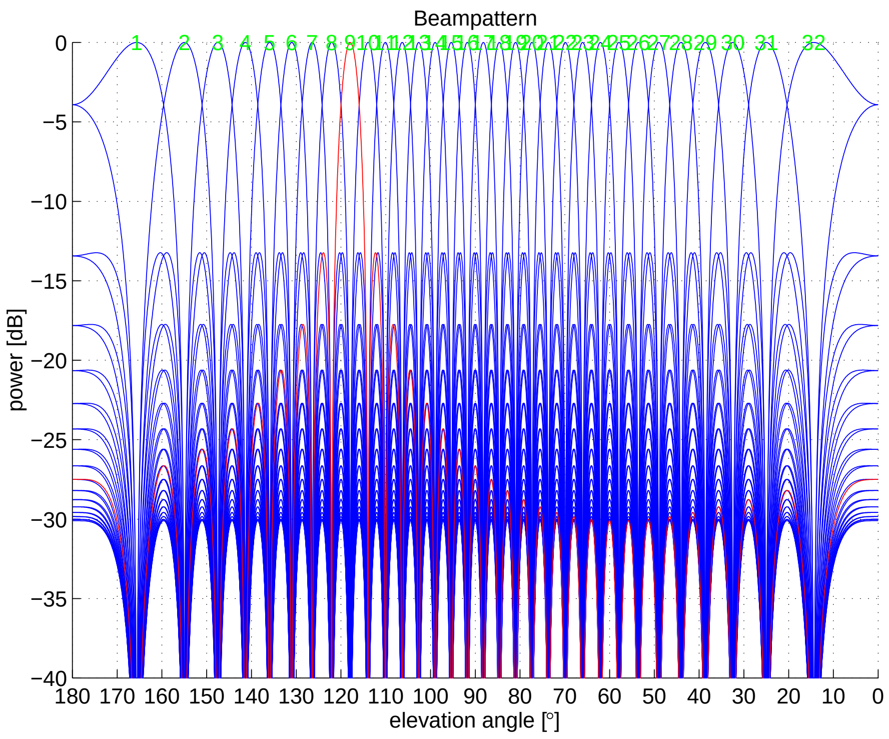


Figure 2.3: Beams formed by a 32 port Butler Matrix connected to a linear antenna array. Cut along the vertical plane containing the antenna elements.

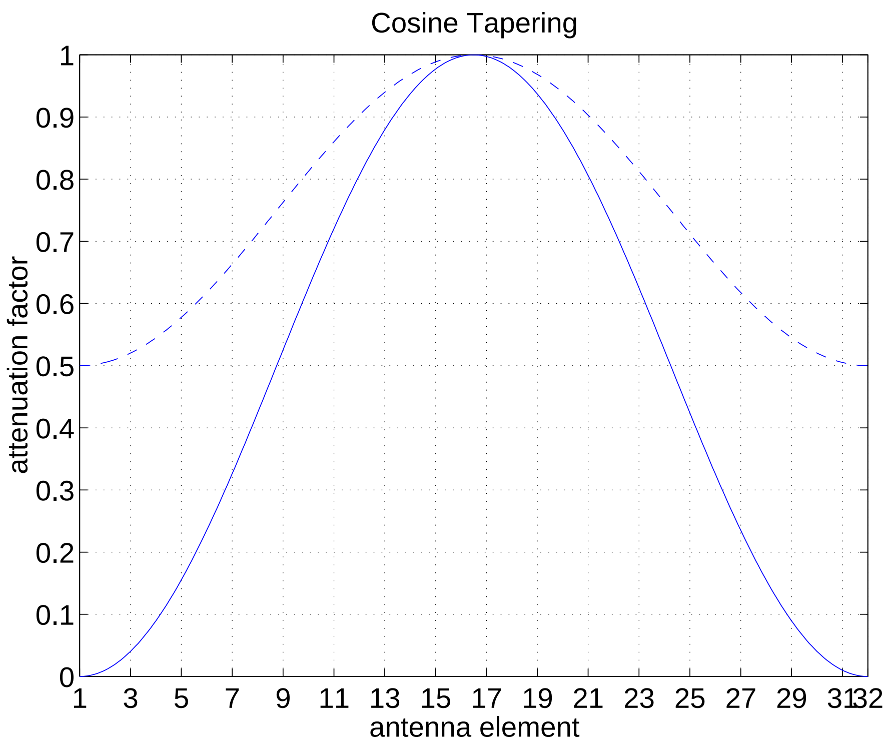


Figure 2.4: Two different ways of cosine tapering for a 32 element linear array. Solid line shows cosine tapering according to equation 2.7, dotted line shows 'half' cosine tapering according to equation 2.8.

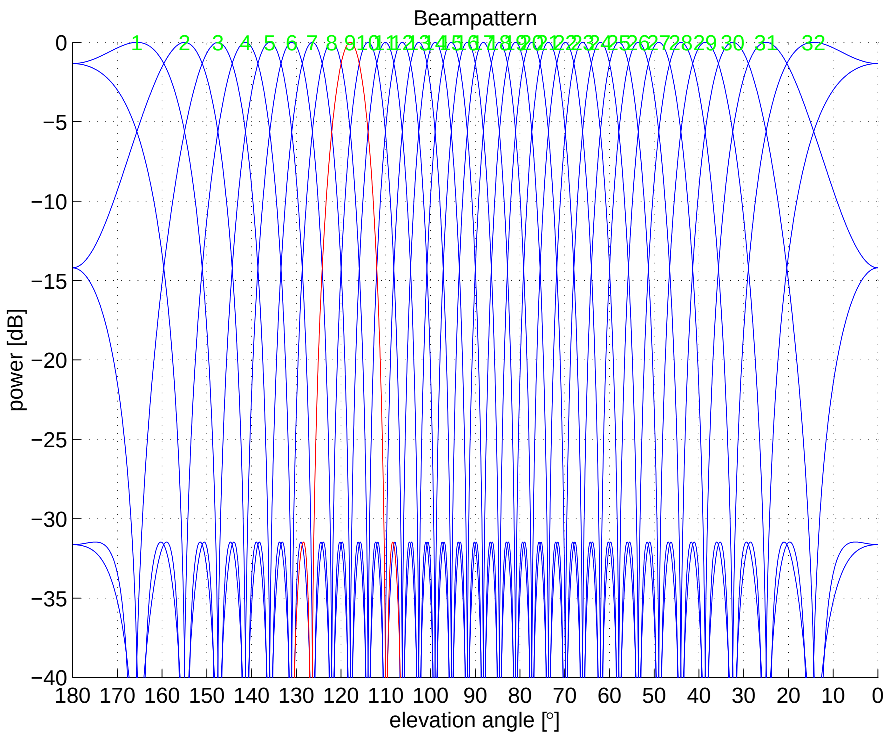


Figure 2.5: Beam pattern for cosine tapered input signals. Note that the level of the first sidelobes is reduced to -32dB.

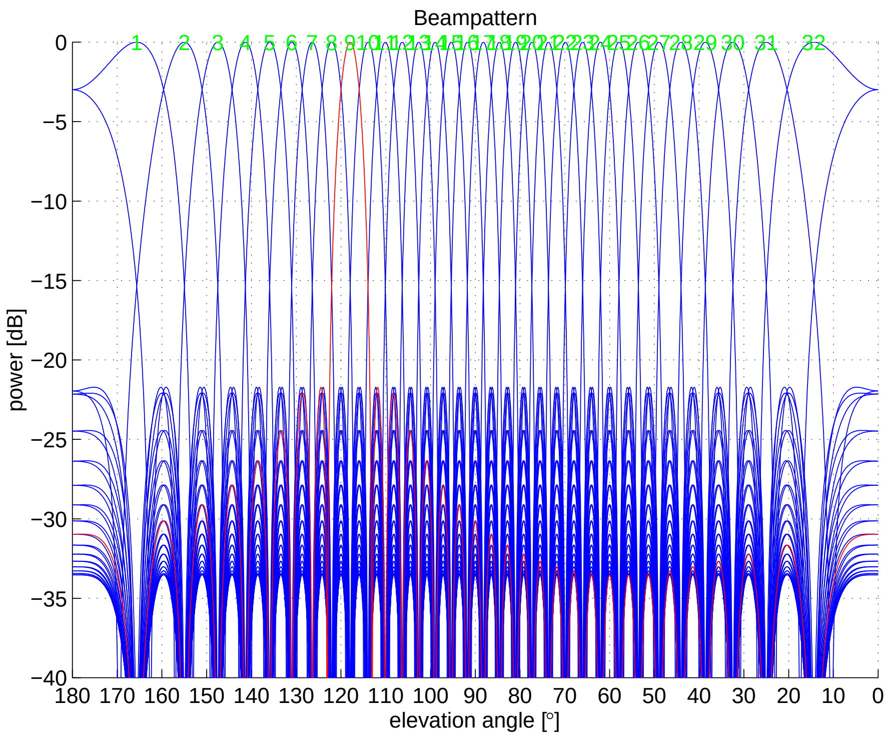


Figure 2.6: Beam pattern for ‘half’ cosine tapered input signals

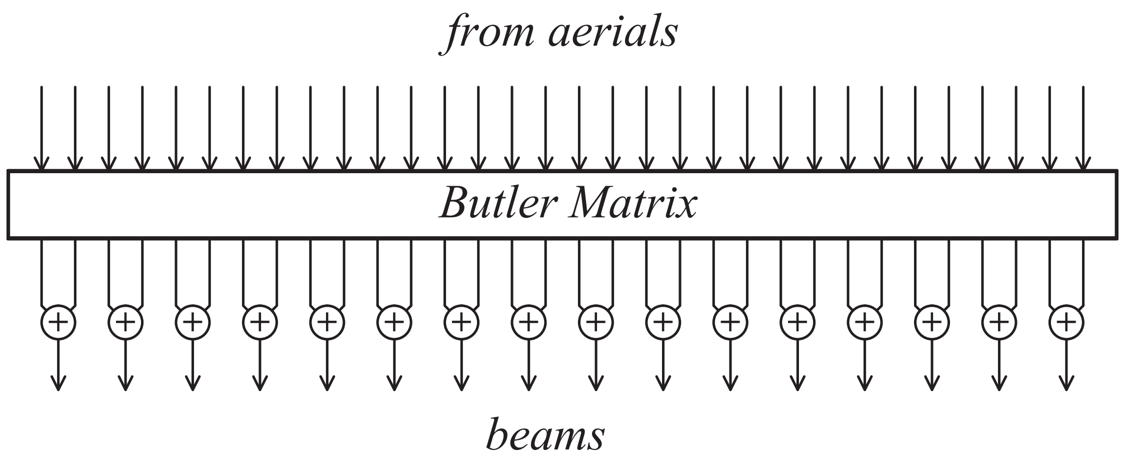


Figure 2.7: Pairwise addition of Butler Matrix outputs

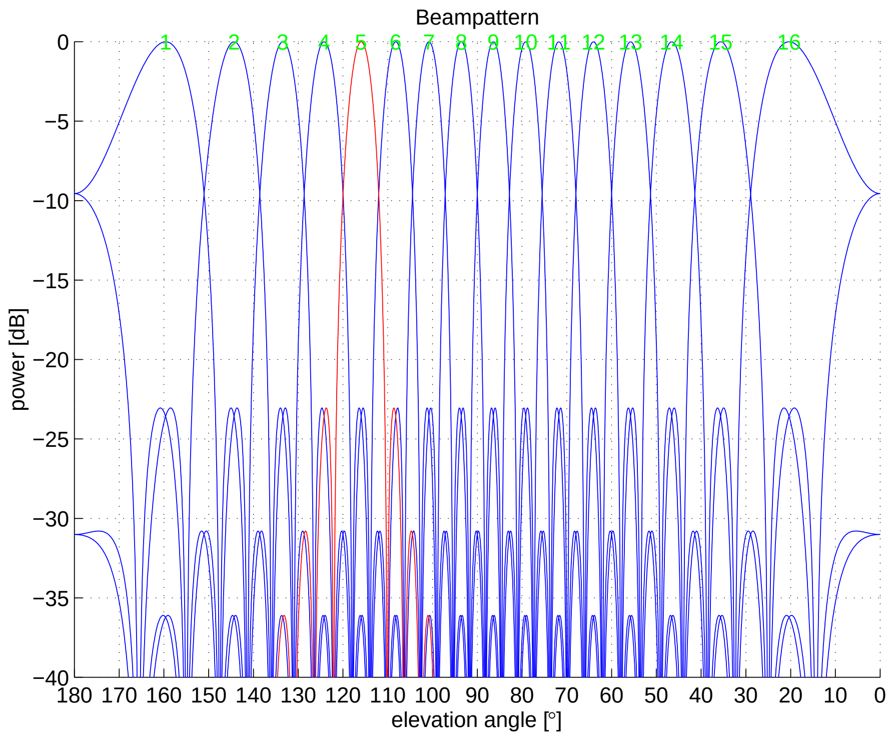


Figure 2.8: Beam pattern of a 32 port Butler Matrix with outputs added pairwise

2.3.7 Conclusion

With additive phased arrays one can achieve highly directional antennas. These arrays tend to suffer from high sidelobe levels which can be reduced by tapering, in turn broadening the main lobe.

Pairwise addition of the Butler Matrix outputs seems to be an interesting alternative to attenuating the input signals. It turns out, however, that the signal combiners needed to perform this pairwise addition are rather more expensive than attenuators.

Also, the pairwise addition method obviously leaves no flexibility for modifications, whereas the attenuator method enables one to shape the beams according to a multitude of tapering functions, cosine tapering (figure 2.4) being only one of many possibilities. Based on these results, it was decided to run initial ARIES Mills Cross experiments without using any kind of tapering. Though this will be of limited use for operation as a riometer later on, we can expect useful results from this configuration. In particular, we should be able to measure the high sidelobes of the untapered array, giving us confidence that the receiving system is working according to specification. The narrow, untapered pencil beams will also enable us to accurately measure the location of strong radio sources, and compare these measurements to theoretical simulations — thereby confirming correct alignment of the antenna array.

2.4 Mills Cross Antennas

2.4.1 A Brief History of Mills Cross Type Antenna Arrays

B. Y. Mills initially used three aerials connected together as Michelson-type interferometers [TMGWS86] at a frequency of 101MHz to examine the galactic distribution of discrete radio sources [Mil52]. Reasonable, albeit less than originally anticipated, accuracy was achieved, with an average probable error in position of around 0.2% for strong sources and 2% for weak sources.

What is now known as Mills Cross antenna is first described in [ML53]. Mills's goal was to construct an aerial system of high resolution but small area and low cost for investigations in radio astronomy. Mills explicitly mentions that this kind of aerial system sacrifices gain (i.e. effective area) for high resolution and low cost. This was more than acceptable because Mills's goal was to accurately map the positions of strong radio sources.

Mills original design is depicted in figure 2.9. Conventional linear dipoles were used as individual elements. Mills's first operational cross-type system was a $24+24$ element cross operating at 97MHz.

Having obtained encouraging results from this first system, Mills built a $250\times 2+250\times 2$ element Mills Cross type radio telescope for use at 3.5m (86MHz) [MLSS58]. Other telescopes were developed at the same time, see for example [Sha58]. Christiansen and Mathewson used a Mills Cross antenna array to scan the solar disk at a wavelength of 21cm with unprecedented detail [CM58]. The interesting thing about this particular Mills Cross antenna is the fact that Christiansen did not care about sidelobes. He designed the array in such a way that the spacing between two adjacent lobes was large enough so that no two antenna lobes could fall on the sun at the same time. Note that Christiansen only used one pencil beam at any given time. He made successive scans of the sun during the course of several days to derive the brightness distribution across the whole solar disc. He also employed phase shifters in one of the two arms of his Mills Cross antenna array to steer the fan beam generated by this arm, therefore also changing the position of the resulting pencil beam.

2.4.2 Working Principle of a Mills Cross Antenna Array

The working principle of a Mills Cross antenna array is as follows.

2.4.2.1 Fan Beams

Initially, each arm of the cross is considered as a separate linear additive phased array (see section 2.3.1). Figure 2.9, panel (b) shows the idealised outline of the radiation patterns of the individual arms for the zenithal case. Each arm forms what is called a fan beam. The pointing direction of this fan beam can be influenced by appropriate phasing techniques such as the ones described in section 2.3.3. For example, if we have an arm of 32 antennas positioned half a wavelength apart from each other, and those antennas are connected to a 32 port Butler Matrix, we get 32 fan beams like the ones depicted in figure 2.3.

It is important to understand that, at this stage, the two arms of the Mills Cross are completely separate. Each arm forms fan beams of its own. If viewed from above and slightly idealised, these fan beams will be perpendicular to each other, just as the linear phased antenna arrays that were used to create them. (This is a simplified view for explanatory purposes. The fan beams are in fact cone-shaped, as can be seen in many of the 3D radiation pattern plots to follow in this and later chapters.)

By themselves, the recordings from these fan beams are still rather useless. They cover a large solid angle, the spatial resolution (at least in one direction) is still very poor, in fact it equals the 'resolution' of a single antenna element. The direction of any signal recorded by such a fan beam can only be estimated in one direction, and only if no other interfering noise sources are present at the same time.

2.4.2.2 Cross-correlation, Pencil Beams

The idea that turns the Mills Cross antenna array into a high resolution array is based on cross-correlation of the signals from two perpendicular fan beams. This will extract only the signals that originate from the overlapping region of the two fan beams. A narrow pencil beam is therefore being formed for each combination of a fan beam from one arm with a fan beam from the other arm of the Mills Cross.

In case of a $32+32$ antenna element Mills Cross array with a 32 port Butler Matrix for each arm, $32 \times 32 = 1024$ pencil beams are therefore being formed, though not all of them are physically meaningful, and even fewer perform well enough in terms of noise level and sidelobe behaviour to be suitable for further use. These issues will be discussed in more detail in subsequent chapters, primarily for one particular system, the Advanced Rio-Imaging Experiment in Scandinavia (ARIES).

Figure 2.10 is a 3D representation of the beamforming process. The two small panels on the left show an example of a fan beam formed by a linear array of antennas along the y-axis (top panel) and along the x-axis (bottom panel), respectively. The small inset in the upper right hand corner of each panel shows an idealised top-down view of several fan beams generated by the arm in question, with the shown fan beam highlighted.

The big panel on the right-hand side depicts the cross-correlation process. Two perpendicular fan beams (shaded) are cross-correlated to produce a narrow pencil beam (solid) pointing in the direction where the two fan beams intersect. Again, the diagram in the upper right-hand corner shows an idealised 2D version of this process as viewed from above. Signals from the two green fan beams are cross-correlated to derive the signal that is common to both fan beams and must therefore originate from the intersecting area, depicted by a red circle.

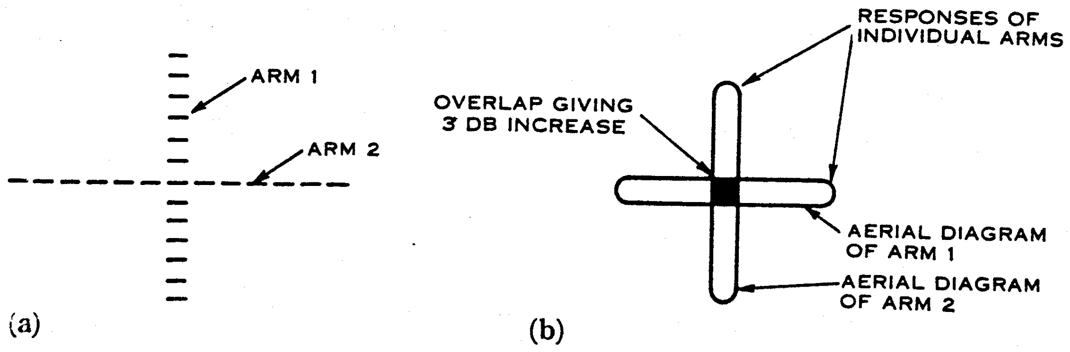


Figure 2.9: Original Mills Cross, taken from [ML53]. (a) Plan view of dipoles in cross arrangement. (b) Idealised response of the cross arrangement, plan view.

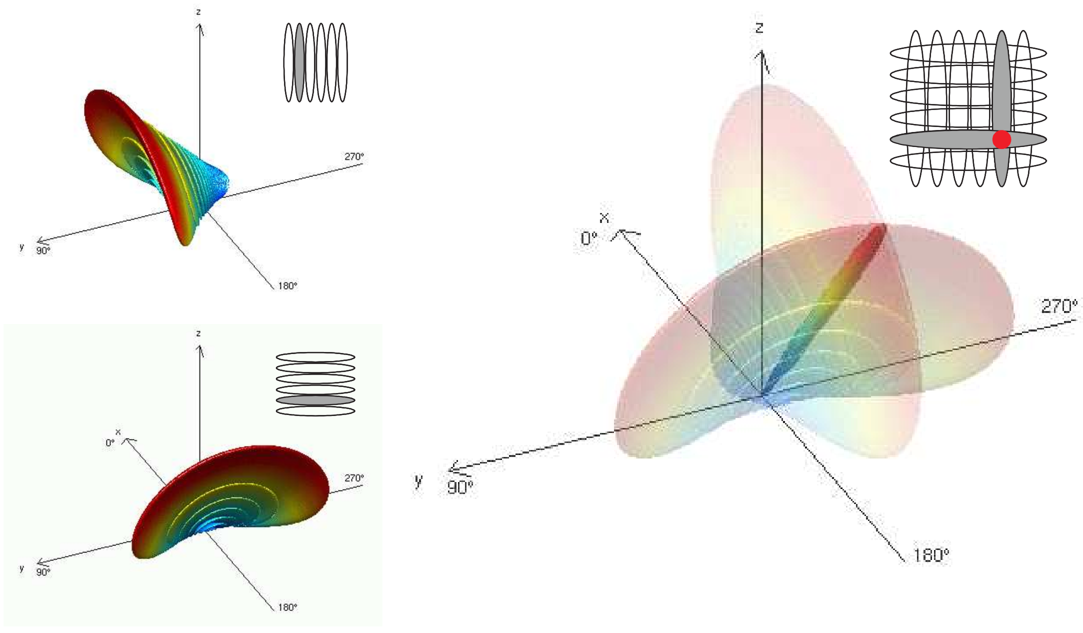


Figure 2.10: Beamforming for a Mills Cross

2.4.3 Disadvantages of a Mills Cross

The following sections list well-known disadvantages of a Mills Cross antenna array when compared to a filled array. Note that these points don't make a Mills Cross inferior to a filled array antenna, since superiority/inferiority always depends on the intended use of a system. Later on in this thesis we will investigate how these issues can be dealt with, and we will find that the Mills Cross still offers advantages over a filled array for use in riometry, mainly due to the by far smaller number of antenna elements required, translating into significant savings in terms of money and real estate.

2.4.3.1 High Sidelobe Levels

This section explains why the Mills Cross antenna array produces higher sidelobes than a corresponding filled array for the untapered case. The Mills Cross forms the output signal of each pencil beam by cross-correlating the signals from two perpendicular fan beams generated by the two arms of the cross. The arms themselves are simple linear phased arrays and therefore produce a sidelobe level of around -13dB (in the untapered case).

Suppose, the received time series from the two perpendicular fan beams are called j_t and k_t, respectively. Each signal has sidelobes, a source signal from the direction of the first sidelobe is therefore attenuated by -13dB in power. The signal itself, however, is therefore only attenuated by $\sqrt{-13\text{dB}}$.

The worst case happens when a source signal x_t is received in the main lobe of one fan beam and in the first sidelobe of the perpendicular fan beam. In this case the cross-correlation between the two signals will only result in a power attenuation of –6.5dB:

$$p_t = \langle j_t \cdot k_t \rangle \tag{2.9}$$

$$p_t = \langle x_t \cdot (\sqrt{-13\text{dB}} \cdot x_t) \rangle \tag{2.10}$$

$$p_t = -6.5\text{dB} \cdot \langle x_t x_t \rangle \tag{2.11}$$

Of course, the well-established technique of tapering (see section 2.3.5) can be employed to reduce sidelobes. We will touch on sidelobe issues for the Mills Cross again in more detail in chapters 4, 9 and 10.

2.4.3.2 Noise Behaviour and Bandwidth

Christiansen [CM58] mentions that he employed different receiver bandwidths depending on the position of the observed source. He used a bandwidth of ‘several MHz’ for observations near the zenith. For observations far from the zenith (in his case the sun in midwinter), he reduced the bandwidth down to 0.3MHz. He notes that this had the effect of increasing the amplitude of noise fluctuations. Quote: “The narrowing of the bandwidth for directions away from the normal to the plane of the array is made necessary by the difference of the path length from the source to the different parts of the array. This difference in path length corresponds to a difference in phase which is not exactly the same at all frequencies in the pass band of the receiver. Hence for a given direction of the source, the bandwidth of the receiver must be kept sufficiently narrow so that phase differences over the pass band are not large enough to cause a significant deterioration in the performance of the array.”

This effect has a limiting influence on the maximum usable bandwidth although first experiments show this to not be a significant issue for ARIES (with a nominal bandwidth of 1MHz), therefore this effect will not be discussed further in this thesis.

2.5 Step-by-Step Guide to Reception from a Mills Cross

Armed with the knowledge from the previous sections, this section aims to present a complete and ordered view of the reception process from a Mills Cross type system from a signal processing point of view. The purpose of presenting this material in detail is threefold. Firstly, to justify the simplified approach taken during the simulations discussed in chapters 4 and 5. Secondly, to lay the foundations for the radiation pattern simulations in chapter 6 and ultimately for the GLEAM algorithm presented in chapter 10. Thirdly, to attempt to shed a bit of light on the mind-boggling phasing issues relating to Mills Cross arrays, particularly when using them to observe spatially distributed sources. With reference to figure 2.11, we can identify the following steps:

1. Any given antenna (element) will respond to a signal from any given direction as described by the antenna’s radiation pattern. This pattern describes the antenna’s response to incoming signals in the two polarisation components as described by two phasors (we adapt an x-y description throughout this thesis, although any set of orthonormal base vectors will work equally well).

2. Just like the radiation pattern itself, any incoming signal can be described using two phasors for the two polarisation components. Note that we need to use the same coordinate system for the next step to be meaningful.

3. Ignoring an arbitrary phase offset (which is constant for all directions), the antenna will exhibit a signal $\text{DirectivityX} \times \text{conj}(\text{Ex\_incoming}) + \text{DirectivityY} \times \text{conj}(\text{Ey\_incoming})$ at its terminals.

4. Beamforming networks, such as the additive beamformer described in section 2.3.3, will combine (phase-shifted=delayed and tapered) versions of these signals to effectively form a new radiation pattern, that of a fan beam in the Mills Cross case.

5. A cross-correlator such as used in a Mills Cross configuration will multiply the signals from two such beamforming networks to produce the narrow pencil beam made up from identical parts of the signals from the two fan beam inputs. This is the only non-linear processing stage in the Mills Cross reception process.
   It is at this stage, that phase offsets introduced by the beamforming networks cause a phase offset in the resulting cross-correlated ‘power’ value for any given direction. As long as only point sources are examined, this is not an issue, as we can always use the absolute power value as a measurement of the power received from that point source. As soon as we examine a spatially distributed source, however, this effect needs special consideration, this will be discussed in further detail in chapter 10.

6. Throughout this description, we work with the basic assumption that signals from different directions are uncorrelated. The cross-correlator will therefore never produce an output for signals from two different directions, and the overall received power for any given pencil beam can simply be calculated as the sum of all signals (from all directions). This sum is the signal visible at the output terminals of the cross-correlator. Note that this is a complex weighted sum according to the phase offsets mentioned above.

7. According to Kraus [Kra88] (from [Sin50]) the voltage response V of an antenna to a wave of arbitrary polarisation is given by

$$V = k \cos \frac{MM_a}{2} \tag{2.12}$$

   Where MM_a is the angle subtended by the great-circle line from polarisation state M to M_a

and k is a constant involving field strength of the wave and size of the antenna. As we are dealing with (incoming) signals of random polarisation, MM_a will vary randomly between 0° and 180°, averaging at 90°. On average, for random polarisation, equation 2.12 will therefore result in a constant factor $< V >= k \cos 45^\circ$ which we can safely ignore for all considerations that are only interested in relative signal levels and/or phase relations.

In our modelling of the Mills Cross, we use the basic arrangement of figure 2.11. The following discussion shows that, for any given direction of interest, even for randomly polarised incoming waves, the phase difference detected by the cross-correlator will be constant and not dependent on the actual state of polarisation of the incoming wave. Furthermore, the amplitude of the cross-correlator output signal will be constant for any given power influx with random polarisation.

The incoming signal is received by the antenna, described by the radiation pattern $\beta_{2,x}(\theta,\phi)$ and $\beta_{2,y}(\theta,\phi)$. The array patterns for arms A and B simply scale those signals in amplitude and phase. In reality, the signals are combined before being passed through the array beamformer, but this is a linear operation and could therefore equally well be performed separately for the two polarisation planes.

Moving the 'reception' stage after the beamformer does not change the signal in any way either, as this is again a completely linear operation. This is in fact why the notion of radiation patterns is such a useful one, as we can now describe the response of an array made up of antenna element patterns and cross-correlator by one resulting 'combined' radiation pattern. Let us now consider phase and amplitude separately:

As we have just seen, the reception process will introduce *the same phase offset* into both branches (representing both arms of a Mills Cross type antenna array) in figure 2.11. The cross-correlator will therefore detect exactly the same phase *difference* between the two signals, independent of the actual received waveform, dependent only on the different phase offsets introduced by the different beamforming networks of the two arms for any given direction of arrival. This phase offset and how it depends on direction of arrival is an inherent system property.

The *amplitude* of the received signals in branches 1 and 2 will vary depending on how well the antenna and incoming signals are matched. However, in the case of randomly polarised incoming signals, equation 2.12 tells us that, on average, we will see both signals attenuated by the same constant value. On average, the observed signal amplitude of the cross-correlator output for any given direction will therefore only depend on the properties of the underlying (combined) radiation pattern and, moreover, will be proportional to the overall 'combined' radiation pattern

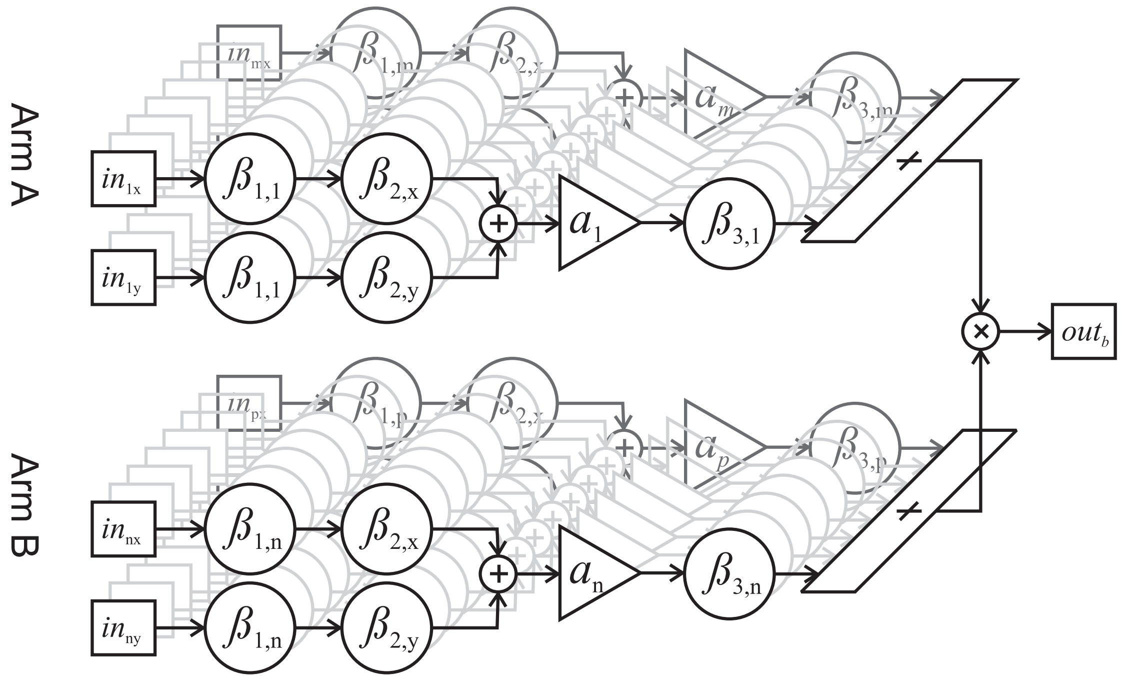


Figure 2.11: Signal-processing view of reception of signals by a Mills Cross antenna array from any one direction (θ, ϕ) and for one particular pencil beam b. Arm A consists of aerials 1...m, arm B consists of aerials n...p. in_{ix} and in_{iy} are the x and y polarisation components of the incoming radio wave (assumed to be identical for all aerials), $\beta_{1,i}$ are phase shifts (delays) due to the location of the aerial in question relative to the phase centre of the array. $\beta_{2,x}$ and $\beta_{2,y}$ describe the element radiation pattern in the x and y polarisation planes (assumed to be identical for all aerials), a_i are the tapering factors, $\beta_{3,i}$ are the delays introduced by the additive beamformer for one particular (fan) beam, out_b is the output signal for this particular direction (θ, ϕ) and pencil beam b.

power gain in that direction.

2.6 Summary

This chapter provided an introduction to antennas in general, filled phased array antennas and finally Mills Cross type antenna arrays. This provides the background for the discussions in the following chapters. In the next chapter we will show how these antennas are currently being used in the field of riometry and how riometers can benefit from the increased spatial resolution of a Mills Cross antenna array.

Chapter 3

Riometers

This chapter introduces Relative Ionospheric Opacity Meters (riometers). The process of how riometers are used for measuring ionospheric absorption is explained and some of the scientific uses of riometers are outlined. The concept of sky maps and radio stars are also introduced. The chapter builds on the previous chapter about antennas (chapter 2). This chapter lays a foundation for the following chapters, which are concerned with various design and development aspects of the Advanced Rio-Imaging Experiment in Scandinavia (ARIES) riometer.

3.1 Working Principle

Riometers (Relative Ionospheric Opacity Meters) measure to what extent cosmic background noise is being absorbed by the ionosphere. The acronym itself appears to have first been used in [LL59]. Some literature uses the extended definition 'Relative Ionospheric Opacity Meter using Extra Terrestrial Electromagnetic Radiation.' Riometers measure this 'ionospheric opacity' indirectly by subtracting the actual received signal power (which depends on the current transparency of the ionosphere) from a predefined quiet-day curve (QDC). The QDC represents the power level that is received on a perfectly 'quiet day,' i.e. when no absorption occurs and the ionosphere is completely transparent. Therefore, the difference between received signal power and the corresponding QDC value directly represents absorption. A continuously updated list of riometers can be found in [Mare].

Figure 3.1 is a Hammer equal-area projection [Paw05] of a map of the celestial sphere in the galactic coordinate system (see appendix B, section B.1.8), with colour representing arbitrary logarithmic power units. As an example, the yellow outlines give the position of one beam

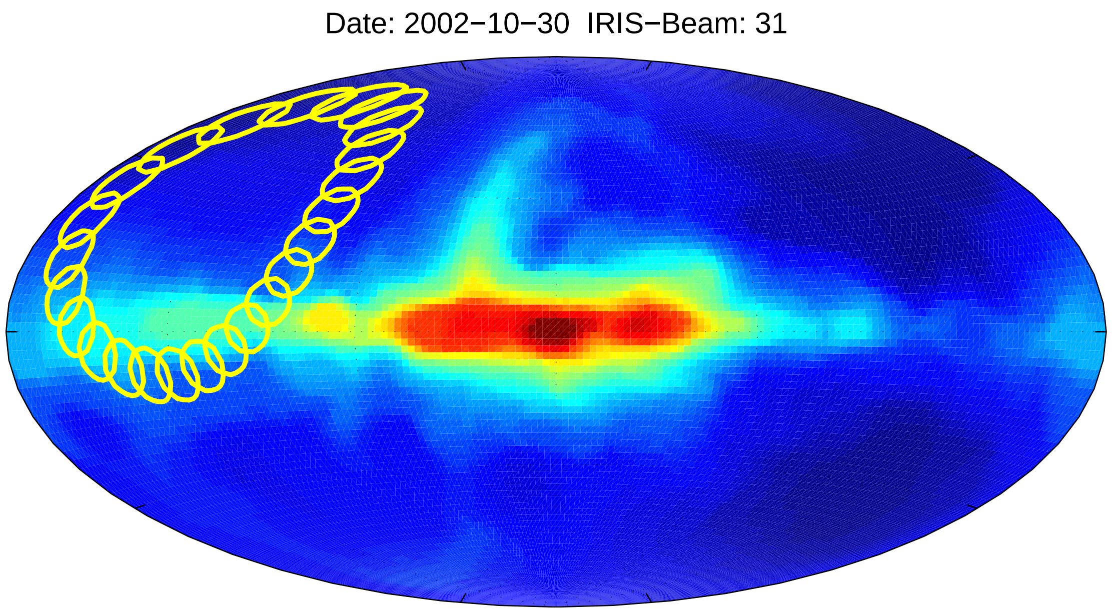


Figure 3.1: Trace of IRIS beam 31 during one sidereal day

(beam 31) of the IRIS riometer in Kilpisjärvi during the course of one day. The time difference between any two outlines is 1 hour. The rotation of the Earth causes the beam to move on the celestial sphere in a repetitive, predictable way. It takes exactly one sidereal day for the beam to come back to its starting position.

A riometer continuously records the received power from its beam(s). On a perfectly quiet day, the recording from this particular riometer (IRIS) for beam 31 would look like the red curve in figure 3.2. The recorded power varies according to the sky temperature in the direction of the beam, and it returns to the same value at the end of every cycle.

Such quiet days do seldom exist in reality, however. The actual recorded data during the course of one day will look more like the blue line in figure 3.2. This line shows real data as recorded by IRIS for beam 9 during the course of the day 2002-10-30.

Different methods for creating quiet-day curves (QDCs) from real recordings are discussed by Tao [Tao04], see also [DS90] (Density method), [KDR85] (Inflection Point method), [MH07] (Percentile method, manuscript in preparation) and references therein. In addition to empirical methods, since we can plot the trace of beam 31 in figure 3.1, such quiet-day curves can also be predicted from simulations, given that the temperature distribution of the cosmic background noise and the radiation pattern of the riometer are known with sufficient detail. We will discuss the generation of theoretical quiet-day curves in section 7.4 in chapter 7.

To arrive at a measurement for absorption, the entity of interest, a riometer now calculates the instantaneous ratio of received signal level to expected (quiet-day) signal level. Per definition, this quantity is called absorption. On a dB scale this operation amounts to a simple subtraction of the QDC from the received signal.

The result of subtracting the received signal as depicted in figure 3.2 from the QDC in the same figure is shown in figure 3.3. Per-beam absorption data as shown in figure 3.3 is the primary output of a riometer.

Note that figure 3.3 also shows areas of 'negative absorption.' While a certain level of 'negative absorption' can be explained by the random nature of the received signal, strong 'negative absorption events' indicate that more power has been received than expected. Common causes for this are solar radio bursts, lightning and man-made interference. Usually, the cause of such events can be identified based on the exact shape and duration of the event, though in some cases this proves difficult and error-prone.

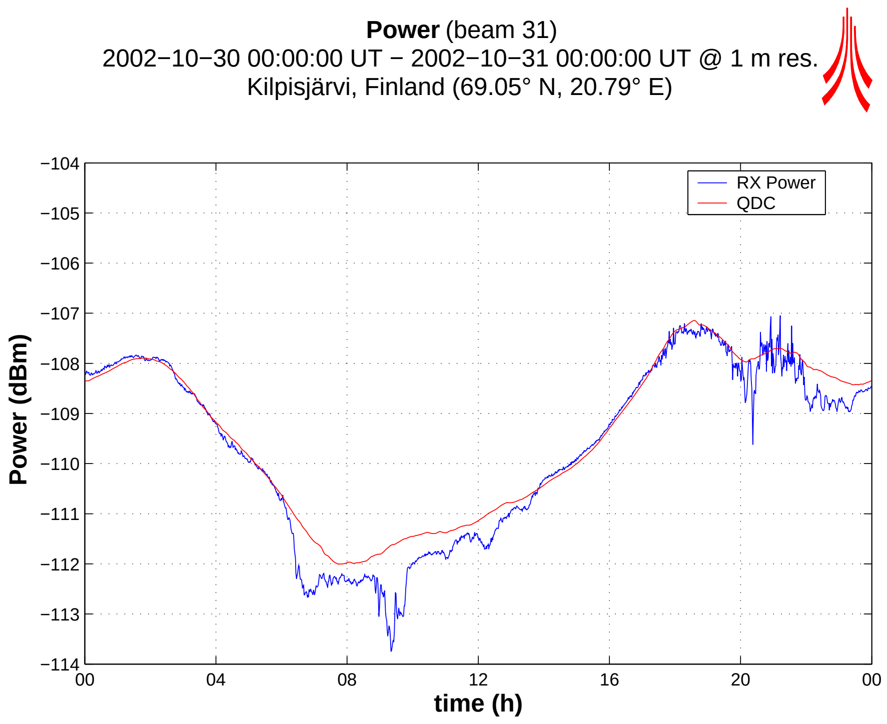


Figure 3.2: IRIS QDC and recorded power for 2002-10-30

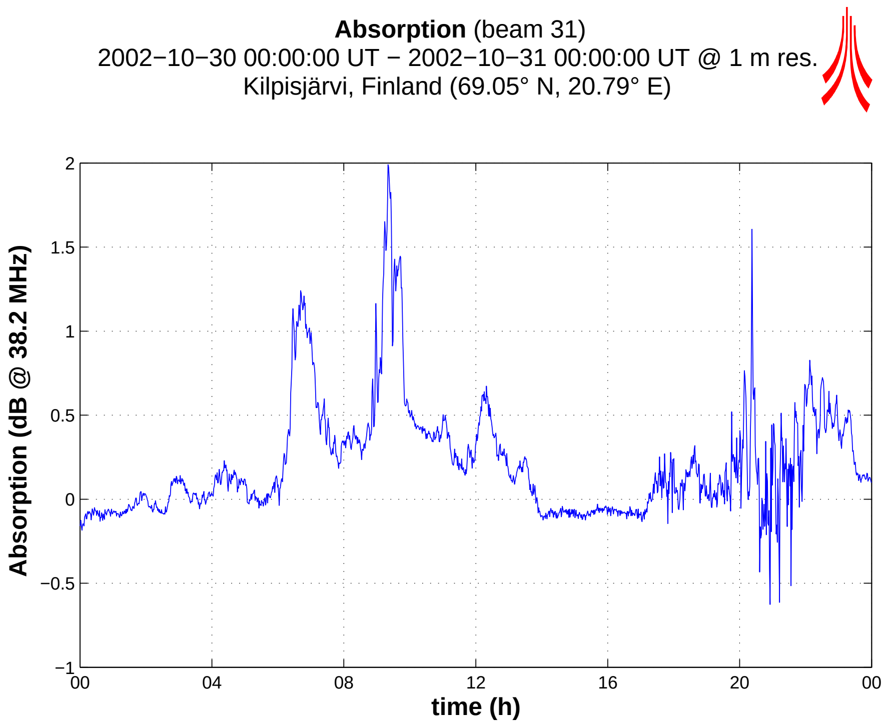


Figure 3.3: IRIS absorption for 2002-10-30

3.2 Types of Riometers

Widebeam riometers (figure 3.4, panel (a)) are the simplest type of riometer. They consist of a single aerial connected to a receiver and data logger and consequently have a very wide field of view but no imaging capabilities. Widebeam riometers are useful for a general overview of the current state of the ionosphere, but do not provide any information on the spatial structure within their field of view.

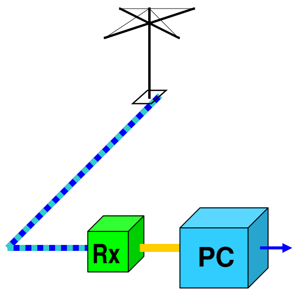
The need for higher spatial resolution led to the development of *imaging riometers* (figure 3.4, panel (b)). They are typically made up of between 64 and 256 antenna elements, connected as a phased array (see section 2.3 in chapter 2). The physical complexity of such a riometer is higher, since beamforming matrices and multiple receivers and/or switching circuitry are required.

The Space Plasma Environment and Radio Science (SPEARS) group at Lancaster operate an 8×8 aerial riometer system, also known as IRIS (Imaging Riometer for Ionospheric Studies, [DR90]). The IRIS type of riometer was originally developed by the University of Maryland [DR94]. The IRIS riometer run by the SPEARS group is located at Kilpisjärvi, Finland and described in [BHH95]. An IRIS type riometer achieves a maximum angular resolution of 13° at zenith [DR90].

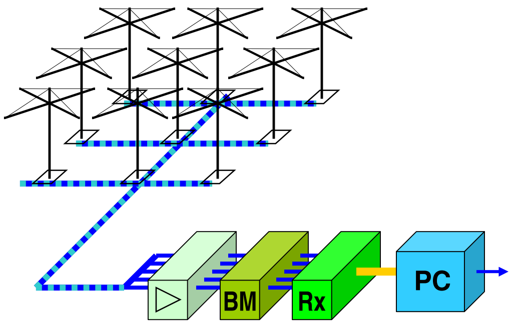
A 256 element phased array imaging riometer such as the one described in [MMK+97] achieves an angular resolution of 6° at zenith which translates to an area of about 11×11km at a height of 90km (90km is usually considered the average height of the absorbing region for riometry purposes, although the actual absorption peak depends on the type of event being observed).

There is still need for higher spatial resolution to resolve small-scale structures. However, to increase the spatial resolution of a filled phased array antenna by n, the number of required antenna elements increases with n^2 (i.e. with n in both x and y directions). This renders high resolution phased array antennas impractical. In fact, even a 256 element antenna array presents a significant expense, and the physical requirements are substantial (a 256 antenna array for operation at 38MHz requires a flat area of about $70 \times 70\text{m}^2$ [Mur, MMK+97].

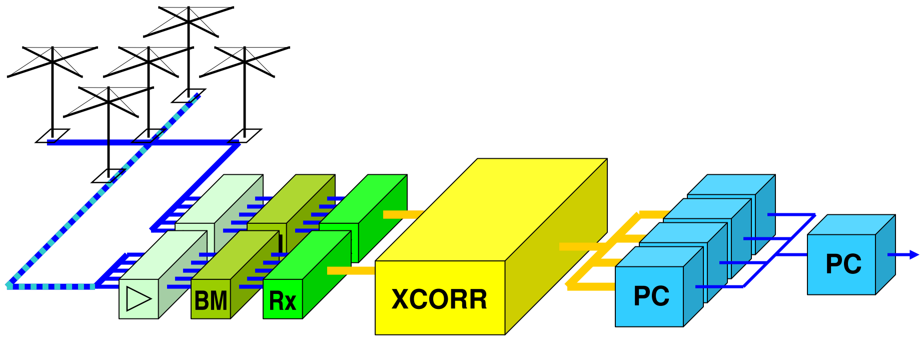
This led to the idea of a riometer based on the *Mills Cross* technique [ML53, Nie01], see figure 3.4, panel (c). Such a riometer, based on the principles discussed in sections 2.4 and 2.5 in chapter 2, can achieve high spatial resolution with significantly fewer antenna elements.



(a) widebeam



(b) phased array



(c) Mills Cross

Figure 3.4: Complexity of widebeam, phased array and Mills Cross type riometers

Figure 3.5 compares the physical dimensions of several riometer layouts. To achieve *n* times the resolution, the number of antenna elements required for a Mills Cross type antenna array only increases linearly with *n*, since only the (one-dimensional) arms become longer.

The use of a Mills Cross type antenna array implies higher requirements for the signal processing part of the system (see especially section 2.5 in chapter 2). We will look at some of the signal processing aspects for ARIES in the following chapters.

A drawback of a Mills Cross type system compared to a filled phased array system is the required integration time. The integration time required for a Mills Cross type system is generally higher, due to the reduced aperture of the antenna array. Preliminary results show that the achievable integration time is still short enough to make a riometer based on Mills Cross technique scientifically useful. More details on estimated achievable integration times will be discussed in chapter 5.

3.3 The ARIES Riometer

The Advanced Rio-Imaging Experiment in Scandinavia (ARIES) as conceived by Nielsen in [Nie01, NH97] is a 32+32 element Mills Cross type imaging riometer located in Northern Norway near Tromsø. Figure 3.6 shows a long-distance shot of the instrument and its surroundings, demonstrating how little impact a Mills Cross type antenna array has on its environment compared to a filled array, see also the discussion in section 3.2 above. In fact, the riometer array and control hut is just about visible in the centre of the image.

Figure 3.7 is a close-up view of the hut and a few central antenna elements. Finally, figure 3.8 shows the inside of the hut containing the receiver electronics. A block diagram of the hardware part of the ARIES system is contained in appendix D courtesy of Peter Chapman, along with a high-level schematic of the receiver circuitry. Note that while during initial tests all processing including cross-correlation and post integration was done in high-level software (ARCOM, see chapter 8), these initial investigations led to some of these tasks being taken over by a Field Programmable Gate Array (FPGA). Figure D.4 in appendix D is a flow diagram illustrating the FPGA-based processing steps as implemented by Keith Barratt, taken from [Bar07]. Also contained in appendix D is a diagram (figure D.5) showing the physical layout of the ARIES antenna array and the numbering scheme for aerials, fan and pencil beams as well as the orientation of the coordinate system adopted throughout this thesis.

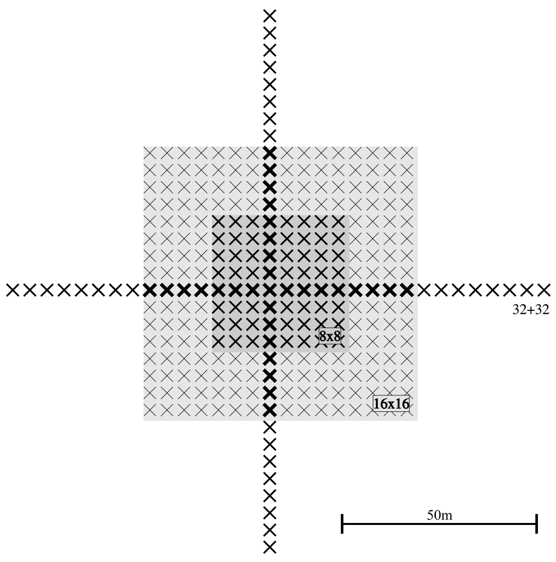


Figure 3.5: Physical layout of several Mills Cross and filled array antenna configurations

Figure 3.6: Long-distance view of ARIES. Aerials and control hut are just about visible in the centre of the picture.

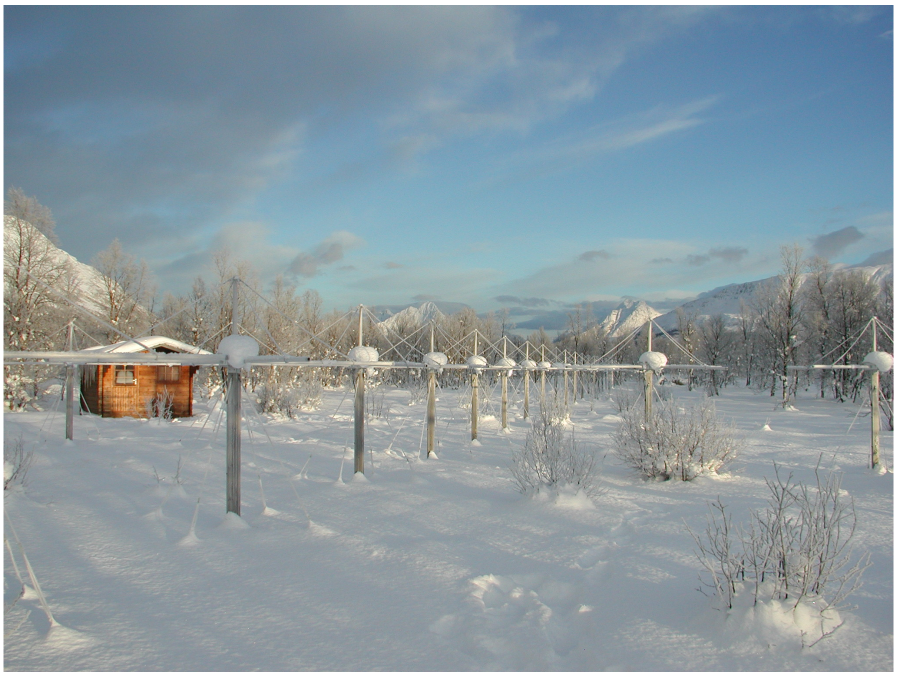


Figure 3.7: Close-up view of ARIES

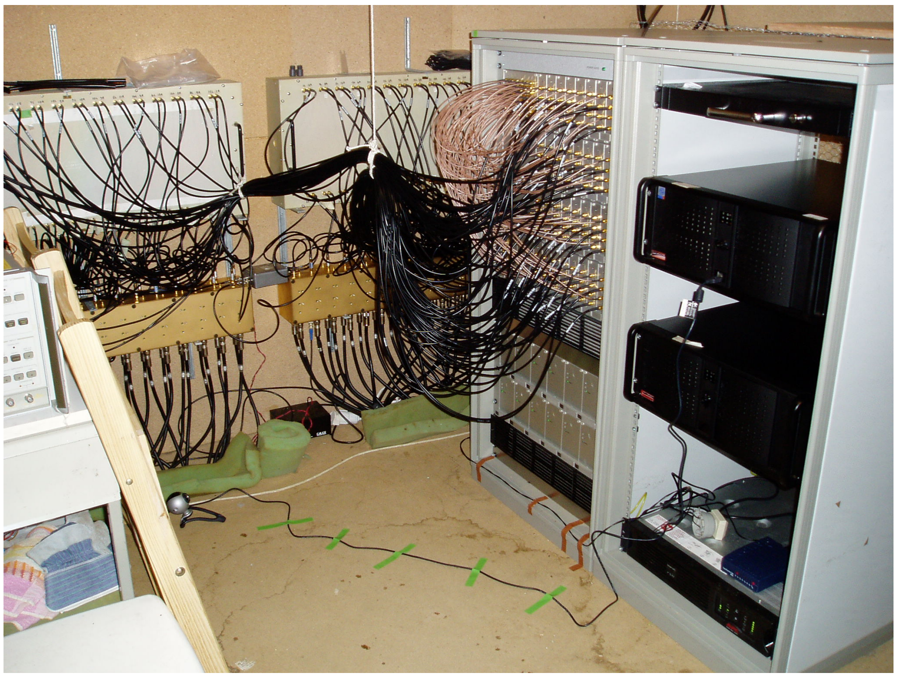


Figure 3.8: Inside the ARIES control hut. This picture still shows the analogue beamforming matrices and pre-amplifiers, which have later been replaced by fully digital beamforming.

3.4 Scientific Applications of Riometry

As pointed out by Wild in [Wil06], “[...] it is not enough to observe the solar system via space missions alone. Large spatial and long time scale measurements made by ground-based experiments are required to complement localised measurements made by spacecraft during relatively short missions.”

Riometers make a significant contribution to these long time scale measurements, with absorption measurements dating back to about 1935 [Har95, p. 66], first (single beam) riometer datasets being available from 1953 [MS53] and the first multi-beam (imaging) riometers operating from about 1965 [Har69, Ans65]. In the auroral oval, absorption is predominantly due to the precipitation of high-energy particles that produce enhanced electron density in the D and E layers of the ionosphere [Kav02, Har95]. Absorption data as measured by riometers is therefore used in a wide range of research. Some areas where riometer data plays an important role are presented below.

3.4.1 Ionosphere

The ionosphere is that part of the Earth’s atmosphere (see figure 3.9) ranging from a height of about 70km up to 1000km. The properties of this part of the atmosphere are determined by the fact that the gas atoms and molecules are ionised. The ionosphere as a whole is still electrically neutral, but ionisation enables the flow of electric current. Ionisation in the ionosphere is mainly caused by solar radiation and the interaction of the solar wind with the Earth’s magnetic field (magnetosphere). It is therefore dependent on location, local time of day, season and on the current level of solar activity, which changes with the 11-year sunspot cycle [AR02].

The ionosphere is divided into different regions, or layers, based on the plasma density that prevails in that layer. These layers have different properties. For example, the lower layers (D, E) reflect radio waves of relatively low frequency (1–10MHz) and enable HF (high frequency) communication over large distances [Bar02]. The higher F layer (sometimes divided into F1 and F2 layers) reflects radio waves of higher frequencies. The exact properties and heights of the different layers of the ionosphere at any given time can be probed with instruments such as an ionosonde, an example is the EISCAT Dynasonde at Tromsø [Dav96].

3.4.2 Riometer Observations

Riometer observations are unique in that they provide continuous information about the lowest region of the ionosphere, the D layer. Due to their wide field of view and continuous coverage, (imaging) riometers provide information on the spatial extent and lifetimes of precipitation features that complement the spot measurements taken by radar and in-situ. Riometer observations provide the spatial context within which to interpret radar data and reveal the dynamics of precipitation regions [CHHW97]. Other instruments such as radars or ionosonde are more sensitive to higher layers. Riometer observations extend the total observable region downwards [Hon01]. Riometers are therefore often used together with other instruments, enabling the derivation of the entire height profile of geophysical events. Such events are usually solar-driven and enable the study of the coupling process between the solar wind, the interplanetary magnetic field (IMF) and the Earth's atmosphere, with riometers observing the 'footprint' of these events in the ionosphere.

3.4.3 Ionospheric Processes

Man-made modifications of the ionosphere, so-called 'heater' or 'artificial aurora' experiments increase ionisation in small parts of the ionosphere using strong transmitters in the range of several MW. Again, the results can be observed with riometers (and other instruments such as optical cameras), and the obtained data allows insight into wave-plasma interactions and chemical processes in the ionosphere.

3.5 Radio Stars

The bright radio sources in the sky stand out considerably from the cosmic noise background. This causes effects like scintillation (rapid variations in apparent brightness of a distant object when viewed through a medium such as the atmosphere or ionosphere, caused by refraction due to small-scale variations in the medium density [Ric77]), which are not always wanted. In any case, it is important to know what strong radio sources there are and where they come from. This section gives a brief description.

Figure 3.10 shows some of the strong radio stars in the sky. We find, that at the frequency of interest (the operating frequency of most riometers, namely around 38MHz — a protected frequency band), the strongest radio sources in the northern hemisphere are Cassiopeia A and

Cygnus A. It is interesting to note, that the quiet (undisturbed) sun is more than an order of magnitude weaker at these frequencies, therefore the quiet sun is not usually visible in riometer data.

For details about the spectra of Cas A, Cyg A and other radio sources, see for example [BGPW77, KPW69].

The term ‘radio star’ is somewhat misleading and is merely used for historical reasons. Sources of strong radiation are not necessarily associated with stars. The first ‘radio star’ was discovered by J. S. Hey after the Second World War [Jen66, p. 52] in the constellation of Cygnus. Soon after that, John Bolton discovered a smaller radio source in the constellation of Taurus, the position of which coincided with the so called Crab Nebula. Finally, the strongest radio source, Cassiopeia A, was discovered by Martin Ryle in 1947 [Jen66, p. 54].

3.5.1 Cassiopeia A

Cassiopeia A is a supernova remnant within our own galaxy. From observations of the motion of individual diffuse filaments in the Cassiopeia nebulosity, it can be deduced that the initial supernova explosion leading to the creation of Cassiopeia A must have happened about 320 years ago [Jen66, p. 56].

Cassiopeia A is becoming weaker over the years, recent studies show a clear decay in power. This decay might just about be spotted in recorded IRIS measurements, though no effort to do this has been undertaken as of yet.

3.5.2 Cygnus A

Cygnus A, the second brightest radio source in the sky, is an extragalactic radio source, situated at a distance of about 550,000,000 light years from the Earth [Jen66, p. 77]. Cygnus A is a so-called binary source: it consists of two centres of emission. This fact was first discovered by R. C. Jennison in 1950 [Jen66, p. 55].

3.5.3 Simulating Reception from Radio Stars

Optical astronomers measure the brightness of objects by measuring the apparent magnitude, or flux density, of the object. The flux density is a measure of the power received from the object per unit frequency, per unit area [AST]. Power received from distant objects at radio wavelengths is often given in units of flux density, the Jansky (Jy) [SCW]. 1 Jansky equals $10^{-26}\frac{W}{m^2 \cdot Hz}$, i.e.

the power received from the object in question with a flux density of d Jy will be $d \cdot 10^{-26}$ Watts per square metre of (effective) antenna aperture and per Hz receiver bandwidth.

The effective antenna aperture A_{eff} in a given direction (θ, ϕ) can be derived from the gain $G(\theta, \phi)$ in that direction by

$$A_{eff} = \frac{\lambda^2}{4\pi} G(\theta, \phi) \tag{3.1}$$

see, for example, [AST].

Thus, to determine the power $P_{receivedfromsource}$ received by a given telescope with gain pattern $G(\theta, \phi)$ and Bandwidth B from a (point) radio source with flux density F at location $(\theta_{source}, \phi_{source})$ the following formula 3.2 can be used:

$$P_{receivedfromsource} = \frac{1}{2} F \cdot A_{eff}(\theta_{source}, \phi_{source}) \cdot B \tag{3.2}$$

$$= \frac{1}{2} F \cdot \frac{\lambda^2}{4\pi} G(\theta_{source}, \phi_{source}) \cdot B$$

The factor $\frac{1}{2}$ once again coming from the fact that we can only receive power from one of the two possible directions of polarisation.

3.6 Sky Maps

Knowledge of how the radiation coming from the sky is spatially distributed enables us to simulate reception of these signals without first having to build the actual instrument. This can give useful information about the dynamic range required by the receiving equipment as well as provide us with sample data before the instrument has even been built.

Sky maps map this cosmic background noise. A variety of sky maps exist, and they differ widely in parameters like resolution and base frequency.

This section gives a short history of two sky maps that are suitable for our purposes. We will use these sky maps in later chapters.

The noise power recorded in sky maps comes from the following different contributors [SLM+90]: The relic or cosmic background radiation (CBR) T_{cbr}, the galactic (thermal plus synchrotron) radiation T_{gal}, and the integrated flux of unresolved extragalactic sources T_{ex}. Sironi et al. [SLM+90] give the proportions at 600MHz (UHF) as:

$$T_{gal} : T_{cbr} : T_{ex} = 7 : 3 : 1 \tag{3.3}$$

At VHF frequencies, the galactic radiation, whose temperature increases as the frequency decreases, becomes even more dominant.

The CBR is the noise received 'from the edge of the universe,' and recent studies (for example [TCC+03]) have succeeded to resolve that component, at least at microwave wavelengths. It is thought that knowledge of the structure of this Cosmic Microwave Background (CMB) will give new insights into the evolution of the universe.

The relative contribution of each of the three sources of radiation is not of particular interest in this thesis, so when we talk about the Cosmic Noise Background in the following sections, we will always refer to the resulting total measurable background noise $T = T_{gal} + T_{cbr} + T_{ex}$. This radiation forms a continuous background that can be mapped.

Some sky maps remove the effects of radio sources that can be clearly identified in the data, like the bright radio stars Cassiopeia A and Cygnus A. We will describe how the specific sky map handles these strong radio sources in the respective paragraphs below.

3.6.1 Purpose

In this thesis, sky maps will predominantly be used for the simulation of different configurations of the IRIS and, more importantly, the ARIES riometer systems. These simulations will produce theoretical QDCs for any given date and they can be used for evaluating dynamic range and absolute power for each of the beams, for a variety of different beam configurations (fan beams, tapered, untapered, etc.).

3.6.2 Requirements

For the purpose described above, the following main requirements for a suitable sky map can be deduced:

- Coverage of at least the celestial hemisphere that is seen by radio telescopes located in Northern Norway.
- Suitable resolution.
- Suitable frequency.

- Suitable content.

3.6.3 Coverage

The received power by an arbitrary antenna–receiver–system can in simple terms be described as the product of antenna radiation pattern R and power $P_{detectable}$ radiated by the the background sky in the direction in question (calculated as discussed in section 3.6.7), integrated over all possible directions as given in equation 3.4. See, for example, [Tao04].

$$P_{received} = \int_{\Omega} R(\theta,\phi) P_{detectable}(\theta,\phi) d\Omega \tag{3.4}$$

With coordinates given in the horizontal coordinate system of the observer (see section B.1.5). Since in our case only antenna systems on the Earth are considered, it is sufficient to deal only with positive elevation angles, as the sky background power in the direction of negative elevation angles will be shielded by the Earth. Thus, to simulate reception by a given radio telescope system at a given location on Earth, we only need to know about the sky background power for that hemisphere of the sky that is not blocked by the Earth itself.

The pink area in figure 3.11 shows the relevant area for the two main systems concerned, IRIS and ARIES, on an equal-area map of the whole sky. Due to the fact that these instruments are located away from the poles, the hemisphere that is seen by IRIS/ARIES sweeps over more than 50% of the total celestial sphere during one sidereal day. For good simulation results, we need to use a sky map that covers all of this pink area.

As we will see later, not many sky maps are available that cover the whole area in question. It will be shown, however, that even with simple interpolation one can get very good results as long as the area of missing data is located in a region of the sky with low structural content.

3.6.4 Resolution

The sky maps themselves were recorded by some sort of receiving equipment, the angular (spatial) resolution of which is obviously limited, even though some radio telescopes achieve extraordinary resolutions nowadays, compared to the beginnings of radio astronomy.

If the resolution of the receiving equipment that was used to produce the sky map is worse (i.e. coarser) than the resolution of the receiving equipment to be simulated, we cannot expect to get useful results, as the sky map will have been subject to ‘smoothing’ due to the radiation pattern of the original recording system [BGS97, p. 49], and it is this 'smoothed' sky that we then see with the simulated system. An important parameter to specify resolution is the beamwidth of the receiving equipment that was used. This value needs to be smaller than the beamwidth of the narrowest possible ARIES/IRIS beam (around 4°).

3.6.5 Frequency

The frequency that the sky map was recorded at needs to match the intended operating frequency of the instrument to be simulated as closely as possible. Even though scaling of a given map of the radio background power is possible to a limited extent, the results derived from these scaled sky maps will always be less accurate than the results from a sky map whose original frequency matches the intended operating frequency of the equipment.

The fact that simple scaling is not sufficient, especially for large offsets between (original) sky map frequency and required frequency, shows clearly in diagrams like the one in figure 3.10. As different radio sources have different spectra, the cosmic sky background power does not vary according to any one given function over the whole sky.

3.6.6 Content

Some sky maps have been manually 'corrected.' Sometimes this means that the brightest radio stars have been removed from the given sky map, since they are not considered to be part of the general cosmic noise background. This means, however, that the influence of these radio stars needs to be taken into account separately. Although this can be done, it involves an additional step and it would generally be preferable in our case to have all cosmic 'background' power recorded in one place, namely the sky map.

3.6.7 Simulating Reception from a Sky Map

The power received from the background sky can be expressed by means of 'equivalent noise temperature' T_B in Kelvin (K). The Nyquist Relation relates temperature to power as follows:

$$P_{available} = k \cdot T_B \cdot B \tag{3.5}$$

where k is Boltzmann's constant and B is the bandwidth of the receiver. Normally, an additional factor $\frac{1}{2}$ is introduced, as any given antenna will only be sensitive to one of the two polarisation modes. This leads to equation 3.6 below.

$$P_{detectable} = \frac{1}{2} k \cdot T_B \cdot B \tag{3.6}$$

In case of the sky, power is not distributed uniformly across the whole spectrum, i.e. the power received from the sky is not white noise. This leads to the fact that a map of the radio sky taken around a certain frequency cannot easily be transformed into a map at a different frequency (see also section 3.6.5), unless a certain power distribution is assumed. If the two frequencies in question are not too far apart, the transformation can be done approximately, and this was used, for example, by Tao [Tao04], who converted Cane's sky map at 30MHz to 38.2MHz, or by Cane himself [Can78], who constructed his 30MHz sky map from several existing sky maps at slightly different frequencies.

For an example, the reader is referred to later chapters of this thesis, first and foremost chapter 7. Figure 7.4 therein shows the simulated power received by simulated IRIS beams from Cane's sky map described in section 3.6.8.1 below during one day. One can clearly see the peak when the respective beam swipes through the galactic plane. Note also, that not all peaks appear as would be expected from the real data shown in the same figure. The real peaks are (at least partly) due to the bright radio star Cassiopeia, and this radio star was removed from this particular sky map (see also section 3.6.6). The influence of these radio stars can be taken into account separately, as has been done, for example, in chapter 9 discussing first ARIES experiment results.

3.6.8 The Sky Maps Used in this Thesis

The following sections describe several sky maps. Table 3.1 gives a summary of the sky maps described. Simulation results using different sky maps will be presented in chapter 7.

A variety of other maps are available, even at the low frequency end of the spectrum. (For example [Kas88], or see [DU90] for a table of low frequency observations. Also see [SKVa] for a list of surveys that are included in the SkyView virtual telescope [SKVb] — most of these are at much higher frequencies.) However, these surveys tend to cover only small parts of the whole sky and are therefore unsuitable for our purposes (see also section 3.6.3).

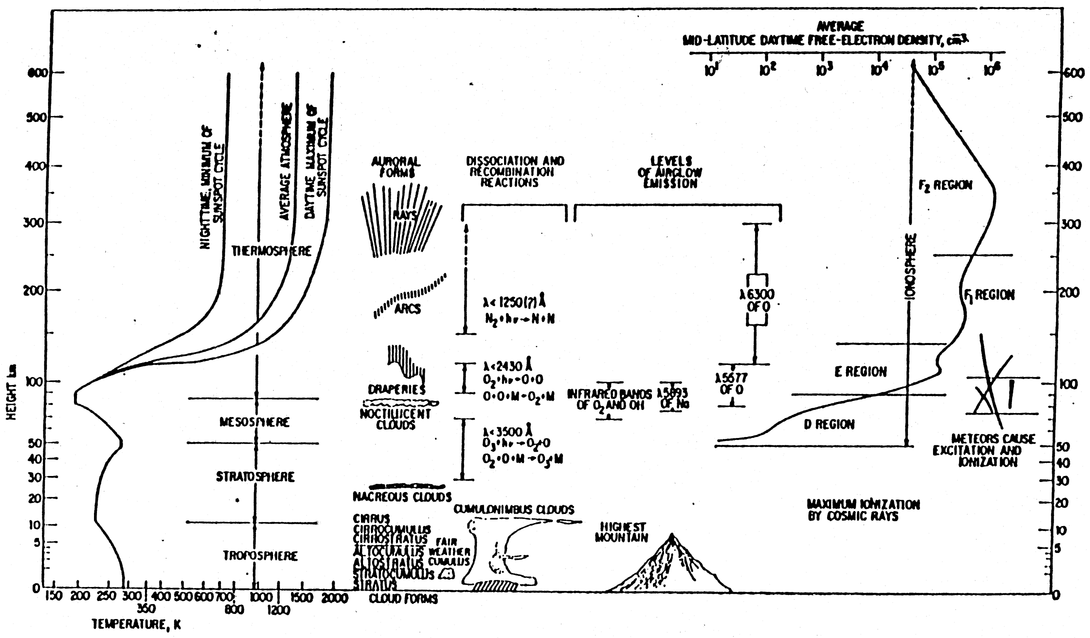


Figure 3.9: The atmosphere of the Earth, taken from [Jon02]

name	frequency	resolution	coverage	comments
<Cane78>	30.0MHz	11°	whole sky	made up from several maps at different frequencies, digitised by Huiyu Tao [Tao04].
GEETEE	34.5MHz	0.7°	nearly whole sky	generated from observations of the GEETEE telescope (India) during one day

Table 3.1: Summary of discussed sky maps

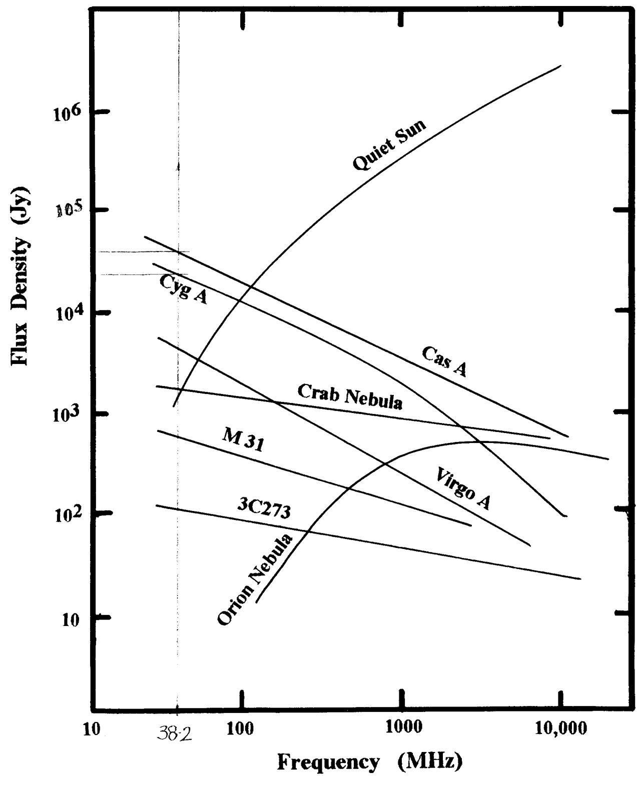


Figure 3.10: Bright Radio Sources, taken from [BGS97, p. 110]

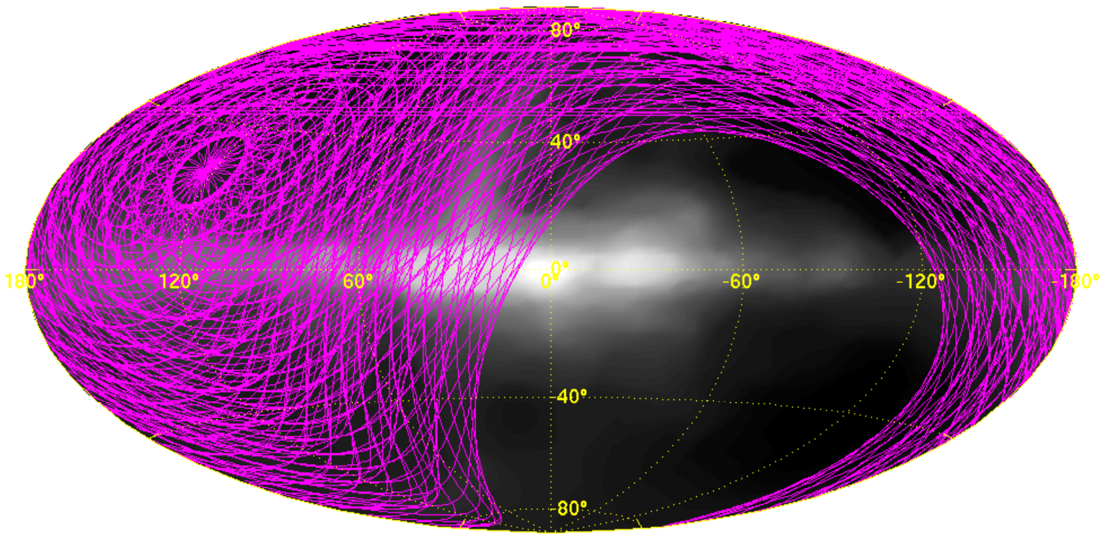


Figure 3.11: The part of the sky that affects ARIES/IRIS

3.6.8.1 Cane's Sky Map

This sky map at 30MHz was constructed by H. V. Cane in 1978 from the results of 4 separate surveys [Can78]. These were the 38MHz survey of Milogradov-Turin and Smith (1973) [MTS73] and the 10MHz survey of Caswell (1976) [Cas76] for the northern hemisphere and the 30MHz survey of Mathewson et al. (1965) [MBC65] and the 10MHz survey of Hamilton and Haynes (1968) [HH68]. Some data has been included from an earlier 13MHz survey of Cane (1975) [Can75].

Scaling the component maps to a common scale was not always straightforward. For example, a comparison of temperatures at 30MHz with those at 38MHz for an overlapping region of the two surveys show that the 30MHz temperatures are much greater than the 38MHz values, even after allowing for the difference in frequency. This comes as no surprise, as we have already mentioned above (section 3.6.5) that the temperature of galactic radiation increases as the frequency decreases.

This map was digitised by Huiyu Tao [Tao04], see figure 3.12. This map has been recorded with beam widths better than or equal to 11°. This is just about sufficient for IRIS simulations, but does not seem adequate for ARIES simulations since, as mentioned in section 3.6.4 above, it does not contain enough detail for (simulated) beams narrower than this native resolution.

3.6.8.2 GEETEE 34.5MHz Sky Map

This survey was made at 34.5MHz using GEETEE, the low-frequency telescope at Gauribidanur [DU90]. This telescope was used in the transit mode, and by performing one-dimensional synthesis along the N-S direction the entire observable sky was mapped in a single day. This survey covers the declination range from -50° to $+70^\circ$ and the complete 24 hours of right ascension. The sensitivity of the survey is 5 Jy/beam.

The worst resolution is 42 arcseconds which is easily sufficient for ARIES simulations.

As can be seen in figure 3.13, the survey does not cover the whole sky. For our simulations, these areas have been filled with data from Cane's sky map. As the missing data is in a lowly structured region of the sky, the fact that the filled-in data has a much coarser resolution does not seriously affect the simulation results.

The actual digital data [DSS95] for this sky map was provided by the NCSA Astronomy Digital Image Library [ADI].

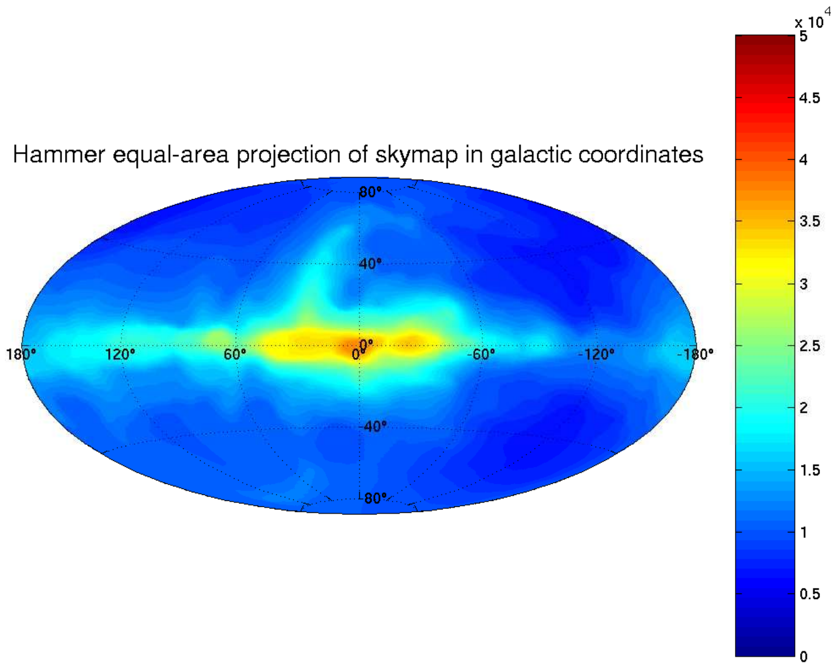


Figure 3.12: Sky temperature in K as mapped by Cane's sky map

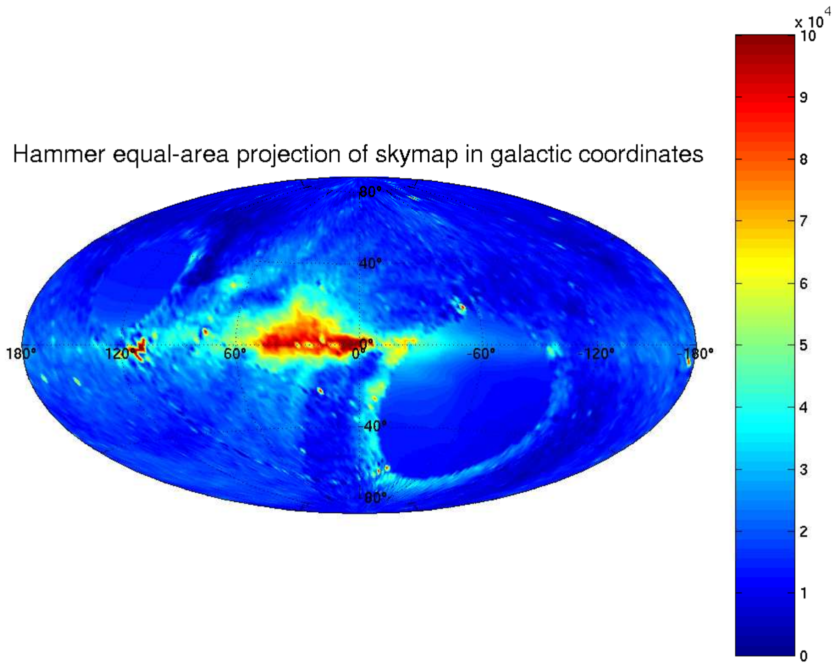

Hammer equal-area projection of skymap in galactic coordinates
80°
40°
0°
-40°
-80°
180°
120°
60°
0°
-60°
-120°
-180°
x 10^4
10
9
8
7
6
5
4
3
2
1
0

Figure 3.13: Sky temperature in K as mapped by the GEETEE sky map. Extremely bright spots are cropped at $10 \cdot 10^4$K

3.7 Summary

This chapter introduced the main types of riometers (Relative Ionospheric Opacity Meters), sensitive radio receivers connected to antenna systems of varying complexity. Riometers measure absorption, i.e. how cosmic radiation is being absorbed by the Earth's ionosphere. As well as being an interesting topic for study by themselves, these long-term continuous coverage datasets provide important background information for a wide range of scientific applications.

Sky maps and radio stars help to understand, simulate and verify the signals received by riometers. The sky maps and radio stars introduced in this chapter will be used for these purposes in later chapters.

Chapter 4

Functional Simulation of ARIES

In this chapter we introduce a set of programs to simulate the data flow through a Mills Cross type system (see chapter 2, section 2.4) from source to the final beam output. These programs (and the results they produce) help to deepen the understanding of the working principle of a Mills Cross type system such as the one used for ARIES.

The simulations discussed in this chapter will also enable us to examine the signals inside the system at various stages, providing test data even before any hardware has been built. This will help to verify that the Mills Cross approach will indeed work as expected and that the suggested approach is capable of delivering results as expected.

The fact that this simulation is done at signal level implies that it is not possible to simulate long periods of time due to the amount of processing power and storage space required. For the same reasons, the simulation cannot be carried out in real-time, and there is a practical limit to the number of sources that can be simulated. Chapter 5 will introduce a different simulation that is geared towards determining estimates for the required integration time in a realistic situation, but the simulations in chapter 5 will not simulate the whole reception process but only the final cross-correlation stage. In particular, the simulations presented here include knowledge about the direction of incoming signals and about beamforming, all of which are details that are beyond the scope of the simulations in chapter 5.

4.1 Data Flow

See figure 4.1 for a general description of the data flow through the simulation. The ‘magnetic disk’ symbols represent data that is immediately accessible as files on the hard drive whereas the rectangular boxes describe a data source or data processing step.

The whole simulation assumes a fixed operating frequency (38.2MHz by default, though this can be changed easily, see section 4.2). All signals are oversampled 32 times (again, this can be changed by modifying the program source code). Where necessary, linear interpolation is performed, see the description in section 4.2. The following sections describe each stage of the simulation in more detail. They relate directly to the description of the beamforming process for a Mills Cross in chapter 2, section 2.4.

4.1.1 Reception

This stage simulates the signal path from far away signal sources (representing incoming cosmic radiation from the galaxy as well as the cosmic radiation background) down to reception through an arbitrary number of aerials on the ground, see figure 4.2. The signal sources can be random noise sources of different bandwidths as well as simple sinusoidal or step sources (the latter being mainly useful for program testing purposes). All noise sources are located on a (hemi-)sphere centred on the centre of the antenna system (see section 4.1.2), which forms the origin of the model's Cartesian coordinate system. The ionosphere is indirectly taken care of by adjusting the signal intensity from each noise source as required. Other ionospheric effects apart from signal attenuation (e.g. scintillation, variable delays) are not taken into account for this model, since simulating ionospheric propagation is not the goal of this simulation, but rather the understanding of the beamforming process for a Mills Cross type antenna system.

The signals from these sources are then received by aerials on the ground. These aerials can be simulated at arbitrary locations (see section 4.2 for details). For the discussions presented in the sections to follow, we use the layout of a 32+32 element Mills Cross antenna just like the real ARIES antenna layout. We use aerials with isotropic radiation patterns, since the beams are mainly influenced by the array factors, not by individual element radiation patterns (see chapter 2). Each aerial receives the sum of all the signals coming from all different sources, each one delayed appropriately depending on the relative position of aerial and signal source, as described in section 2.3.2 in chapter 2 (see especially figure 2.1).

The composite signal from all the sources looks different to each aerial due to the fact that each aerial is located at a different location. The digital representation of this signal is stored in a separate file for each aerial. This will result in n files for n simulated aerials ($n = 64$ in the discussed case[1]). Each file contains the ASCII representation of one sample per line. The simulation is usually run for a given number of samples s, resulting in n output files with s lines each.

These files are by default called `aerial<n>`, n being the aerial number. These files represent the output of stage 1 (the reception stage) of the simulation and can either be examined manually or fed into stage 2 (the beamforming stage), see below.

4.1.2 Beamforming (Fan Beams)

The presently implemented beamformer for this model is the equivalent of a 32 port Butler Matrix [BL61], applying appropriate phase shifts (delays) to the 32 input signals before adding them up as described in section 2.3.4 (see especially equation 2.6 for $N = 32$). The input signals from one arm of the array (say the W-E-arm, represented by the files `aerial01` to `aerial32`) are fed into one instance of the beamformer, which in turn delivers the signals received by each beam in the output files `beam01` to `beam32`, see figure 4.3. An identical copy of the beamformer is used to process the input signals from the other arm (the S-N-arm).

At the end of the beamforming stage, the simulated signals received by each of the 64 ($= 32+32$) fan beams can be examined in the files `beam01` to `beam64`, and/or be fed into the third and final stage (cross-correlation) as described in section 4.1.3 below.

4.1.3 Cross-correlation and Integration

This stage combines the process of cross-correlating the 64 fan beams with each other (resulting in the formation of — theoretically — 1024 pencil beam signals as described in section 2.4.2.2) and integrating the power of these resulting signals. The cross-correlation and integration processes were combined because of the huge amounts of data being produced at the cross-correlation stage (one data stream for each pencil beam). A similar approach was later taken in the actual design of the final system, with the FPGA hardware performing the cross-correlation and integration steps before passing the data on to the ARCOM software (see chapter 8).

The cross-correlator (figure 4.4) reads its input data from the files `beam01` to `beam64` gener-

[1]Note that the actual ARIES antenna array actually only consists of 63 aerials, one aerial being shared between the two arms of the Mills Cross. For simplicity, and to make the simulations more generic, we build the simulated array from two arms of 32 aerials each, with aerial 15 of the South-North arm being exactly co-located with aerial 15 of the West-East arm.

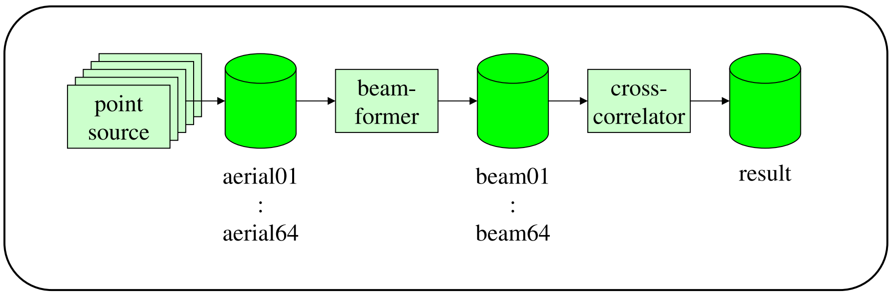


Figure 4.1: Model data flow (summary)

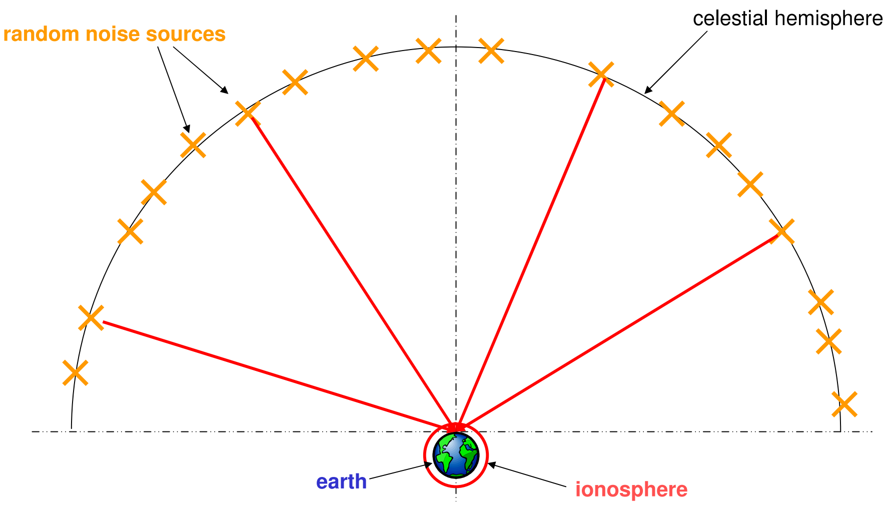


Figure 4.2: ARIES system model, stage 1: Reception

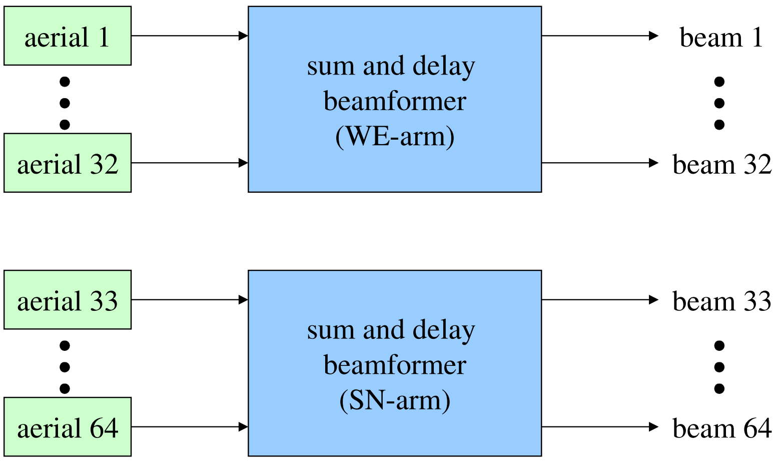

aerial 1
aerial 32
sum and delay beamformer (WE-arm)
beam 1
beam 32
aerial 33
aerial 64
sum and delay beamformer (SN-arm)
beam 33
beam 64

Figure 4.3: ARIES system model, stage 2: Beamforming

ated during the previous (fan beamforming) stage. It cross-multiplies all the signals with each other as shown in figure 4.4 and integrates the multiplication results for each beam. The result of this stage is an array of 1024 ($= 32 \times 32$) numbers in relative linear power units, representing the power received by the 1024 pencil beams. Note again that not all of these beams are physically meaningful, since not all fan beams actually overlap. See the results section (4.3) for a visual explanation of this effect. Also note that the so-called fan beams are not flat (planar) fans but rather have the shape of cones (see for example figure 2.10 in chapter 2).

4.2 Implementation Details

The three stages of the simulation are implemented by three separate C++ programs. Data is passed on from one stage to the next through simple text files as described in section 4.1 above. The whole simulation can be run automatically by invoking a simple shell script called `run` (see appendix E.1), which in turn makes sure that all binary files are up-to-date and then sequentially invokes each stage of the simulation. The following subsections describe the internal structure of each of the three programs. The program for stage 1 (reception) is by far the most complex piece of software in this simulation, since simulating reception requires detailed knowledge about all noise sources and antenna elements as well as of all simulation parameters including operating frequency and desired duration of the simulation. The programs for stages 2 (fan beamforming) and 3 (cross-correlation and integration) are relatively straightforward since they do not require any knowledge about how their input signals have been generated. Stage 3 is basically just a simple loop, cross-multiplying and integrating all input signals.

4.2.1 Reception: `model`

The main `model` program performs the following steps:

1. Instantiate a **CCommandLineOpts** object to parse its command line arguments. **CCommandLineOpts** itself encapsulates calls to the standard **popt** library [JT98, Tro] for command line argument handling.

2. Instantiate a **ModelMasterControl** object.

3. Initialise the **ModelMasterControl** object with the given command line arguments. During initialisation the **ModelMasterControl** object will create all sources and aerials as

specified by the command line arguments. See below for details.

4. Hand over control to the **run()** method of the **ModelMasterControl** object. See description below.

Table 4.1 shows all available command line arguments for the `model` program. This table has been produced by calling `model` with the `--help` argument.

Figure 4.5 is a class diagram of the `ModelMasterControl` class and related classes.

4.2.1.1 ModelMasterControl::init()

On call to **init()**, the **ModelMasterControl** first sets some basic simulation parameters such as sampling rate and operating frequency, all defined in one separate include file called `modeldata.h`. It then creates all aerials that together make up the Mills Cross. Each aerial is represented by a **CAerial** object. **CAerial** objects have the capability of autonomously receiving signals from sources, see the description of the **run()** phase below.

Secondly, **init()** creates all the sources to be used in the simulation by calling the **createSources()** member function, which in turn parses the `.source` file that contains direction and amplitude of all sources. See appendix E.2 for an example `.source` file and explanation of syntax. `.source` files do not necessarily need to be generated manually, in fact some PERL [Wala, WCS96, Hie95] scripts have been written to generate `.source` files for certain cases, some of which are presented in the results section below (section 4.3). After the call to **createSources()** returns, the **ModelMasterControl** owns a number of **CBrainySource** objects. Each **CBrainySource** object represents a signal source at a given location. It has the capability of returning properly delayed signals depending on the relative position of source and receiving aerial through the use of a **CDelayBuffer** object. The basic ('dumb'[2]) signal source itself is represented by a derivative of **CNoiseSource**. For clarity, only the **CNoiseSource** superclass is shown in figure 4.5.

CNoiseSource objects and derived/related classes are described in more detail in chapter 5, as they are also used in the integration time estimations presented there. See especially figure 5.1 for a class diagram of **CNoiseSource** and its derived classes.

For this discussion, it is sufficient to know that any given **CNoiseSource** object represents

[2]We call this kind of source 'dumb' since it does not know anything about its location and can only issue a continuous stream of samples. This is different to **CBrainySource** objects which contain a **CNoiseSource** but are also conscious of their location and of propagation delays.

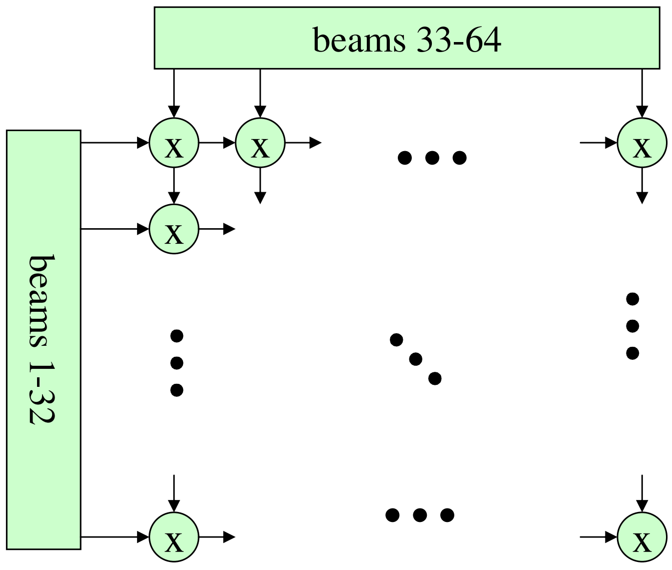


Figure 4.4: ARIES system model, stage 3: Cross-correlation and integration

option	description
-f, –sourcefile=filename	source definition file
-s, –nrofsamples=nr. of samples	Number of Samples to calculate
-n, –noisesourcetype=type	Noisesource: 1 = sine, 2 = wideband random, 3 = narrowband random (CustomIIR), 4 = step
-c, –coefficients=filename	filter coefficients for type 3 filters
-?, –help	Show this help message
–usage	Display brief usage message

Table 4.1: Available command line arguments for `model`

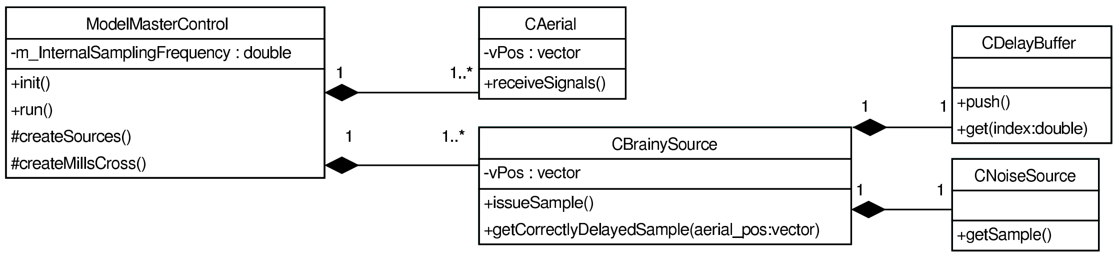


Figure 4.5: Class diagram for ARIES model

a certain type of noise source (e.g. a random noise source with a certain bandwidth) and will advance in time by one step and return the current signal amplitude on each call to **getSample()**.

4.2.1.2 ModelMasterControl::run()

The **run()** member function is one large loop. For each iteration, it firstly triggers each source to issue one more sample by calling the respective **issueSample()** member function. **issueSample()** will in turn retrieve a sample from the 'dumb' **CNoiseSource** and store it inside the **CBrainySource**'s internal **CDelayBuffer**.

ModelMasterControl will then ask each **CAerial** to receive the combined signals from every source by calling **receiveSignals()**. **receiveSignals()** will call **getCorrectlyDelayedSample()** for each individual source. **getCorrectlyDelayedSample()** is able to calculate the required delay based on the position vector of the receiving aerial and the direction vector of itself (the source). It will return the sample by looking it up in its **CDelayBuffer** which is essentially a continuously updated look-up table (LUT) with linear interpolation. Linear interpolation enables the **CDelayBuffer** to return sample values for every moment in time, even if they are not multiples of the sampling period. In this simulation there is no need to go for higher-order interpolation mechanisms, since the internal sampling rate used throughout is (by default) 32 times the operating frequency of ARIES, which is 16 times above the (required) Nyquist rate.

4.2.2 Fan Beamforming: `beamform`

The beamforming stage is implemented by a single C++ program called `beamform`. It takes one command line parameter specifying the 'starting aerial number' *n* (see below). `beamform` implements a sum- and delay beamformer for generating the same beams as the ones that would be produced by a 32 port Butler Matrix. It makes use of the **CDelayBuffer** class described above to delay samples from the aerials appropriately.

The general sequence of processing steps is as follows:

1. Get one set of samples (one line from the files `aerial<n>` until `aerial<n+31>` each).

2. For each beam direction: Sum the properly delayed samples and store the result in file `beam<m>` where *m* stands for the beam number (plus an offset of *n*).

3. Repeat until no more samples are left in the input files.

When `beamform` has finished, the user ends up with (in case of the ARIES simulations presented here) 64 files (`beam01`–`beam64`) representing the signals received by each of the fan beams. Again, these files can be examined directly or passed on into the next stage, cross-correlation and integration.

4.2.3 Cross-correlation and Integration: `xcorr`

A third C++ program has been written to implement the cross-correlation and integration stage. This program is called `xcorr`. It takes the signals from the fan beams and cross-correlates them to produce the pencil beams. Due to the amount of pencil beams (theoretically 1024), the resulting signals are not stored in files but integrated straight away, so that the result of this stage is a matrix of 1024 values representing integrated cross-correlation results. The basic algorithm is represented in full below, as it is at the heart of the Mills Cross working principle:

```
while( count < NrOfSamples )
{
  // read 1 sample from each beam in the S->N arm
  for( i=0; i<32; i++ )
  {
    ifile[i+32] >> SN[i];
  }
  // read 1 sample from each beam in the W->E arm
  for( i=0; i<32; i++ )
  {
    ifile[i] >> WE[i];
  }
  // cross correlate samples
  for( j=0; j<32; j++ ) // S->N
  {
    for( i=0; i<32; i++ ) // W->E
    {
      result[j][i] += SN[j] * WE[i];
    }
  }
} // while
store_result();
```

The last line calls a function **store_result()**, which stores the **result[32][32]** matrix in a plain text file called `result`. This file can then be read by (for example) MATLAB to generate plots of the simulation results. Some results are presented in the next section.

4.3 Results

Figure 4.6 is an example of a basic simulation. The `.source` file used for this simulation is shown in appendix E.2. This and subsequent figures use MATLAB to map the 32×32 **result** matrix onto a hemisphere, thereby directly visualising the beam pointing directions. Each (distorted) square represents one pencil beam that was generated during the cross-correlation and integration phase as described in section 4.2.3. The red pin heads are a means of visualising the directions of the sources used for the respective simulation as defined in the respective `.source` file. This allows the observer to immediately verify whether the simulation is working correctly and which input parameters (i.e. source directions) have been used.

Note that this way of displaying results is very different from the beam pattern plots presented in chapters 6 and 7. The plots in this chapter give no heed to beam patterns, they simply project a square result matrix onto a hemisphere to visualise the main pointing direction of the beam power values represented by the matrix. This is simply a convenient and less distorted way of visualising results for many beams in one figure. It is physically correct in that the main pointing direction of all beams can immediately be seen in the plots as the centre of the respective (distorted) square.

This visualisation also once again demonstrates the fact that some of the pencil beams do not actually exist, namely all the beams that are represented by squares that do not lie within the footprint of the hemisphere used for visualisation.

A colour scale from 0 to -20dB is used throughout this section unless otherwise noted. All power values are therefore relative to the respective maximum value.

4.3.1 Three Sources

The three sources simulated for figure 4.6 are of equal power. The source at the zenith can immediately be seen to equally affect the four pencil beams next to the zenith. Note that there is

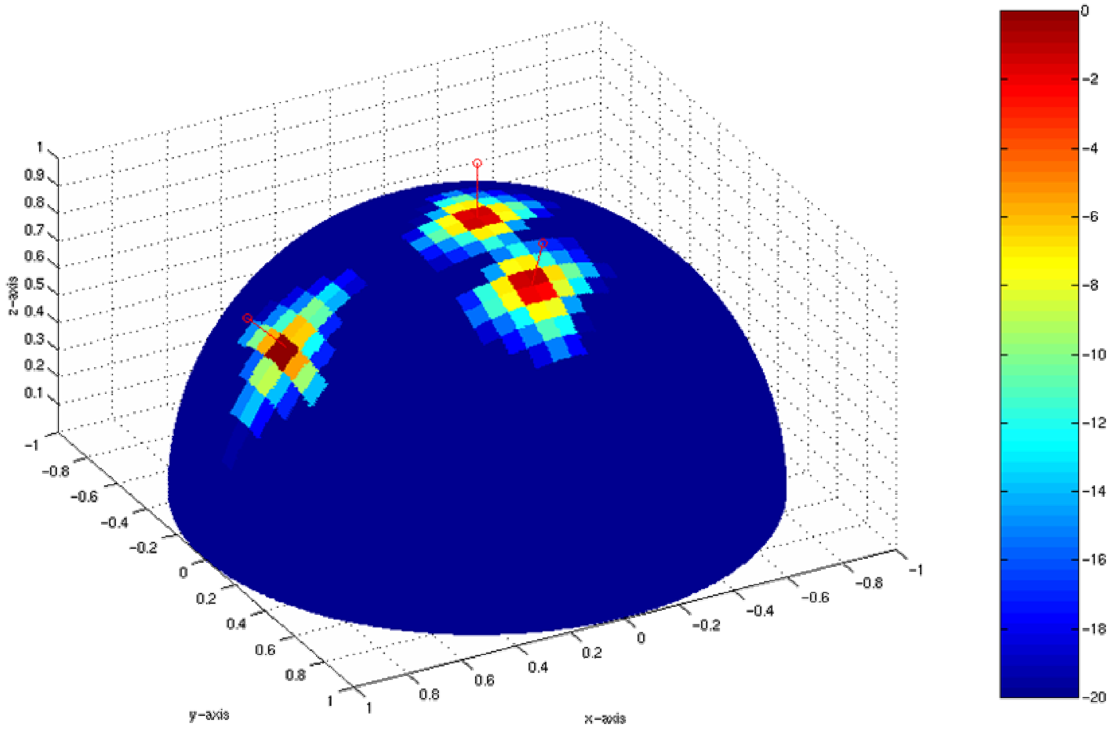


Figure 4.6: 3 sources, 4×10^5 samples

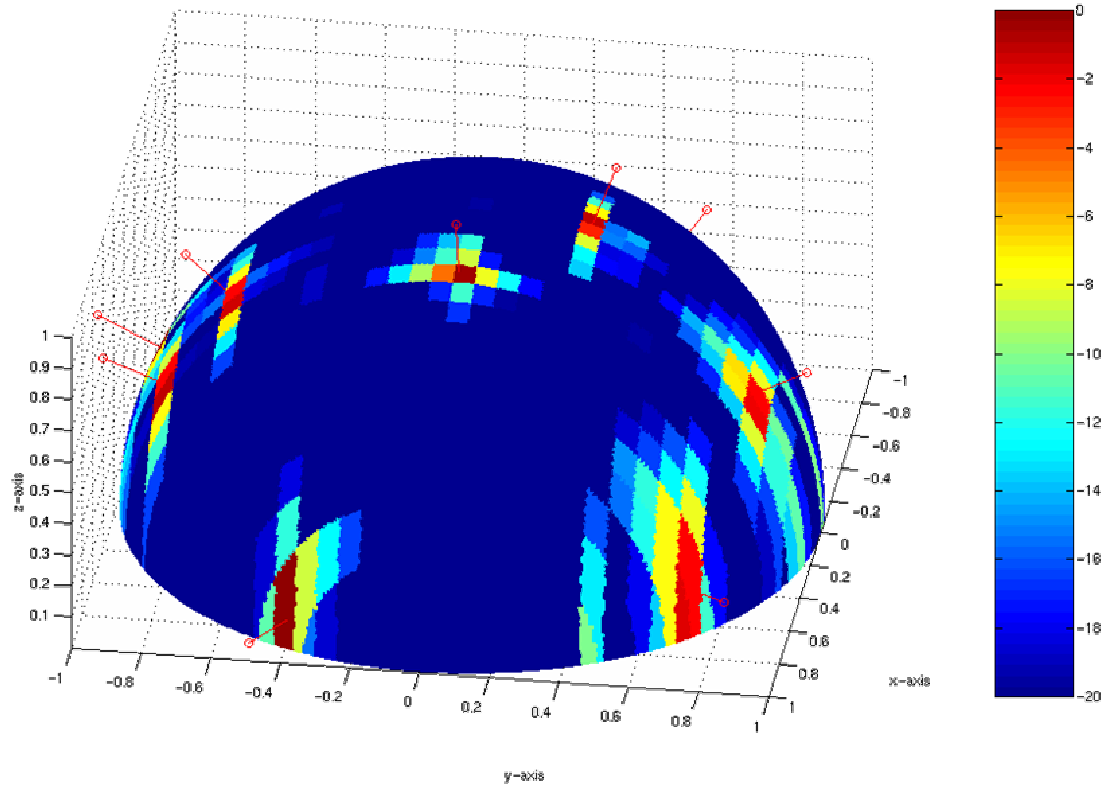
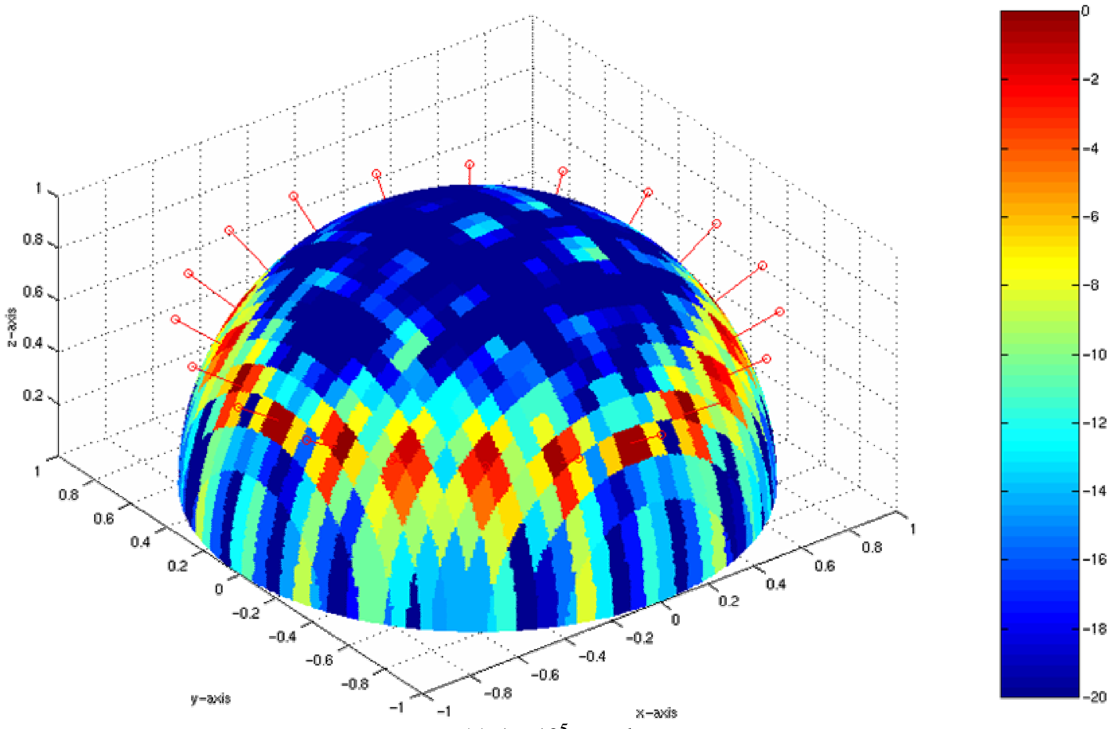
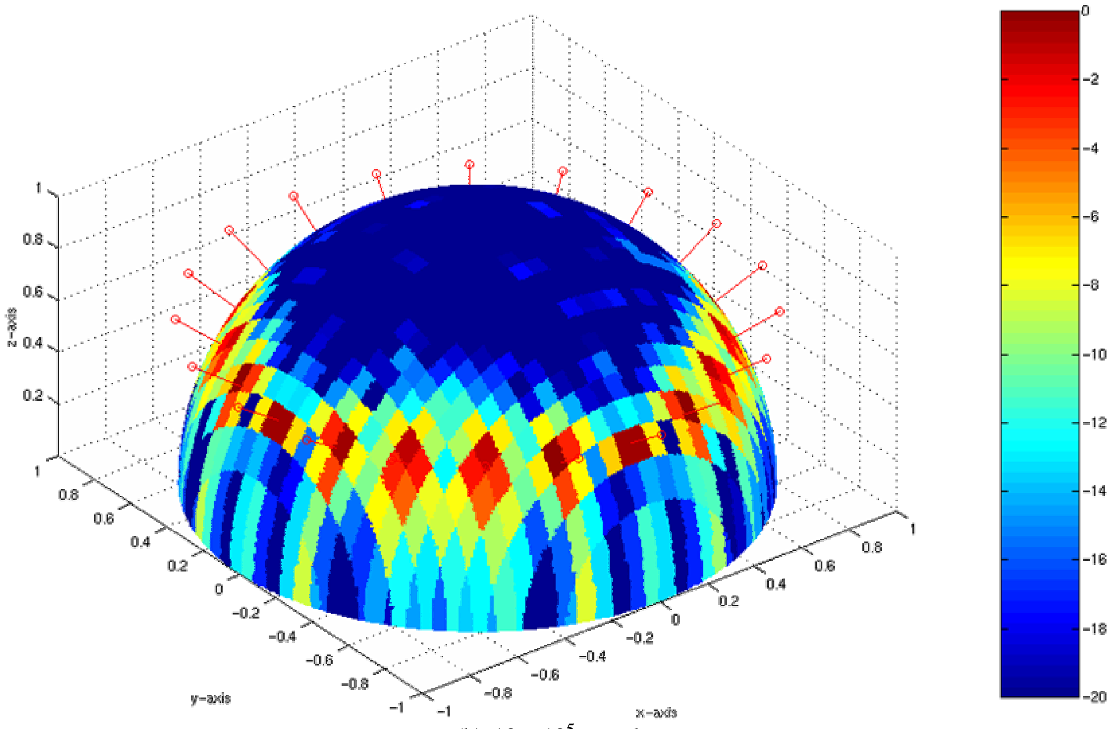


Figure 4.7: 10 sources, 4×10^5 samples



(a) 1×10^5 samples



(b) 10×10^5 samples

Figure 4.8: The same 20 sources, different integration times

no zenithal beam with a 32 port beamforming matrix such as the one used in this simulation.

The source at 45° from zenith towards the positive x-axis is still exactly in the middle between two pencil beams in the y-direction, but not exactly between two beams in x-direction. This can be seen by the already much weaker signal seen by the beams 'above' the pin.

The source down the positive y-axis is again quite well-centred between four pencil beams, slightly biased towards the more zenithal ones.

4.3.2 Ten Sources

Figure 4.7 is an example of ten sources randomly distributed around the hemisphere. It clearly shows the effect of different source directions on the responses from the individual pencil beams. Also clearly seen can be the distortion of the beams that occurs towards the horizon. The ten-source example demonstrates that given a long enough integration time, the Mills Cross can indeed resolve multiple sources from different directions.

Note again that all figures presented here do not show beam projections in the sense of the beam projections presented in chapter 6 (discussing the high-level riometer simulation toolkit RIOSIM), but simply a visualisation of the 32×32 result matrix and the directions associated with each value in the matrix.

4.3.3 Long/Short Integration Time

The panels in figure 4.8 were generated using exactly the same `.source` file, defining a ring of 20 random noise sources of equal power around the zenith. However, for the simulation result in panel (a), only 1×10^5 samples have been integrated, whereas panel (b) shows the result for an integration length of 10×10^5, i.e. 10 times as long.

Comparing the two panels, one clearly sees the influence of integration time on the measurements (or the simulation results in this case). Short integration times result in a 'spotty' (noisy) result, note the many erroneous signals around the top cusp of the hemisphere with quite high power values of up to around -10dB. As one moves towards longer integration times, the result gets less and less noisy, panel (b) already showing no erroneous values greater than about -16dB.

The effect of integration time on the accuracy (noisiness) of the result will be investigated in more detail in chapter 5.

4.3.4 Phase Centre Offset

Figure 4.9 shows a problem that was encountered during the first tests of the simulation software. In this simulation, 120 sources of *equal* power are distributed in a circle. The simulation, however, yields very different source intensities. For this particular case, intensities have been plotted directly as the linear signed values contained in the result matrix (not their absolute value in dB as in the other figures). It can clearly be seen that the cross-correlation yields negative results in some areas, positive results in other areas, and there are four locations on the circle where no power is detected at all.

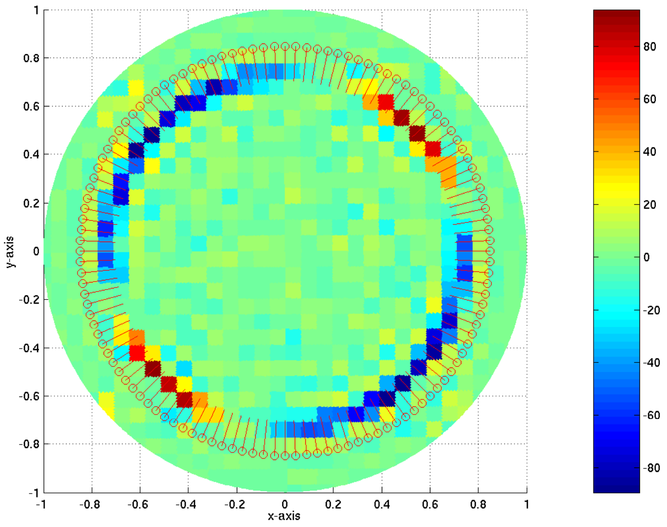
Further investigation[3] finally led to the conclusion that differences in the phase centres (an arbitrary point in space used as a reference from which all relative phase offsets for signals received from the antenna(s) are calculated) of the two arms that made up the Mills Cross in this simulation were responsible for the effects. If the phase centres of the two arms do not match, then, depending on the pair of fan beams being cross-correlated for a given pencil beam, the two signals from the two fan beams have a certain phase difference. The cross-correlation of the two signals will give different results depending on this phase difference. In particular, if the phase difference is close to $180°$, the result will be negative. And if the phase difference is about $90°$, the result of the cross-correlation will be close to zero. Although this issue was easily resolved by ensuring common phase centres for the two arms, it was to be a forerunner of similar effects encountered during initial experiments with the real ARIES system (see chapter 9).

Also in figure 4.9, note panel (a). This shows a 2D top-down view of the same simulation result and therefore once again visually illustrates how the result matrix gets projected onto the hemisphere in these figures. This panel once again makes clear that certain parts of the result matrix, namely the four corners, cannot be associated with a physically meaningful direction of arrival.

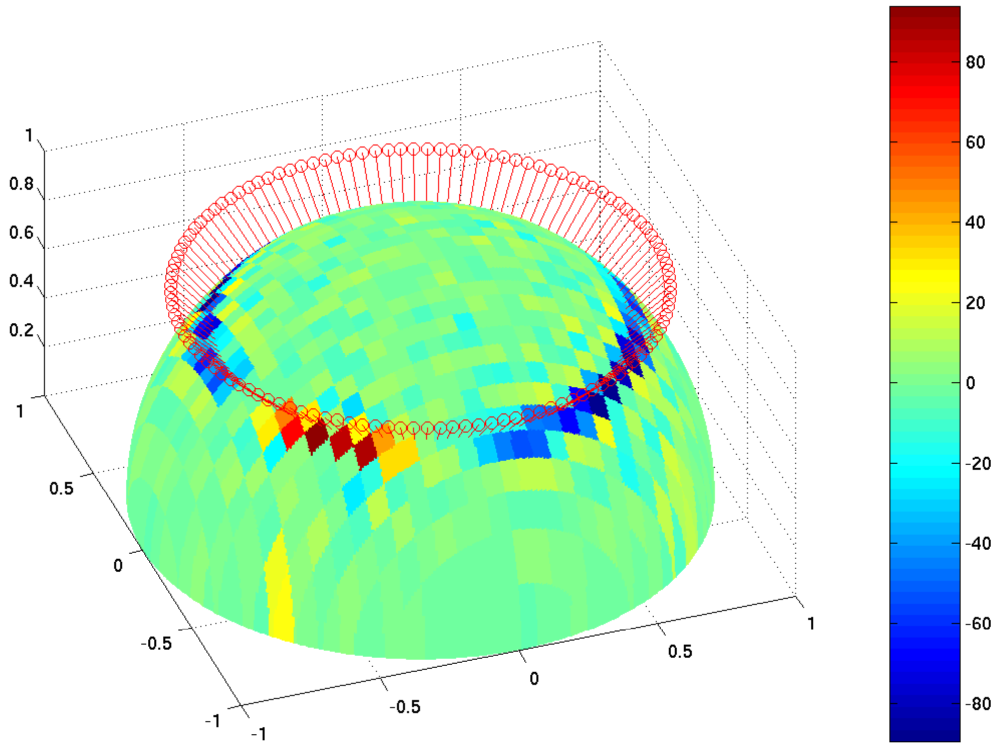
4.3.5 Ghost Images in the Sinusoidal Case

Figure 4.10 illustrates another phenomenon associated with the cross-correlation technique employed in a Mills Cross receiving system. Panel (a) is the result of a simulation run simulating two random noise sources at the positions indicated by the pin heads. As expected, the respective pencil beams close to the direction of the source give high signal readings. Pencil beams

[3]This is one occasion where the flexibility of having different types of noise sources was really useful. In particular, a **CStepNoiseSource** allows one to trace a phase front through the whole system.

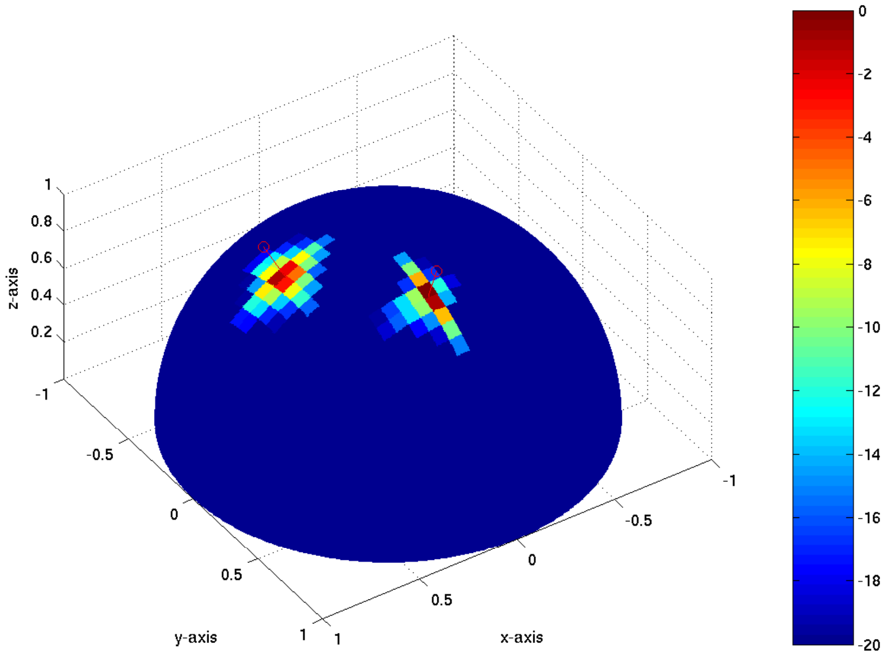


(a) top-down view

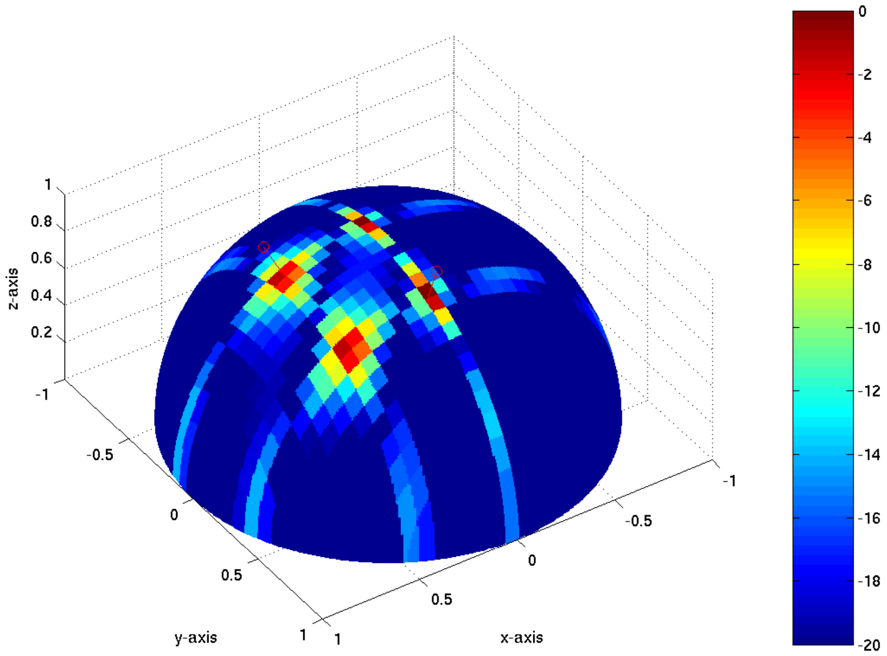


(b) 3D view

Figure 4.9: Phase centre issues



(a) Two Random Noise Sources



(b) Two Sinusoidal Sources

Figure 4.10: Effect of pure sinusoidal sources

pointing away from the sources give negligible readings.

Panel (b) shows the same simulation, but this time employing sources that emit a perfect sine wave. Not only do we get a strong reading in the directions we expect (as in panel (a)), we also get (in this case two) ‘ghost images’ wherever two fan beams pointing at both sources overlap.

More precisely: Let us assume that a pencil beam A is formed by cross-correlating fan beams F_{aX} and F_{aY} (i.e. pencil beam A is the overlapping area of fan beams F_{aX} and F_{aY}), and a pencil beam B is formed by cross-correlating fan beams F_{bX} and F_{bY}. In the case of sinusoidal sources we will then get two additional (erroneous) readings at the intersection of F_{aX} with F_{bY} and at the intersection of F_{bX} with F_{aY}. This can be explained as follows: Since the signals from the two sources are identical (they are pure sine waves), the cross-correlation stage cannot distinguish between the two signal sources. In other words, the signal from one source correlates just as well with the signal from the other source as it does with itself.

This is, of course, not the case with (two or more) random noise sources, since the signals from any two random noise sources are per definition completely incoherent. The random nature of the noise received from the sky is therefore the reason that the Mills Cross works as it should and that ‘ghost images’ are not a problem in real applications.

4.3.6 ‘Negative Sidelobes’

A final interesting observation can be made by looking at figure 4.11. This is the result of simulating one single noise source at the zenith, but similar to the figures relating to section 4.3.4 the result matrix is plotted directly on a linear scale (not the absolute values on a logarithmic scale as for the other figures in this chapter). In addition to the (expected) positive sidelobes, figure 4.11 clearly shows ‘negative’ sidelobes as well. This is again related to the way the Mills Cross works. In case of the ‘negative’ sidelobes, signals from different order lobes from the two fan beams get cross-correlated, and as later considerations will show (see chapter 9), these signals inherently have phase offsets and these phase offsets get picked up by the cross-correlator. These issues will be discussed in more detail in a later chapter (chapter 9, see especially section 9.7.2 and the corresponding figures 9.8–9.10).

4.4 Summary and Conclusions

The simulations described in this chapter followed the path of the signals received by a Mills Cross antenna array from the source through aerials, beamformer and cross-correlator to the output of the integrator. Some important conclusions can be drawn from the work on these simulations and the simulation results:

- The proposed Mills Cross system works as expected in that we can determine the direction of arrival of any given incoming signal, even in the presence of signals from other sources.
- The effect of longer integration times on the noise level of the result was visualised.
- The simulation used discrete sources. To simulate a continuous (noise) background, the number of sources can be increased. The simulation shows qualitatively that it takes longer integration times to resolve signal source directions if more power is being received from off-target signal sources. Chapter 5 will quantify this observation.
- The findings from section 4.3.4 show that special care has to be taken concerning phase-related effects that are unique to a Mills Cross system and do not occur in traditional filled phased array systems like IRIS. In particular, the phase centres of the two arms of the Mills Cross need to coincide.
- The model presented is computationally intensive even for a small number of sources and very short integration times. To simulate real-life behaviour, one would need to approximate the continuous sky background as a large number of sources, and then integrate for long durations. Chapter 5 tackles this problem by using a different model (eliminating the simulation of the reception process) and showing that for purposes of determining worst-case integration times, we do not need to simulate many sources.
- Following on from the previous point, this model is especially not suitable for near-real-time simulations (e.g. simulation of the variation of received signal values during one whole day). Again, a different approach (higher-level simulations) needs to be taken. This will be done in chapters 6 and 7.

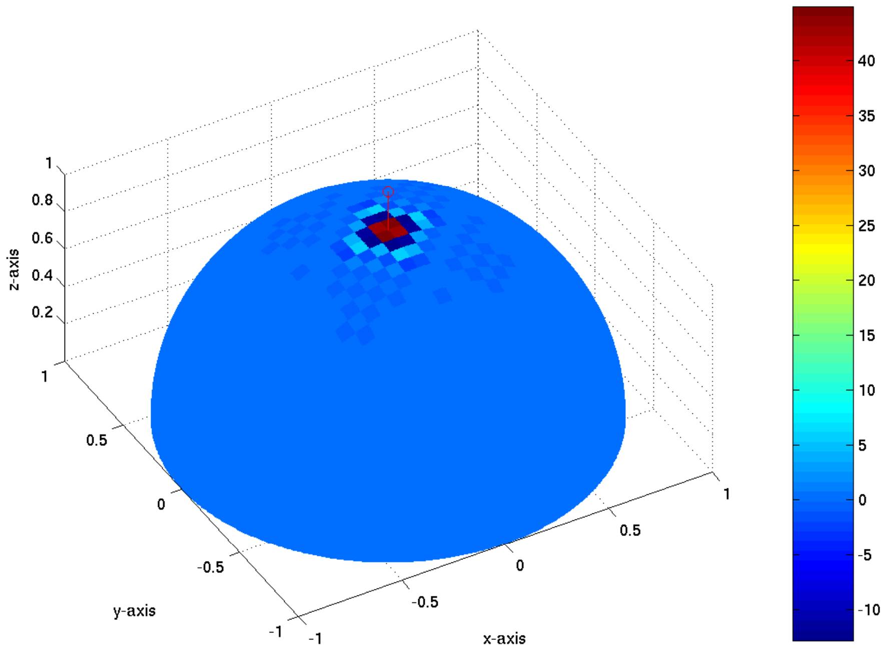


Figure 4.11: ‘Negative sidelobes’

Chapter 5

Investigations into the Achievable Integration Time

One of the main requirements for the Mills Cross imaging riometer experiment to succeed is to understand how integration time affects the precision of the result. Specifically, we want to determine the minimum required integration time that still gives us the required power resolution. Even from the very first description of the Mills Cross (see section 2.4) it is clear that the noise performance of such an array is inferior to that of a filled array. It is also clear, however, that useful results can be obtained with the Mills Cross. The aim of this chapter is to establish the minimum integration time that can be expected to give a scientifically useful precision.

The first step is to understand how different factors contribute to the required integration time. These results are then used to extrapolate a realistic estimate of the required integration time to achieve a certain instrument precision.

This chapter mainly describes a simulation approach to this problem. It is partly based on an idea that came up during Prof. Yamagishi's (National Institute of Polar Research — NIPR, Japan) visit to Lancaster University in December 2001 [Yam]. The following section 5.1 will describe the basic software structure that was used to run the simulations, section 5.2 will then present the 'Yamagishi' approach and the results of the simulations that were run. Two more mathematical approaches by Nielsen and Hagfors (both at Max Planck Institute for Solar System Research — MPS, Germany) are also presented in sections 5.3 and 5.4, respectively. There is a short summary at the end of this chapter.

Note that one goal of the work presented in this chapter was to come up with suitable operating parameters for the first experiment. The experiment and the results obtained from it are discussed in chapter 9.

5.1 Basic Simulation Software Structure

Since we want to simulate the behaviour of various parts of the ARIES system, all relevant internal signals have to be represented as digital values. As we have already seen in the previous chapter, data flow for all simulated experiments originates at some kind of radiating source in the sky. This source, or many of these sources, model the continuous radiation received from the sky, both from the cosmic noise background and from discrete radio sources like Cassiopeia, etc.

In order to get a digital representation of an analogue signal, the analog signal has to be sampled at a sampling rate which is at least twice as high as the highest frequency component of the signal. This theorem is known as the sampling theorem, the minimum sampling rate is known as the Nyquist-rate.

Even though the cosmic background noise does not have a limited bandwidth, the highest useful frequency for the ARIES project is obviously limited by the aerials and receivers of the system. So, since according to the above paragraph we cannot possibly find a finite sampling rate to digitally represent a signal with infinite bandwidth, we will model the sky noise sources as band-limited random noise sources.

This will not affect the validity of any of the obtained results, since, as stated above, the receiver part of the system will limit the bandwidth of the signals anyway, and since this part of the system is completely linear, we might as well reduce the bandwidth in the first place.

Especially, once we have band-limited noise sources, there is no need to immediately model the receivers, as we already have a digital representation of the received signal.

Since one of the important issues to investigate will be the influence of different system bandwidths on the behaviour of the overall system, we will start off by building a pool of random noise sources with different bandwidths. We can then easily use any of these noise sources in any of the models, and can therefore easily simulate the influence that a change in bandwidth has.

Technically, each random noise source is a random number generator and a band pass filter in series. In our case, this functionality is encapsulated in a class **CNoiseSource** and its subclasses, as seen in figure 5.1. Subclasses of **CNoiseSource** not only implement random noise sources, but also other kinds of noise sources. The class **CSineWaveSource** for example will prove useful later on for studying the effect of one single noise source on the system, without having to bother with long integration times due to the random noise characteristics of the test signal.

As can be seen in figure 5.1, the random noise source classes make use of a class **CFilter**, whose subclasses represent various types of digital filters.

All random noise sources are designed to run at a sampling rate of 32×38.2MHz. This is 16 times the required sampling rate (see above) for a system operating around 38.2Mhz, not taking into account any limitations in bandwidth that might enable us to downconvert the signal. By sampling at such a high rate, we can later on study the effects of reducing the sampling rate, starting from the quasi-analogue case. This will especially be done as part of the simulations in section 5.2.8.

Table 5.1 lists all noise sources that have been used during the simulations in this chapter.

Note that all noise source classes contained in table 5.1 are derived from the same superclass (figure 5.1), which means that a program using one of the noise sources does not necessarily have to know exactly which kind of noise source it is dealing with. This is a key property of object-oriented systems, known as polymorphism. It makes it easy to replace one kind of noise source by another one, without having to change program code in several different locations.

Please note that source types 1 and 2 are implemented by the class **CRandomNoiseSource**, whereas types 3 to 7 are implemented by the class **CIIRRandomNoiseSource**. Each instance of **CIIRRandomNoiseSource** can be individually associated with a filter definition file from which it creates its own **CCustomIIRFilter**. Those filters can easily be designed with existing filter design tools. In our case, they have been designed with the MATLAB `SPTOOL`, which is a part of the MATLAB Signal Processing Toolbox [Matc].

5.2 Yamagishi-Model

During his visit in December 2001, Prof. Yamagishi proposed a new, simple way of looking at the cross-correlation process that is part of the working principle of any Mills Cross type antenna array (see section 2.4). The following text firstly describes the idea, followed by detailed descriptions of the simulations that were run. At the end, it derives some important results from these simulations.

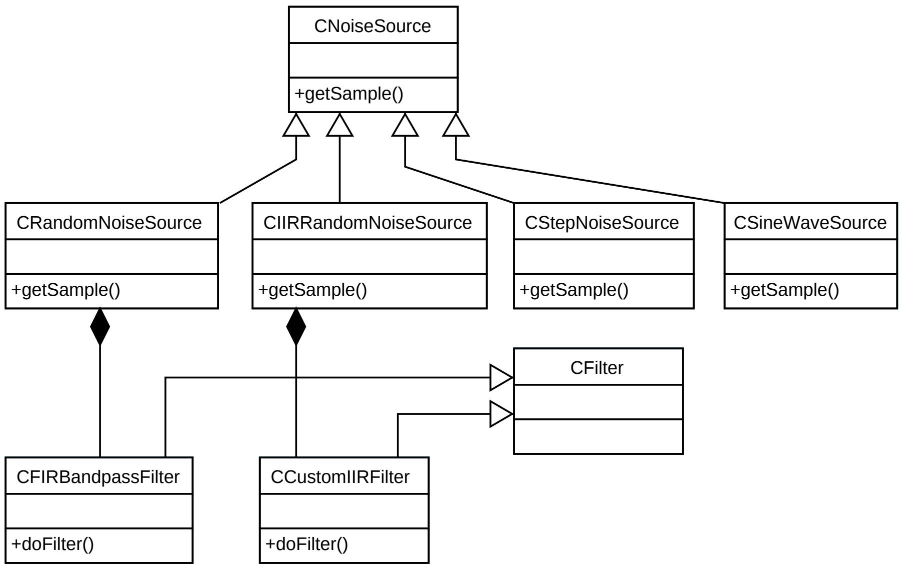


Figure 5.1: Class diagram for noise sources in Yamagishi Model

ID	implemented as class	parameter file	filter order and -type	measured bandwidth at –10dB
01	`CRandomNoiseSource`	N/A	801 (FIR)	4MHz
02	`CRandomNoiseSource`	N/A	201 (FIR)	17MHz
03	`CIIRRandomNoiseSource`	`cheb_02`	6 (IIR)	0.6MHz
04	`CIIRRandomNoiseSource`	`cheb_01`	8 (IIR)	1.0MHz
05	`CIIRRandomNoiseSource`	`cheb_03`	8 (IIR)	2.0MHz
06	`CIIRRandomNoiseSource`	`cheb_04`	10 (IIR)	4.0MHz
07	`CIIRRandomNoiseSource`	`cheb_05`	12 (IIR)	15MHz

Table 5.1: Noise sources used in ARIES simulations

5.2.1 Idea

As described in section 2.4.2, beamforming for the Mills Cross is basically done in two steps. The first step is forming the fan beams from each of the two arms of the Mills Cross. In a second step, the received signals from each of the fan beams from one arm are cross-correlated with the signals from each of the fan beams from the other arm of the Mills Cross.

The process of forming the fan beams from the two arms is well understood. Each arm of the Mills Cross acts as a linear phased array of its own.

For now, we are only interested in the cross-correlation stage of the beamforming process. The question is, how long an integration time is required to get results that satisfy a certain precision criterion.

Figure 5.2 shows the cross-correlation stage of the beamforming process for one single pencil beam. The areas j and k represent two fan beams formed by the West-East-arm and the South-North-arm, respectively. As explained above, the signals from these beams are produced by stage 1 of the beamforming process. Let us denote the time series of the signal received from fan beam j by $r_j(t)$ and the signal received from fan beam k by $r_k(t)$, respectively.

Each of these beams receives signals from an (ideally infinite) number of random noise sources. All these sources are completely independent from each other.

Both $r_j(t)$ and $r_k(t)$ therefore consist of all the signals received from all the sources within the respective beam. Please note that we are not taking into account any sidelobes for the following observations. If we denote the signals from each of the sources in fan beam j by $s_{j,i}(t)$ and the signals from each of the sources in fan beam k by $s_{k,i}(t)$, we can then write:

$$r_j(t) = \sum_{i=1}^{n_j} s_{j,i}(t) \qquad \text{and} \tag{5.1}$$

$$r_k(t) = \sum_{i=1}^{n_k} s_{k,i}(t), \tag{5.2}$$

assuming that we have n_j sources in fan beam j and n_k sources in fan beam k, respectively. As a first simplification we will assume that we have an equal number of sources in both fan beams j and k, thus $n_j = n_k = n$. As we will see in section 5.2.3, this does not affect the suitability of the model for our purpose in any way. It will, however, reduce the number of independent parameters for the model, which is a benefit from the programmer's point of view. It also makes it easier to evaluate the results.

As can be seen in figure 5.2, not all sources $s_{j,i}$ are completely independent of the sources $s_{k,i}$. In fact, all the sources within the overlapping area c are common to both fan beams, and it is the power received from these 'coherent' sources, that the cross-correlation stage of the beamforming process is supposed to evaluate. Put another way, the n_c signals from sources in area c are received by both beams, and these are the signals that we want to isolate in the cross-correlation stage of the beamforming process.

Let us denote the signals from the coherent sources within area c by $s_{c,i}$. We can then express r_j and r_k as follows:

$$r_j(t) = \sum_{i=1}^{n-n_c} s_{j,i}(t) + \sum_{i=1}^{n_c} s_{c,i}(t) \qquad \text{and} \tag{5.3}$$

$$r_k(t) = \sum_{i=1}^{n-n_c} s_{k,i}(t) + \sum_{i=1}^{n_c} s_{c,i}(t). \tag{5.4}$$

The cross-correlation stage of the beamforming process calculates the cross-correlation of the signals r_j and r_k.

$$r_p(\tau) = \frac{1}{\tau} \sum_{t=0}^{\tau} r_j(t) \cdot r_k(t) \tag{5.5}$$

In the long run, r_p is therefore a measure for the power received from within area c.

Since, in a simulation, we know all the signals $s_{j,i}(t)$, $s_{k,i}(t)$ and $s_{c,i}(t)$, not only can we investigate the signals from the two fan beams as a whole, but we can also see how these signals are made up of their components. This means we can directly 'measure' the power from area c.

A diagram that represents such a 'measurement' is shown in figure 5.3. The x-axis is in arbitrary time units and the y-axis is in arbitrary power units. There are 10 sources of equal amplitude in each fan beam, 2 thereof are located within the common area c, just like shown in figure 5.2.

The curve denoted 'rp(t)' shows the result of the cross-correlation according to equation 5.5 above. In addition to r_p, the diagram shows the power received from within area c (curve 'c'), calculated as

$$r_c(\tau) = \frac{1}{\tau} \sum_{t=0}^{\tau} \left(\sum_{i=1}^{n_c} s_{c,i}(t) \right)^2 \tag{5.6}$$

As expected, 'rp(t)' and 'c' coincide for large values of τ. The green curve, denoted as 'i', shows the cross-correlation of the two incoherent parts of the two fan beams. As expected, this

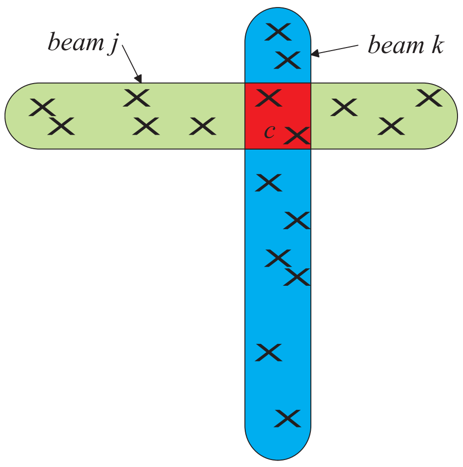


Figure 5.2: Two intersecting fan beams. Area *c* is termed the ‘coherent’ area in this chapter, that is the area that is common to both fan beams.

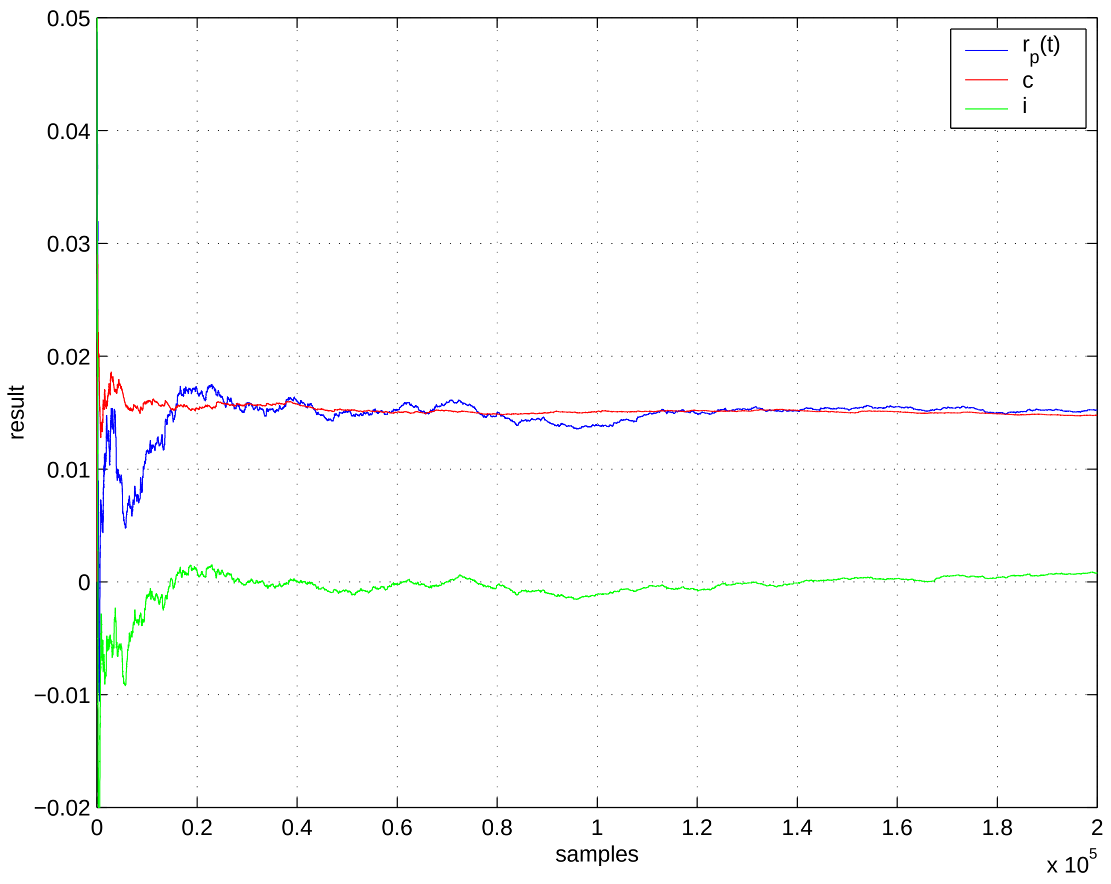


Figure 5.3: Measurement example

signal tends to 0 for long times τ, because the cross-correlated signals are completely uncorrelated.

As we can see in figure 5.3, the coherent signal reaches its final value fairly quickly, compared to the cross-correlated signal from the two fan beams. This is due to the fact that the area covered by the intersection of the two fan beams is much smaller than the total fan beam area.

Therefore, a good measurement of the accuracy of the measurement after a specific integration time really is the 'normalised' value of the received signal, defined as

$$r_{pN}(\tau) = \frac{r_p(\tau)}{r_c(\tau)} \tag{5.7}$$

Since, as derived above, r_t will tend to c_t, we expect r_{pN} to tend to 1.

If we now define a margin ε around 1, we can measure the integration time τ that is required to get the measured value r_{pN} within the range $1 \pm \varepsilon$.

An example plot of $r_{pN}(\tau)$ can be found in figure 5.4, the horizontal dotted red lines showing the margin $\varepsilon = 0.02 = 2\%$, and the vertical red line showing the time T when $r_{pN}(\tau)$ finally enters that margin.

5.2.2 Aims

Our primary aim is to get a good estimate of the required integration time for the ARIES system. Therefore, in the following sections, we will investigate how different experiment constellations affect the required integration time. We can then deduce an estimated integration time for the real ARIES system by taking into account all these results.

Firstly, we will investigate how the number of simulated sources affects the accuracy of our results. This is done in section 5.2.3. Not too surprisingly, we will find that there is no need to simulate many sources, in fact as little as 2 sources per fan beam are sufficient.

We will then show the relation between various source properties of interest and the result of the cross-correlation:

- The influence of sources with different intensities (section 5.2.4) — which is in fact equal to different sizes of fan beam-area and intersecting area.
- The influence of how close a boundary ε is set up on the required integration time (section 5.2.5).

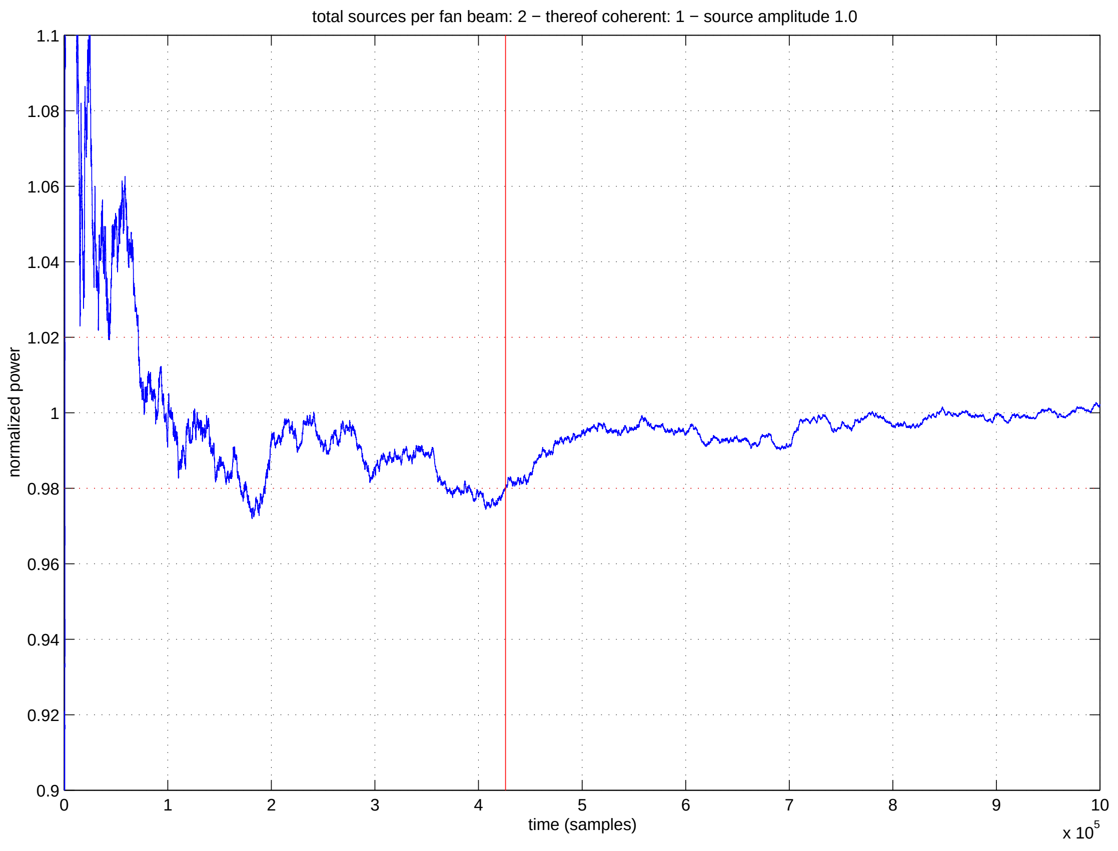


Figure 5.4: Normalised simulation results

- How long it actually takes to get 50% (80%, 90%,...) of all experiments to reach the $1 \pm \varepsilon$ boundary (section 5.2.6).
- The influence of the bandwidth of the noise sources on integration time (section 5.2.7).
- The influence of different sampling rates on integration time (section 5.2.8).

For each of the following subsections, all parameters except the one under observation are kept constant. In this way, the influence of each individual factor can be determined.

Having established all those factors that contribute to the required integration time, along with a rough quantitative estimate of how big their influence on integration time is, we can then calculate a minimum required integration time for our application. This is done in the final section 5.2.9.

5.2.3 The Number of Noise Sources

How does the number of simulated noise sources affect the result of the simulation? Do we have to simulate an infinite number of sources in order to get correct results? After all, the real sky brightness is a continuous distribution.

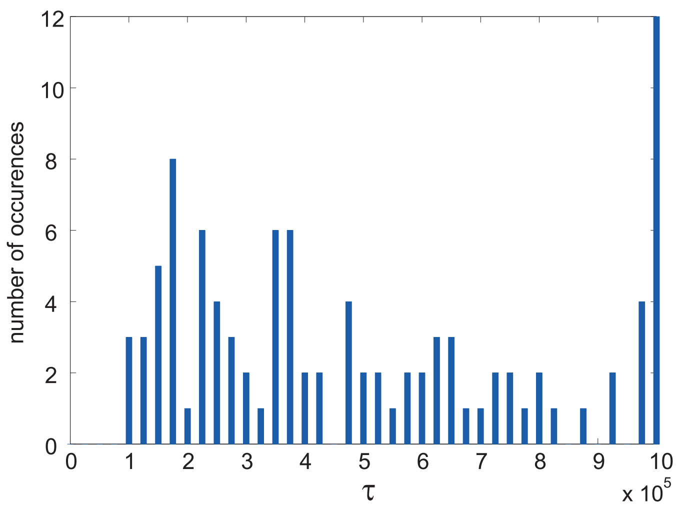
To measure the influence of the number of sources, the simulations described in table 5.2 were run. All simulations were run for 1,000,000 samples at a sampling rate of 32×38.2MHz. The `cheb_05` source type (see section 5.1) was used (source bandwidth 15MHz).

As can be seen from this table, the same fraction of power is emitted from the sources in the coherent area c compared to the power emitted from the incoherent areas.

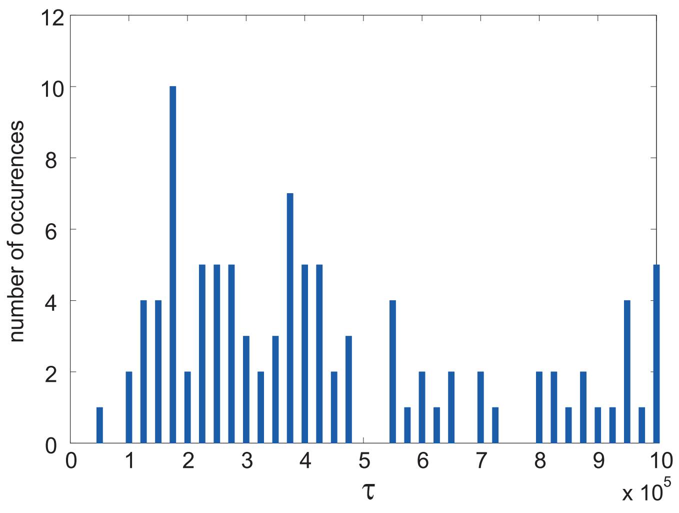
A typical result of such a simulation can be seen in figure 5.4 (discussed previously). Figure 5.5 shows the results of all runs in a graphical form (histograms). The possible range for τ was divided into 41 bins, shown along the x-axis of the diagrams. The y-axis shows how many values of τ fall into each bin. Figure 5.5 shows the results of these simulation runs: Panel (a) shows the results of simulation run 1, panel (b) shows the results of simulation run 2 and panel (c) shows the results of simulation run 3, respectively.

All histograms look similar. In particular, we find the first peak to be around $\tau = 175000$ in each case. There is no significant difference in the result of the three simulations.

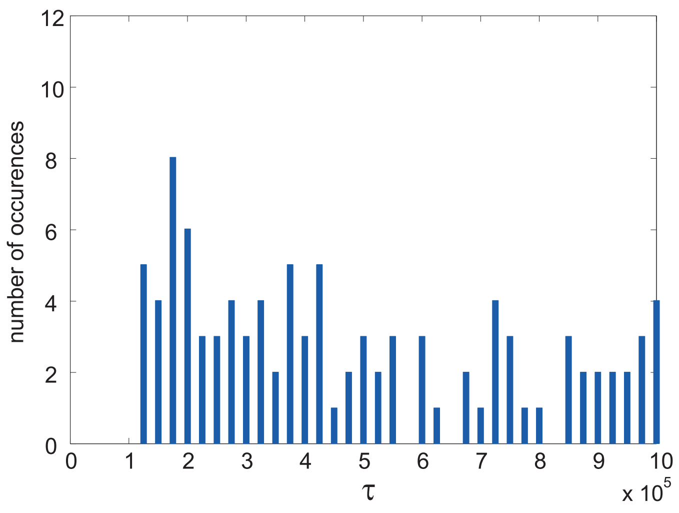
This leads to the conclusion that, rather than simulating a huge number of low power sources in order to get meaningful simulation results, it is sufficient to simulate a few 'high power'



(a)



(b)



(c)

Figure 5.5: Simulation results for different quantities of simulated noise sources. (a) run 1 (2 sources per fan beam), (b) run 2 (4 sources per fan beam), (c) run 3 (8 sources per fan beam)

sources. Knowing this, we can of course speed up any further simulations by using only a small number of sources.

5.2.4 Noise Sources of Different Intensities

We now want to find out how the ratio of power received from outside the overlapping area to power received from within that area affects the required integration time τ.

Since section 5.2.3 suggests that the results do not depend on the number of simulated sources, we run the simulations in this section and in all the following sections with only three sources:

Source 1 is unique to beam j, source 2 is unique to beam k and source 3 is the common source within the the coherent area c.

In order to be able to easily compare the different results, all of the following experiments are run according to the following rules:

- For each source configuration we run the simulation 500 times. This results in 500 values for τ for each source configuration, distributed similarly to the values in figure 5.5.
- The values $\tau_{50\%}$ and $\tau_{80\%}$, when 50% respectively 80% out of the 500 simulations reached the $\pm\varepsilon$ boundary, are stored.
- These values are then used to compare the influences of different source configurations.

With these simulations, we are especially looking for answers to the following questions:

- How does the amount of incoherent radiation affect the necessary integration time τ?
- Does the required integration time τ solely depend on the ratio of total incoherent power to coherent power, or does it make a difference if most of the incoherent power is received from within one beam?

Table 5.3 shows the simulations that were run, along with the results. Figures 5.6 and 5.7 show the results in a graphical format. Both figures show the power received from the incoherent part of the fan beams normalised to the power received from the coherent part on the x-axis. The y-axis shows $\tau_{50\%}$, the number of samples that was needed for 50% of all simulations to reach the $1 \pm \varepsilon$ boundary.

The results were divided into two figures (5.6 and 5.7) for ease of viewing.

Run	Experiment nr.	Sources per fan beam	thereof coherent
1	1–100	2	1
2	101–200	4	2
3	201–300	8	4

Table 5.2: Simulations to determine the influence of the number of noise sources, source amplitude=1.0

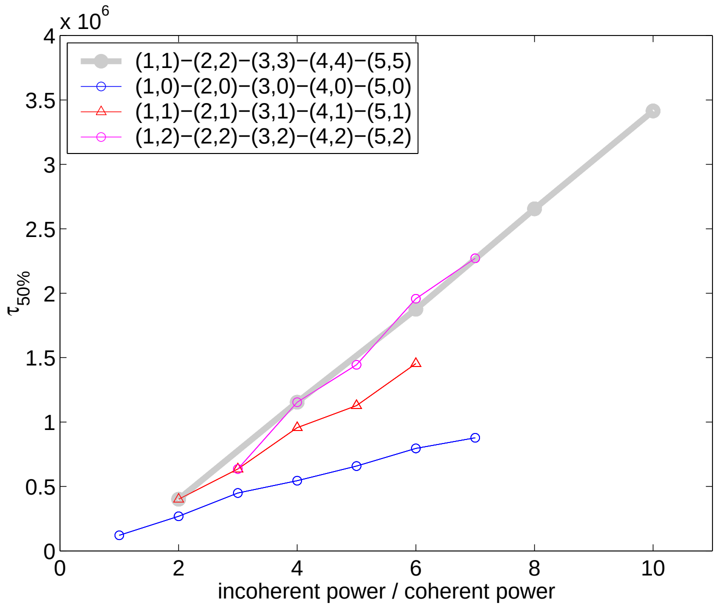


Figure 5.6: Influence of incoherently received power on the integration time. Part 1. The value pairs in parentheses denote the conditions for each single measurement point, e.g. (3,2) means source with relative power ‘3’ in fan beam 1 and source with relative power ‘2’ in fan beam 2. All relative values are relative to the source power from inside the ‘coherent’ area *c*.

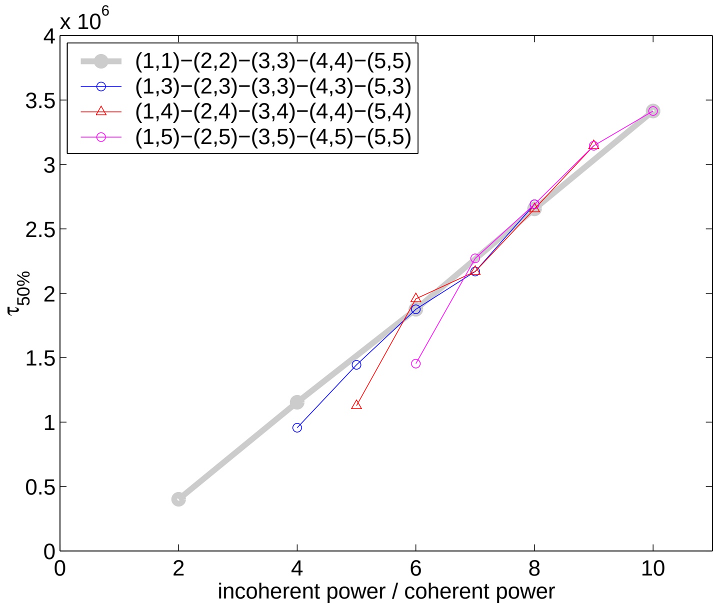


Figure 5.7: Influence of incoherently received power on the integration time. Part 2. Explanations see figure 5.6.

As can be seen clearly in the graphical representation of these results, the worst case happens when there is a strong source present in each of the two incoherent areas of the two fan beams. Also, as the thick grey line (which represents the same data series in either figure) clearly indicates, there is a linear relationship between the amount of incoherent power received from the two fan beams and the time $\tau_{50\%}$ it takes for 50% of the measurements to reach a value within the $1 \pm \varepsilon$ boundary.

We conclude, that for accurate estimates of how long an integration time is required, we need to simulate strong incoherent sources in both arms (clearly, having only one source results in lower values for $\tau_{50\%}$, as can be seen in figure 5.6), as this represents the worst-case situation.

From the measurement in figures 5.6 and 5.7 we can easily derive the linear relationship between $p = \frac{total\ incoherent\ power}{coherent power}$ and $\tau_{50\%}$ as

$$\tau_{50\%}(p) = a_1 \times p + b_1 \tag{5.8}$$

where, for the given measurements, we find $a_1 = 3.7678 \times 10^5$ and $b_1 = -3.52937 \times 10^5$.

5.2.5 The Effects of Varying ε

Because, in reality, we might be interested in getting results that are much more, or possibly less, accurate than $\pm 2\%$, a series of simulations was run for different values of ε. As before, each simulation was run 500 times, and the value $\tau_{50\%}$ indicates after how long an integration time 50% of all simulation runs reached the $1 \pm \varepsilon$ boundary. The simulations that were run are listed in table 5.4, along with the results. The results can be viewed in graphical form in figure 5.8.

We can approximate the curves for small ε with straight lines. This was done for $\tau_{50\%}$ in figure 5.8. These lines show the exponential relationship between ε and τ (note that the x-axis has a logarithmic scale). Since we are interested in the required integration time for good results (i.e. small ε), the non-exponential behaviour for large ε can be ignored.

The straight line approximation can be described as

$$\tau(\varepsilon) = a_2 \times \log_{10}\varepsilon + b_2 \tag{5.9}$$

where, for the given measurements, we find $a_2 = -1.6958 \times 10^7$ and $b_2 = 8.1550 \times 10^6$.

Experiment ID	amplitude of source in beam 1	amplitude of source in beam 2	amplitude of source in coherent area	$\tau_{50\%}$	$\tau_{80\%}$	k
job01	1	1	1	400623	822659	0
job02	1.4142	1	1	635422	1392970	2
job03	1.7321	1	1	956465	1885439	5
job04	2	1	1	1127850	2192383	17
job05	2.2361	1	1	1453947	2609013	34
job12	1.4142	1.4142	1	1154188	2210206	11
job13	1.7321	1.4142	1	1444399	2813817	31
job14	2	1.4142	1	1957109	3695207	60
job15	2.2361	1.4142	1	2271122	3892246	81
job23	1.7321	1.7321	1	1875462	3458175	55
job24	2	1.7321	1	2168734	3827143	73
job25	2.2361	1.7321	1	2690699	3999378	99
job34	2	2	1	2654735	3998268	97
job35	2.2361	2	1	3144514	Inf	125
job45	2.2361	2.2361	1	3414824	Inf	151
jobz1	1	0	1	120857	272371	0
jobz2	1.4142	0	1	268369	531629	0
jobz3	1.7321	0	1	449572	890393	0
jobz4	2	0	1	545126	1092463	1
jobz5	2.2361	0	1	658451	1315792	0
jobz6	2.4495	0	1	795421	1628161	5
jobz7	2.6458	0	1	877658	1925866	5

Table 5.3: Simulations with sources of different intensities. 4,000,000 Samples. 500 runs. k denotes the number of runs that did not reach the $1 \pm \varepsilon$ boundary. $\varepsilon = 2\%$. Source bandwidth 15MHz.

Experiment ID	ε	$\tau_{30\%}$	$\tau_{50\%}$	$\tau_{80\%}$	k
eps01	8.0%	121070	193909	421427	0
eps02	4.0%	522880	843783	1651782	0
eps03	3.0%	962579	1557933	3161771	1
eps04	2.0%	2362917	3386110	6777717	20
eps05	1.8%	2589395	3825576	7375680	28
eps06	1.5%	3468376	5177072	9262067	60
eps07	1.2%	4927062	7033661	inf	112
eps08	1.0%	6372734	8404444	inf	157
eps09	0.8%	7726645	9798213	inf	215
eps10	0.6%	9259003	inf	inf	266
eps13	0.3%	inf	inf	inf	379
eps14	0.2%	inf	inf	inf	421

Table 5.4: Simulations with different boundary conditions ε. 10,000,000 Samples. 500 runs. Source bandwidth 15MHz. The experiment simulated two equally strong sources (one in each fan beam), each emitting 2 times the power of the coherent source. k denotes the number of runs that did not reach the $1 \pm \varepsilon$ boundary.

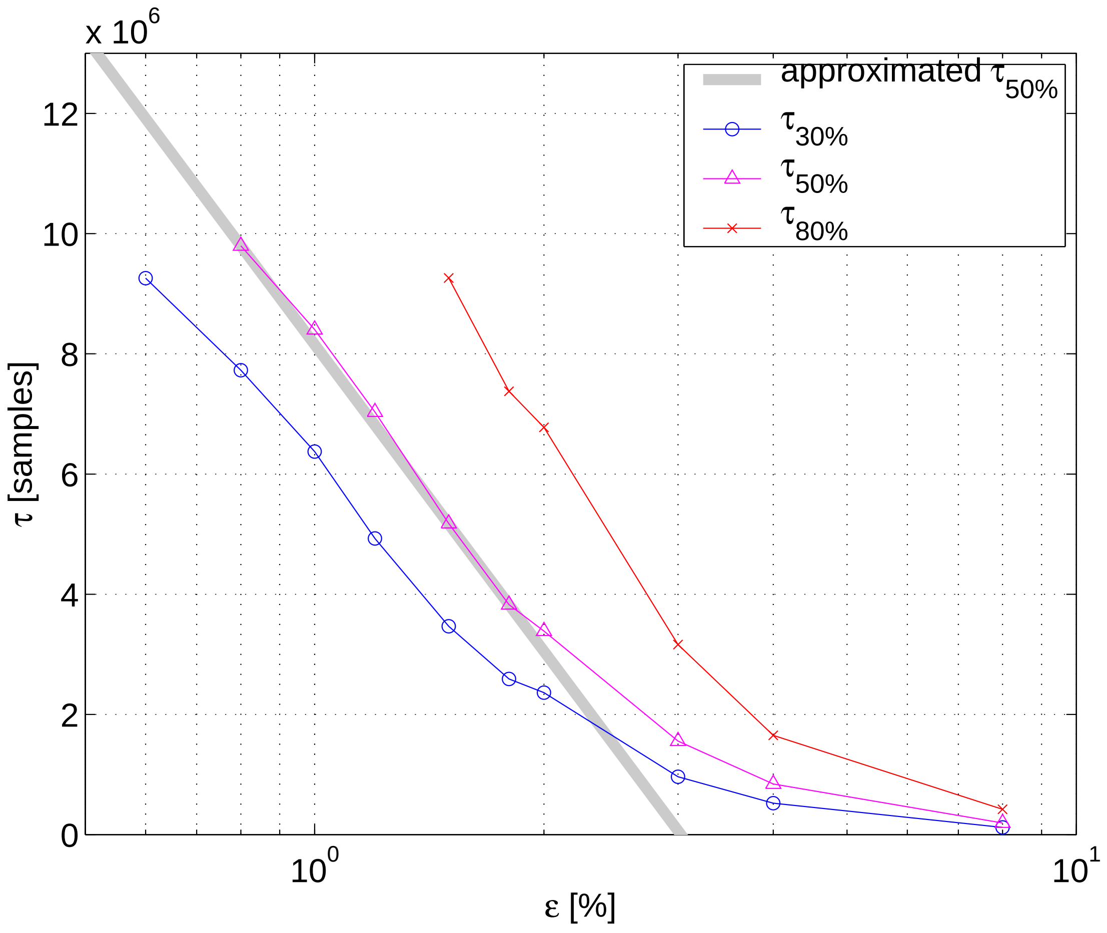


Figure 5.8: Effect of different boundary conditions ε on integration time

5.2.6 Different $\tau_{n\%}$

The previous sections mainly used $\tau_{50\%}$ as an indicator as to how quickly an experiment reached the set boundary (see section 5.2.4 for an explanation of $\tau_{50\%}$). In this section we will show how τ, i.e. the required integration time, varies, if we want a certain percentage of experiments to reach the set boundary. We can easily get this information from one of the previous simulation runs, in this case we selected the results from the runs ‘job01’, ‘job03’ and ‘job23,’ see table 5.3 already discussed in section 5.2.4.

Figure 5.9 shows the result in a graphical format. The x-axis is the percentage of experiments that are to reach the $1 \pm \varepsilon$ boundary, and the y-axis shows the corresponding value of τ that is required to achieve this goal.

From the blue line (representing ‘job01’) in this figure we can infer that it takes about 8 times ($=\frac{2954305}{400623}$) longer for nearly 100% of all runs to reach the set boundary, than it takes for 50% of the runs. Therefore it is reasonable to measure $\tau_{50\%}$ or even $\tau_{30\%}$, we are nevertheless able to predict how long it will take to get a much larger amount of runs to reach the set boundary.

5.2.7 System Bandwidth

For the sake of simulation speed, all simulations in the previous sections were run with noise source ID 07 (see table 5.1). In this section we will investigate how the bandwidth of the noise sources (and therefore the bandwidth of the whole system, as explained in section 5.1) affects the required integration time. For this, the five experiments in table 5.5 were run.

Figure 5.10 shows the results in graphical form. As expected, half the bandwidth requires roughly twice the integration time and vice versa.

5.2.8 Varying the Sampling Rate

With our model running at 16 times the required sampling rate (Nyquist-rate, see section 5.1) the simulated signals resemble the analogue signal very closely.

However, for a real system, we are looking for a more efficient way of sampling, as the amount of data grows proportionally with the sampling rate.

The simulations described in this section are run to find out how a reduction in sampling rate affects the required integration time, respectively the amount of data required to get the same result as with a higher sampling rate.

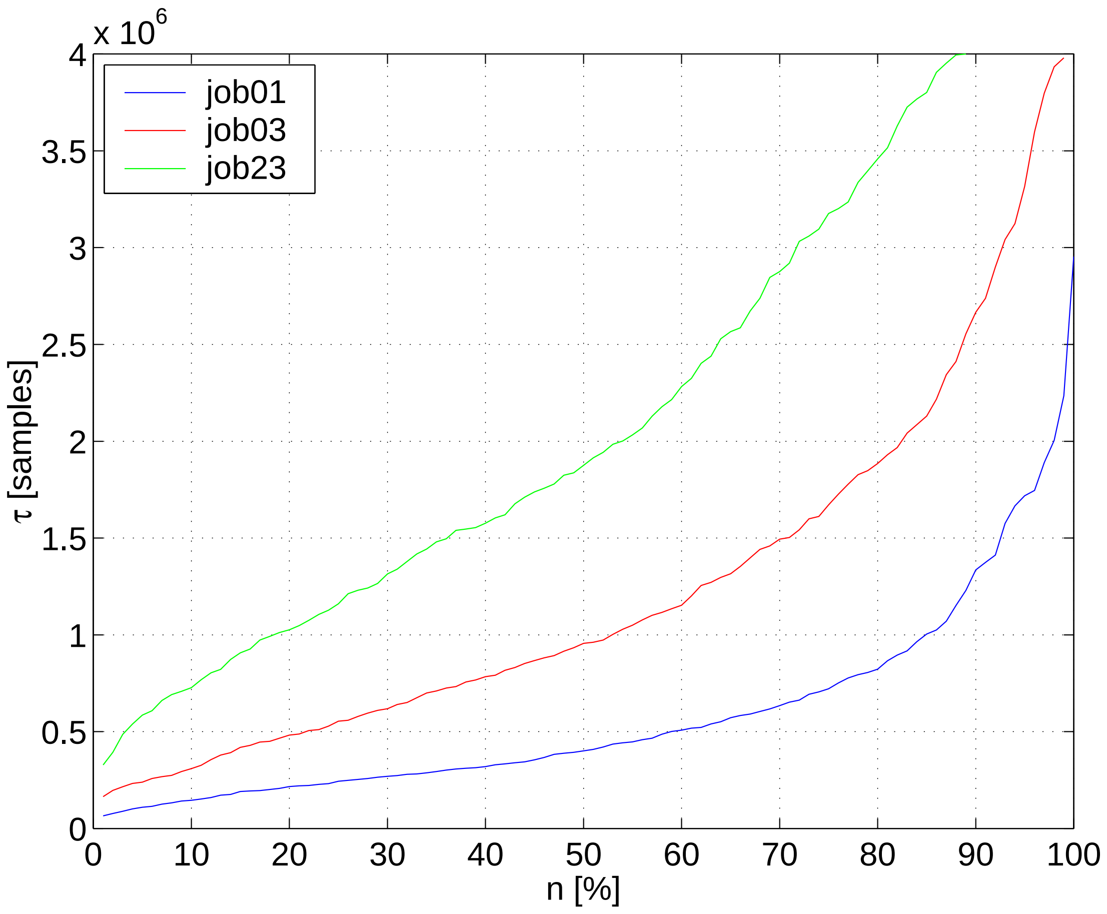


Figure 5.9: Required integration time to measure $\tau_{30\%}$, $\tau_{50\%}$,...,$\tau_{n\%}$

Experiment ID	used filter parameter file	bandwidth	$\tau_{30\%}$	$\tau_{50\%}$	$\tau_{80\%}$	k
ch1	`cheb_01`	1.0 MHz	3736746	5866927	9875740	82
ch2	`cheb_02`	0.6 MHz	5800412	7935051	Inf	145
ch3	`cheb_03`	2.0 MHz	1943446	3145463	5812332	23
ch4	`cheb_04`	4.0 MHz	1152412	1763927	3620320	2
ch5	`cheb_05`	15 MHz	287363	426201	902578	0

Table 5.5: Simulations with different bandwidths. 10,000,000 samples. 500 runs. k denotes the number of runs that did not reach the $1 \pm \varepsilon$ boundary. $\varepsilon = 2\%$. All three noise sources issue the same power.

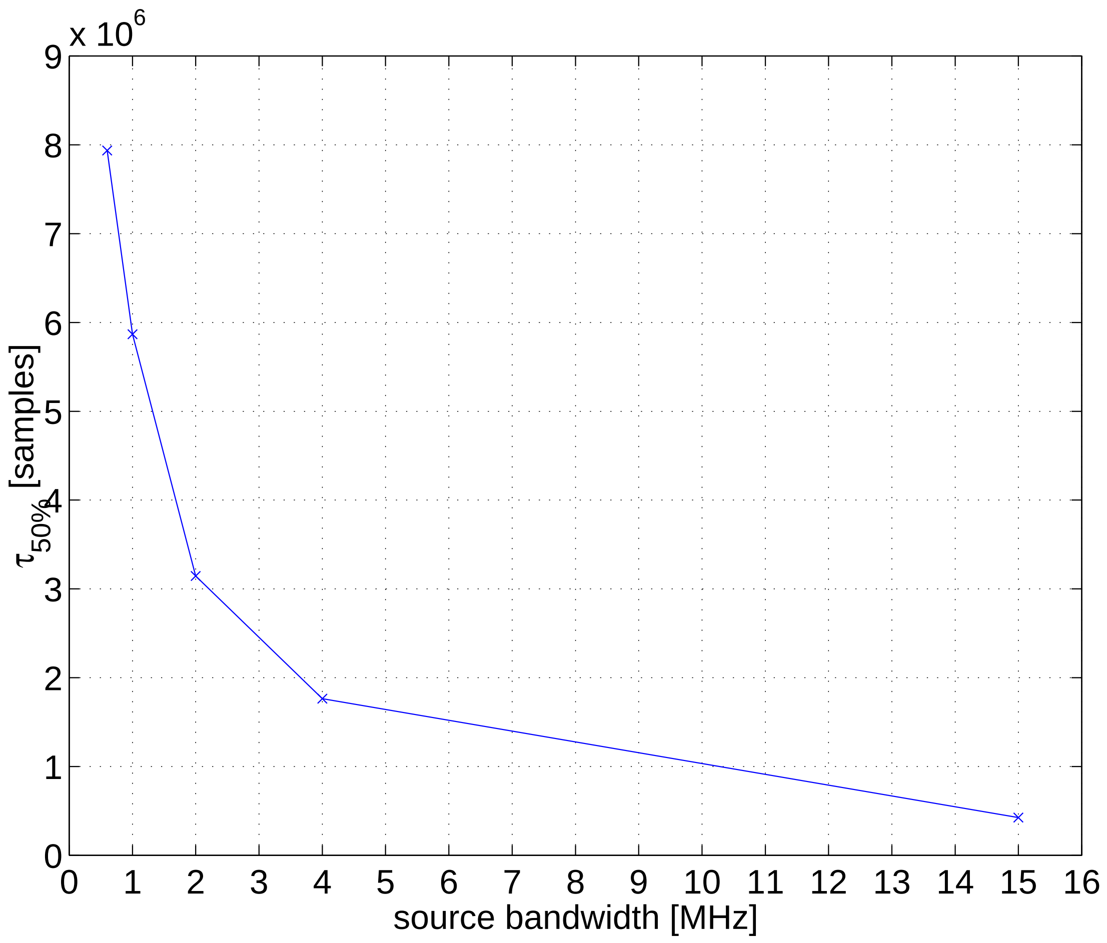


Figure 5.10: Required integration time in samples versus system bandwidth

We run these simulations by simply omitting some of the samples issued by the simulated random noise sources (described in section 5.1). This is a simple form of decimation, see for example [Nie91]. As long as we stick to this simple but perfectly adequate approach of omitting some samples, not much change to the existing program is required. Table 5.6 shows the experiments that were run, together with the results. Figure 5.11 shows the results in graphical form.

Not surprisingly, as the number of samples per period reduces, so does the number of total samples required to achieve the same result. In fact, the ratio of required samples to sampling rate remains roughly constant as long as the sampling rate is above the Nyquist rate. This can clearly be seen in figure 5.11.

From this figure, we can also see that results start to become worse as soon as we get below the Nyquist rate. It seems, however, that we still get results, although at the cost of a longer integration time. This behaviour is known as aliasing and causes loss of information. Real systems will not work as intended if the sampling rate goes below the Nyquist rate. For our particular concern in this chapter, however, this is not of interest.

5.2.9 Conclusion

In this section we will use the results derived in the previous sections to calculate a minimum required integration time for the ARIES system, according to the model described in this chapter.

We can get a first estimate for the required amount of samples from the relationships discussed in section 5.2.4. (The number of noise sources as discussed in section 5.2.3 has no influence on integration time.)

In section 5.2.4 we derived a formula (5.8) that gives us the required integration time to reach the $1 \pm \varepsilon$ boundary, given the ratio of total incoherent power to coherent power. We can estimate this ratio from existing data of the IRIS riometer.

The maximum power ever recorded in a single IRIS beam (apart from scintillation effects that overload the IRIS receivers) was $P_{max,IRIS} = -105\text{dBm}$. This amount of power is recorded, when Cassiopeia A (see chapter 3) is located within the respective beam.

As shown in section 5.2.4, the worst case in terms of required integration time happens, when there is a big source in both of the fan beams to be cross-correlated. Very pessimistically, we therefore assume an equally strong source of about the power of Cassiopeia in both fan beams, giving us a total incoherent power of about $P_{max} = 2 \times P_{max,IRIS} = -102\text{dBm}$.

Experiment ID	f_s	$\tau_{30\%}$	$\tau_{50\%}$	$\tau_{80\%}$	k
dec01	32.00	2654735	3998268	N/A	97
dec02	16.00	1131628	1696224	3743129	1
dec03	10.66	681846	1011096	2088641	0
dec04	8.00	501269	820403	1765324	0
dec05	6.4	428822	654436	1448093	0
dec10	3.20	207887	302672	639440	0
dec16	2.00	261884	404004	775031	42
dec24	1.33	88279	131415	283566	1
dec32	1.00	132568	200295	389125	44
dec48	0.66	83080	128975	283910	16
dec64	0.50	60138	97492	203648	13

Table 5.6: Simulations with different sampling rates. f_s shows the effective sampling rate in multiples of the operating frequency of the system (38.2MHz). 500 runs. dec00: 4e6 samples, dec01-dec05: 10e6 Samples. dec16-dec24: 1e6 samples. dec32-dec64: 5e5 samples. Only 260 runs were simulated for experiment dec05.

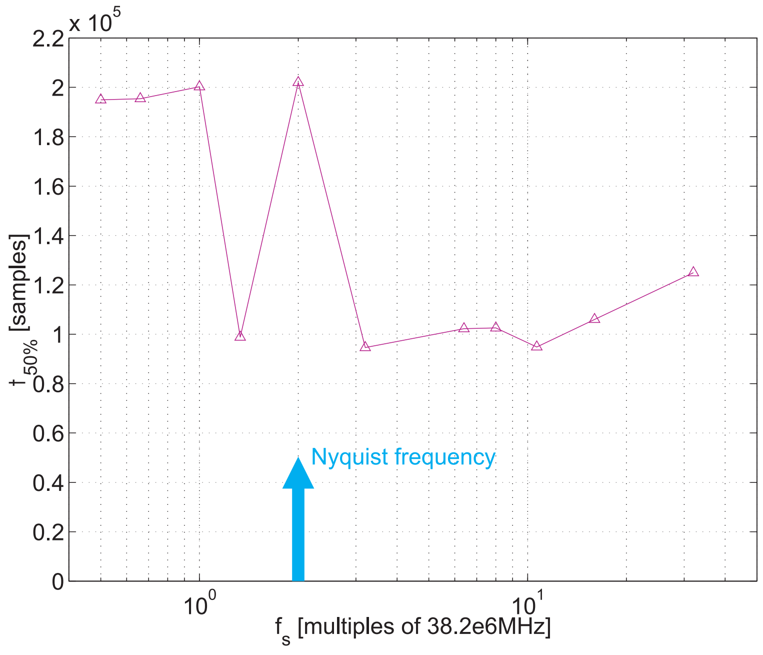


Figure 5.11: Required integration time in samples versus sampling rate

The lowest power ever recorded from a single IRIS beam was during the high absorption period around the 15th of July 2000. The received power was around $P_{min,IRIS} = -129$dBm. This is the noise received from the cosmic radio background, attenuated by very strong absorption in the Earth's ionosphere.

Now, the effective covered area of one of IRIS's beams is about 16 times as large as the effective area of an ARIES beam. This means, that the signal received from the coherent area of the two fan beams that represents one pencil beam from ARIES is likely to be $\frac{1}{16}$th of the minimum power recorded in an IRIS beam, so we have $P_{min} = \frac{1}{16} \times P_{min,IRIS} = -141$dBm.

Altogether, we end up with a worst-case ratio of total incoherent power to coherent power as follows:

$$p = \frac{P_{max}}{P_{min}} = \frac{-102\text{dBm}}{-141\text{dBm}} = \frac{10^{-102/10}}{10^{-141/10}} = 3.98 \times 10^3. \tag{5.10}$$

The second contributing factor (see section 5.2.5) is the required precision of the result, represented by ε. For now, we specify a boundary condition ε of 2%, which will give us results with a precision of better than ± 0.1dBm. As equation 5.8 is already based on $\varepsilon = 2\%$, we do not need to take equation 5.9 into account. However, this equation is still useful for evaluating the required integration time for different values of ε.

In addition to the above factors, we do not want only 50% of all runs to arrive at the result, but close to 100%. According to section 5.2.6 we therefore have to multiply the calculated integration time by 8.

Finally, as τ is given in number of samples throughout the preceding sections, we need to divide τ by the sampling rate in order to get the required integration time T in seconds. Note that, according to section 5.2.8, the sampling rate does not affect the result as long as we are sampling above the Nyquist rate.

Equation 5.11 below collates all this information into one equation.

$$T = (32 \times 38.2 \times 10^6 \frac{1}{\text{s}})^{-1} \times 8 \times (a_1 \times p + b_1) \tag{5.11}$$

And for the given values for a_1, b_1 and p as determined in the previous sections we find

$$T_{ARIES,B=15MHz} = 9.8\text{s} \tag{5.12}$$

This is the worst-case required integration time for a system with a (quite unrealistic) bandwidth of about 15MHz. The influence of different bandwidths is investigated in section 5.2.7 above. We find, that for a more realistic system bandwidth of 0.6MHz, we require about $\frac{7935051}{426201} = 19$ (see table 5.5) times the integration time calculated above, thus

$$T_{ARIES,B=0.6MHz} = 19 \times T_{ARIES,B=15MHz} = 186\text{s}. \tag{5.13}$$

5.3 Nielsen's Estimates

In [Nie02b, NHG04] Nielsen derives the fluctuations of absorption measurements in dB for the Mills Cross cross-correlation riometer as

$$dA = 10\log(1 + \frac{(N-1)^2}{\sqrt{2 \cdot \tau \cdot B}}) \tag{5.14}$$

as opposed to

$$dA = 10\log(1 + \frac{1}{\sqrt{\tau \cdot B}}) \tag{5.15}$$

for an antenna with arbitrary radiation pattern, but without the cross-correlation stage.

N in equation 5.14 stands for the fraction of total area of the two beams to be cross-correlated compared to the overlapping area, τ is the integration time used, and B is the bandwidth of the system.

Nielsen aims for a $dA < 0.1$dB and finds that for a bandwidth of 250kHz the required integration time is

$$\tau > 3.2 \cdot 10^{-3} \cdot (N-1)^4. \tag{5.16}$$

He concludes that for an integration time in the range of 10 to 20 seconds, a fan lobe should not be subdivided into more than about 10 pencil antenna lobes. This means that the power received in a pencil lobe should exceed 10% of the total power received in the fan lobe.

For an array with N=16 as outlined in the original design [Nie01], Nielsen finds an integration time of $\tau = 300$s. N=16 can be achieved by adding the outputs of each Butler Matrix pairwise, or by linear/cosine tapering of the inputs, where cosine tapering gives preferable sidelobe performance [Mue72], see also section 2.3.5.

In [Nie02b, chapter II], Nielsen goes on to describe a way of modifying the existing ARIES antenna array to achieve smaller values for N. This can be achieved by adding a second set of 32+32 antennas, connecting each existing antenna to a new antenna half a wavelength apart. This would produce narrower fan beams, thus reducing N. However, the field of view of the system would also be reduced, from 300×300km to about 150×150km.

Finally, for the 2002 test, Nielsen suggests using the tapered system (i.e. N=16) and integrating for 5min. This should lead to observations that fluctuate about 0.1dB about the mean. Refer to chapter 9 for the actual results obtained during this experiment.

5.4 Hagfors's Estimates

In [Hag01b], Hagfors tackles some general issues that have been neglected in Nielsen's report [Nie01]. These include the effects of sidelobes, polarisation problems and the effect of snow on the ground. None of these issues seem to be of particular concern for the 2002 experiment.

[HGH03] (based on earlier notes in [Hag01a]) goes into the details of the difference between cross-correlation and filled aperture riometers. Without going into details of the statistical considerations used in this description, we will only quote his final result. Hagfors states that "If one inquires as to the amount of integration time one must have to make up this handicap [of the Mills Cross] compared to the filled array, the integration time ratio must be larger by a factor of 100 to 900."

If we use IRIS ($B = 250$kHz, $\tau = 0.045$s) as an example of a filled aperture riometer, this suggests an integration time for ARIES of about 5s to 45s.

5.5 Summary

Table 5.7 summarises the estimated integration times from the previous sections. Note that the three very different approaches to determining reasonable integration times (Grill/Yamagishi, Nielsen, Hagfors) lead to similar results, and later chapters will show that the Mills Cross system can indeed achieve these integration times.

System specification	Nielsen	Hagfors	Grill
$B = 600$kHz, untapered			$\tau = 186$s
$B = 250$kHz, cosine-tapered	$\tau = 300$s		
$B = 250$kHz		$\tau = 5...45$s	

Table 5.7: Summary of integration time estimates

Chapter 6

Radiation Pattern Simulations: RIOSIM

Having looked at the basic working principles of antennas and riometers in chapters 2 and 3, this and the following chapter operate on a slightly higher level of abstraction, focusing on radiation patterns and how they can help in the evaluation and deployment of real system designs. As discussed in chapter 2, the receiving properties of each antenna or system of antennas are fully described by its associated radiation pattern. Depending on the point of view, this pattern is also referred to as antenna directivity or sensitivity pattern. It describes how the antenna system in question reacts to an incoming signal from any possible direction. In (imaging) riometry, we want to have a clear peak sensitivity in one direction and as low a sensitivity as possible in all other directions, in other words we want to form pencil-shaped beams with low sidelobes.

Now ideal pencil beams are unfortunately a purely theoretical thing, in fact many of the chapters in this thesis come back to this issue. The aim of this chapter is therefore to simulate the radiation pattern that various configurations of the Mills Cross can be expected to produce. Nielsen did some radiation pattern simulations in [Nie01], and we will refer to this in the appropriate places. The main purpose is not to imitate work that has already been done, but to put it into a greater, more versatile, context (framework), using the radiation patterns to derive results that can be expected when operating the system as specified, and using these simulated results for validating data received by real systems. The toolbox developed in this chapter will enable us to apply all findings to arbitrary riometers or, in fact, antenna systems.

It is worth mentioning that there are different ways of actually deriving the radiation pattern for (Mills Cross) antennas. While Nielsen's results are based on theory, it is also possible to simulate the Mills Cross, or any other antenna, using finite element method (FEM) software. Initial steps toward this have already been taken by the author in collaboration with G. Dekoulis, and although these are not discussed further in this thesis, FEM-simulated radiation patterns are readily supported by RIOSIM and one example can be found in section 6.3.10 (discussing the **RNECPat** class). FEM simulations give further insight into the real world behaviour of antennas, as they can take into account real-life effects such as imperfect ground planes that cause the real radiation pattern to deviate from its predicted theoretical shape.

6.1 Design Goals

Having verified the basic fitness of the Mills Cross system for our purposes in chapter 5, the aim of this chapter is to use radiation patterns to simulate the actual results that we can expect from the system. This includes simulations of the received signal during 'quiet days'[1] and simulations concerning the influence of strong celestial radio sources, the latter enabling us to predict, amongst other things, scintillation effects. All the results from this and the following chapter have directly influenced the schedule for the various ARIES on-site experiments, first and foremost the one whose results will be described in chapter 9.

Through the abstraction of radiation patterns, all results that are achieved in this chapter can easily be applied to any riometer system, as long as its radiation pattern is known. Due to a completely object-oriented approach, the core software does not have to be modified in any way to be able to adapt to new radiation patterns. This means that we can, for example, predict scintillation in every existing riometer with the same piece of software.

To summarise, the aims of the toolkit implemented in this chapter are to:

- Integrate different sources of radiation patterns (simulated, calculated, measured) into one program/framework.
- Integrate digitised sky background noise maps.
- Enable creation of theoretical quiet-day curves based on the different available radiation patterns and sky map(s).

[1]Similar simulations have been done by Huiyu Tao [Tao04] for the IRIS system, and some of the basics of this chapter are based on Tao's work. The tools developed in this chapter will, however, be much more flexible, as the following sections will show.

- Take into account tilted base planes and other imperfections of the instrument in question.
- Predict when certain radio stars will pass through which beam(s) and, based on this,
- predict scintillation effects.
- Enable the development of experiment schedules, taking into account the results of the above simulations.
- Develop all these algorithms in a general way so that they can be applied to other existing riometers.

The remainder of this chapter will describe the RIOSIM framework that was implemented for performing the tasks above. This framework will be used throughout the rest of this thesis. Chapter 7 in particular is dedicated to presenting some major applications.

6.2 Implemented Object Structure

With the background knowledge on coordinate systems (appendix B.1) a software structure can be developed to represent the given problem in a flexible enough way to be able to solve all the tasks at hand. This software consists of different layers, or building blocks, see figure 6.1. The bottom layer provides all the necessary objects of the application (problem) domain. These objects can then be used in higher-level layers to generate QDCs, predict scintillation etc.

There is also an independent collection of useful helper functions that are not part of any objects and a small set of functions that are only used by RIOSIM internally and should not be used by other applications. Finally, we have the application layer. Programs in this layer are the ones that make use of the RIOSIM classes and functions.

From the general description of how antennas and riometers (or radiotelescopes in general) work (see chapters 2 and 3), we find that we have to deal with three basic groups (types, classes) of objects. A *radiation pattern* represents the sensitivity of the given aerial system under investigation. The convolution of a radiation pattern with the *cosmic sky background* gives the received signal strength. For the prediction of scintillation and for instrument alignment and calibration, *radio stars* play an important role.

It is therefore sensible to implement a class hierarchy that integrates these principal objects. The developed class hierarchy will be described in the following sections.

6.3 Radiation Patterns: RRadPat and Descendants

Radiation patterns can be classified into a wide variety of different types. Although they all describe antenna sensitivity, the way in which the actual sensitivity is calculated varies with the type of antenna that is being described. Some antenna patterns have mathematical descriptions. Others can be generated by simulation. Antenna patterns can be derived from the ground up (for example the pattern of a crossed dipole derived from the mathematical description of the radiation patterns of two individual dipoles), or by the use of some simplified description (as used in, for example, **RXDipNielsenPat** — see section 6.3.5).

All types of radiation patterns commonly used in riometry have been integrated into the object tree in figure 6.2. All radiation pattern objects are derived, directly or indirectly, from the same base class **RRadPat**[2]. This results in complete interchangeability ('polymorphism'), a client that requests a **RRadPat** will also accept any of its descendants, for example the radiation pattern of an isotropic radiator (**RIsoPat**) or the simulated radiation pattern generated by the NEC finite-element simulation software (**RNECPat**).

Through **RRadPat**, all radiation patterns inherit a set of common functionality. They can return the gain in any given direction in various formats through their method **getGain()**. They can return plain text information about themselves. All radiation patterns also come with extensive plotting capabilities (implemented as common functionality in the base class **RRadPat**) that enable the user to analyse and verify results simply by looking at a range of graphical representations of the radiation pattern (section 6.3.2).

The following subsections will discuss the available common functionality of all radiation pattern (**RRadPat**) objects. The final sections from section 6.3.4 onwards will deal with specifics for certain concrete types of radiation patterns as represented by the bottom layer of objects in figure 6.2. As a rule, these specifics need only be known when new radiation patterns are being created. Once the radiation pattern has been created, it will behave in exactly the same way as all the other types of radiation patterns. (Although some will be considerably slower than others due to internal processing — one reason for not always using the most accurate or detailed representation.)

All coordinates that are used within **RRadPat** and derived objects are per definition in the horizontal spherical coordinate system (section B.1.5) described in the underlying 'mathemati-

[2]We will ignore the base class **motherofallobjects** in this description. This class is a pure implementation detail, providing common get/set and help functionality for all derived objects.

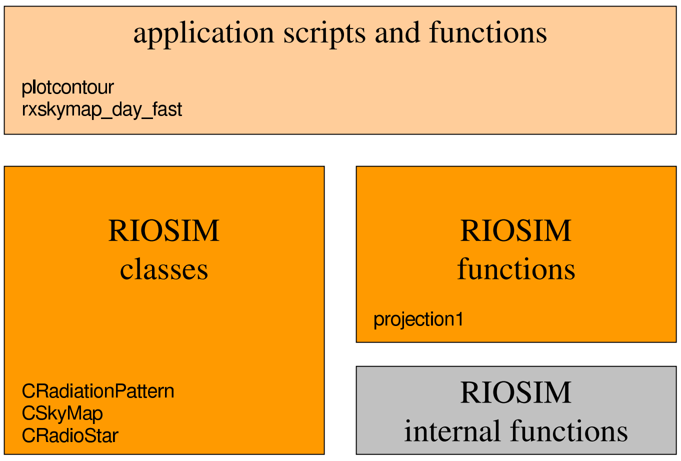


Figure 6.1: RIOSIM architecture

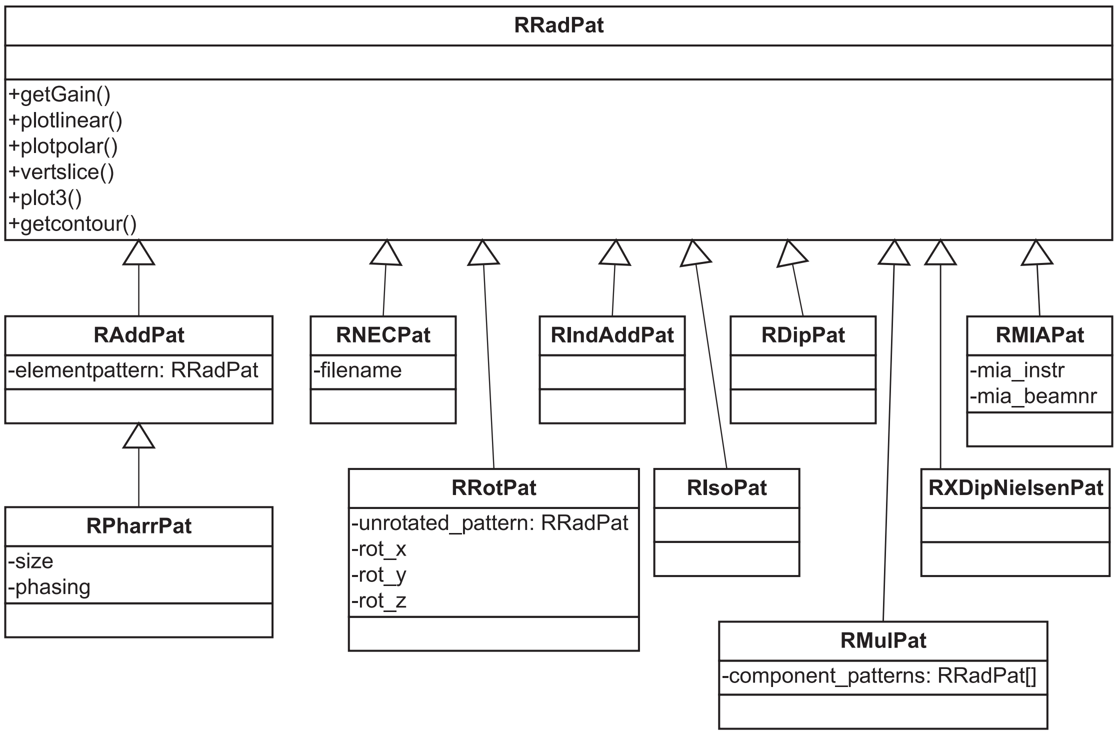


Figure 6.2: **RRadPat** and descendants

cal' spherical coordinate system (section B.1.2). All angles are expressed in radians. Azimuth angles therefore run from 0 (North) through $\frac{\pi}{2}$ (West), π (South), $\frac{3\pi}{2}$ (East) to 2π, elevation angles from $-\frac{\pi}{2}$ (directly underneath) through 0 (at the horizon) to $+\frac{\pi}{2}$ (at zenith). The 'mathematical' spherical coordinate system as described in section B.1.2 is is the only coordinate system that **RRadPat** knows (and needs to know) about, and it is being used consistently throughout the RIOSIM toolkit.

6.3.1 Gain Retrieval

The primary objective of a radiation pattern is to represent antenna gain (directivity, sensitivity) — see chapter 2. To retrieve this gain information, **RRadPat** provides the **getGain()** method.

getGain() maintains compatibility with simple power-based real-valued descriptions (as used in previous versions of the RIOSIM toolkit as well as in MIA [Marc]), while also enabling the use of complex gain values (describing phase offsets) and antenna polarisation.

getGain() takes matrices of azimuth and elevation angles and, depending on the requested output format, returns the gain in linear power units, relative to an isotropic radiator, in all directions defined by these angles:

a = GETGAIN(***pattern*****,** ***az*****,** ***el*** **), [a, AZ, EL] = GETGAIN(** ***pattern*** **)**

Returns the power gain in linear power units relative to that of an isotropic radiator for all directions specified by *az*, *el*. If no directions are specified, **getGain()** returns the gain in all directions as defined by a default grid with suitable resolution for the radiation pattern in question, the grid itself is returned in AZ and EL.

[Ex, Ey] = GETGAIN(***pattern*****,** ***az*****,** ***el*** **), [AZ, EL, Ex, Ey] = GETGAIN(** ***pattern*** **)**

Returns the electric field strength along the x and y polarisation planes for all directions specified by *az*, *el*. Positive x-axis of the polarisation vector points towards zenith. If no directions are specified, **getGain()** returns the gain in all directions as defined by a default grid with suitable resolution for the radiation pattern in question, the grid itself is returned in AZ and EL.

6.3.2 Plotting

As mentioned above, every **RRadPat** object can plot the radiation pattern it represents in a number of formats. This is useful to quickly visualise the basic properties of the given radiation pattern. Sidelobes, beamwidth, etc. can be compared qualitatively to other radiation patterns.

All plotting functions take certain common parameters in the form of name-value parameter pairs. If no special parameters are given, a plot with default settings is produced. In most cases, this plot will be a good starting point for further customisation.

The following parameters are available for all plotting commands:

stepsize_az, stepsize_el Specify the granularity of the underlying spherical grid in azimuth and elevation directions.

color An equation describing how colour information will be calculated in this plot. Defaults to (Ex.*conj(Ex)+Ey.*conj(Ey)), giving the total power in both polarisation planes. (Note the American spelling for consistency with existing MATLAB commands and toolboxes.)

color_scaling The scaling mode for the colour axis. At the moment, 'linear' and 'db' are supported. The external 'scale' function is used to perform the scaling. Defaults to 'db'.

color_scalingreference Specifies a reference power value used for relative scaling. Defaults to NaN (=no relative scaling for 'linear', automatic scaling relative to maximum for 'db')

color_min, color_max Colour values get capped at these minimum and maximum values. Default to -20 and 0 for 'db'.

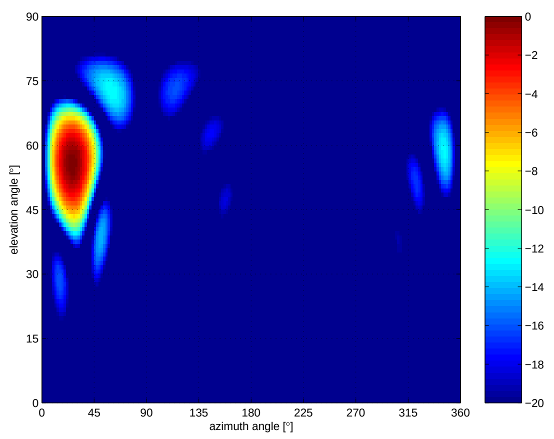
only_upper_hemisphere Whether to plot the whole sphere, or just the upper hemisphere of the radiation pattern. In riometry, antennas are usually located on a ground plane and only receive signals from overhead, so it is usually sufficient to only deal with the upper hemisphere of any given radiation pattern.

linestyle This parameter is passed on to the underlying low-level plotting functions to specify style parameters for line segments.

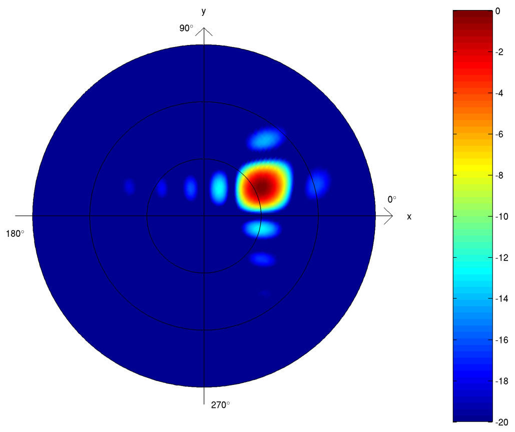
6.3.2.1 Basic Plots

The following plotting commands produce two-dimensional graphical representations of the radiation pattern. See the examples in figure 6.3, panels (a)–(c). All example plots show the same radiation pattern (that of the IRIS riometer's beam number 10).

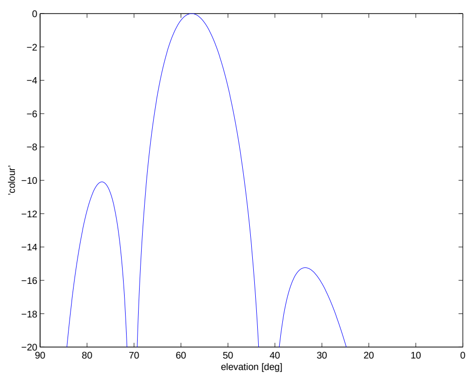
plotlinear() This is the simplest and quickest plot, aimed at quickly evaluating the basic shape of a radiation pattern. The x-axis shows azimuth, the y-axis shows elevation. By de-



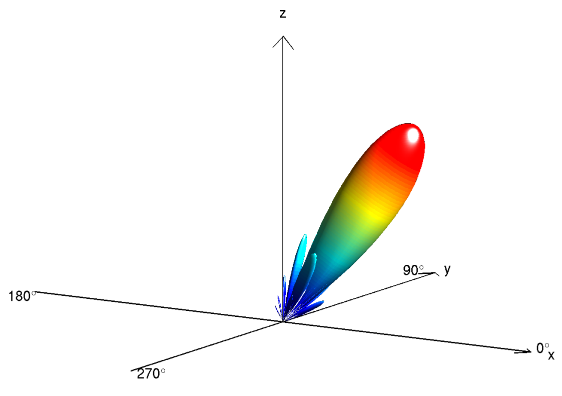
(a)



(b)



(c)



(d)

Figure 6.3: Basic plotting capabilities of a **RRadPat** radiation pattern object. Radiation pattern shown is IRIS's beam 10. (a) linear, (b) polar, (c) vertslice for $AZ = 42°$, (d) 3D. Colour scales in dB below maximum.

fault, colour represents the power gain in the given direction, though, as for all plotting functions, this can be customised with the **color_*xxx*** options.

plotpolar() This plot shows the same information as the linear type, but in a more commonly used 'polar diagram' format. Elevation angle is proportional to distance from the centre of the plot. 30°, 60°and 90° ($\frac{\pi}{6}$, $\frac{\pi}{3}$ and $\frac{\pi}{2}$) are indicated by concentric black circles. Polar plots will always be generated for the upper hemisphere only ($0 \leq EL \leq \frac{\pi}{2}$).

vertslice() This plot type is a line plot representing a vertical slice through the radiation pattern at a specified azimuth angle.

6.3.2.2 Three-dimensional Plots

In addition to the colour-related parameters (see above), the 3D plotting function **plot3()** also supports a similar set of parameters for radius scaling. Radius and colour information are therefore two completely independent datasets. Traditionally, the radius in a 3D plot of a radiation pattern represents the logarithmic power gain in the given direction. By default, radius and colour are therefore calculated using exactly the same expression. However, the separation between colour and radius allows for plots that, for example, combine gain and phasing information, such as the example in figure 6.4, panel (a). It also enables the generation of spherical diagrams such as the one in figure 6.4, panel (b) that still maintain the appearance of the simpler polar plot, while giving a much clearer (less distorted) representation of the actual radiation pattern shape.

In addition to the parameters described in section 6.3.2.1 above, the following parameters are supported for **plot3()**:

radius An equation describing how radius information will be calculated in this plot. Defaults to (Ex.*conj(Ex)+Ey.*conj(Ey)), giving the total power in both polarisation planes. Figure 6.4, panel (b), uses (ones(size(Ex)), which results in a constant radius of 1 for all directions.

radius_scaling, radius_scalingreference, radius_min, radius_max Similar to the equivalent **color_*xxx*** options described above, but referring to radius scaling.

6.3.3 Contouring

Any **RRadPat** object can return its contours at a specified level below maximum gain through the **getContour()** method. It does, however, not know anything about how to plot these contours. This is a deliberate design decision, as plotting contours also requires knowledge about projection methods and maps, both concepts that are only very loosely related to the concept of a radiation pattern and according to the 'high cohesion' principle of object-oriented design should therefore not be included in the design of a radiation pattern class. In order to plot radiation pattern contours, all this additional knowledge is required, and it is the responsibility of the contour plotting functions described elsewhere (chapter 7, section 7.1) to harness this knowledge to actually plot radiation pattern contours.

6.3.4 The Radiation Pattern of a Simple Dipole: RLinDipPat

This class represents the radiation pattern of a (perfect) linear dipole according to equation 6.1 as adapted from [Kra88].

$$E_\theta = \frac{-jZ_0 I_0 e^{(-j\beta r)}}{2\pi r} \times \frac{\cos(\beta L \cos(\theta)/2 - \cos(\beta L/2)}{\sin\theta} \tag{6.1}$$

The length of the dipole can be specified in multiples of the operating wavelength λ using the **length** parameter (represented by L in equation 6.1). The dipole is always centred on the origin and aligned along the z-axis. To rotate it into another position, use this radiation pattern in combination with **RRotPat** (section 6.3.9).

6.3.5 The Simplified Radiation Pattern of a Crossed Dipole: RXDipNielsenPat

Instead of calculating the response of a crossed dipole above (perfect) ground from the responses of its individual components (a cross of dipoles driven by 90° phase shifted signals and a mirrored cross of dipoles to simulate reflective ground), a simplified approach is often more suitable. This is especially true as soon as the array factor dominates the considerations, which is clearly the case when looking at phased array antennas. **RXDipNielsenPat** describes the radiation pattern of a crossed dipole above perfect ground as used by Nielsen in [Nie01] (his equation 19):

$$G = 2\sin(2\pi h \times \sin\theta), \tag{6.2}$$

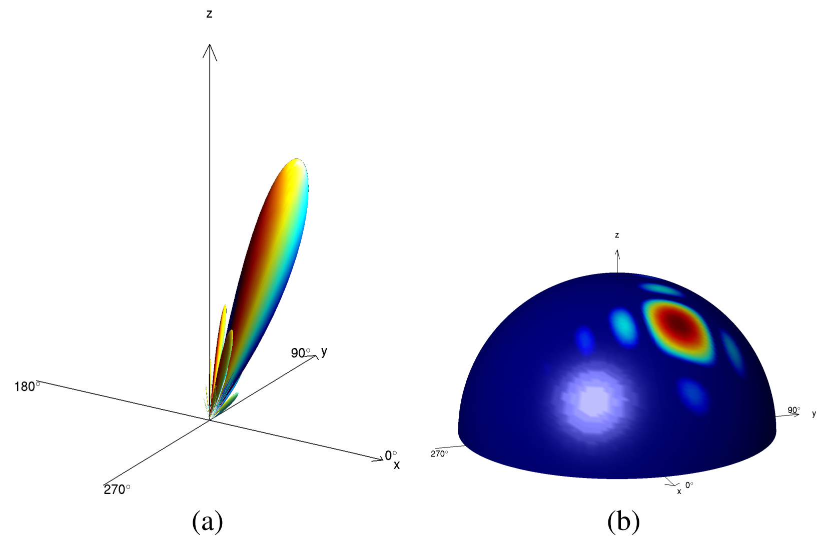


Figure 6.4: Advanced 3D visualisation options. (a) 3D phase plot — colour scale from $-\pi$ to $+\pi$, (b) 3D polar diagram — colour scale as in figure 6.3 (b). Both panels show IRIS beam 10 as in figure 6.3.

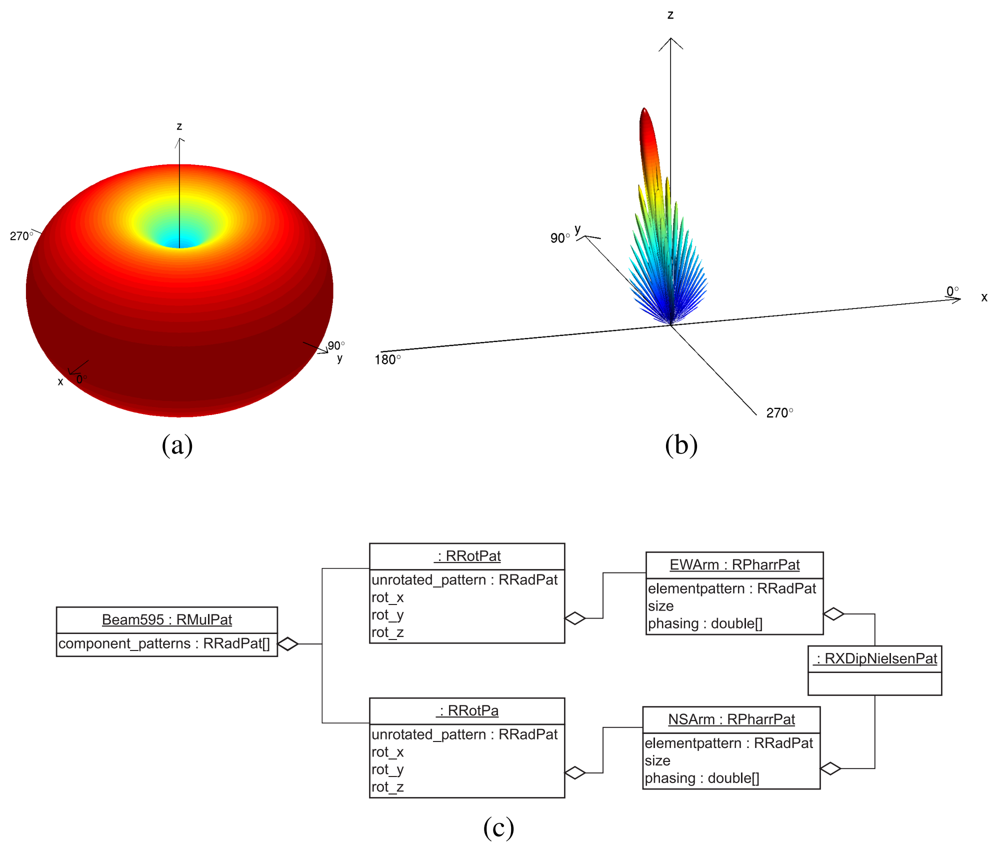


Figure 6.5: Some RRadPat-derived radiation patterns. (a) **RLinDipPat**, (b) **RMulPat** for two perpendicular tilted linear arrays, this is the radiation pattern that is used to describe ARIES pencil beam 595, (c) Object diagram for situation (b) showing the internal composition of this particular **RMulPat** object.

where *h* is the height of the half-lambda crossed dipole above ground and θ is the elevation angle.

Figure 6.6 shows an example of an FEM-simulated radiation pattern for a crossed dipole above a perfect ground plane (i.e. a mirrored crossed dipole) in comparison to the simplified mathematical representation of a crossed dipole as used by Nielsen [Nie01] as represented by **RXDipNielsenPat**. Whereas the pattern as used by Nielsen is perfectly symmetric to the z-axis, the actual shape as simulated by miniNEC exhibits a slight dependency on azimuth angle. Also, the miniNEC simulation (although not visualised in the figure) inherently gives signal strength for both polarisation planes (*Ex* and *Ey*), whereas the simple formula used for **RXDipNielsen-Pat** only returns overall signal strength, which **RXDipNielsenPat** returns as being entirely x-polarised for simplicity.

It should also once again be noted that the differences in element patterns quickly grow insignificant for phased arrays, as the array factor quickly starts to dominate the overall shape of the radiation pattern, especially around zenith. See chapter 2 for a more detailed discussion of phased arrays.

6.3.6 Linear Additive Arrays: RAddPat and RPharrPat

Additive linear phased arrays (see chapter 2) are used extensively in riometry. **RAddPat** (additive array pattern) and **RPharrPat** (phased array pattern) are designed to represent these types of antenna systems. **RAddPat** supports an arbitrary number of antenna elements at arbitrary locations and with arbitrary phasing. **RPharrPat** is a special case of **RAddPat** for rectangular antenna arrays with fixed spacing and phasing. The beam patterns plotted in figures 6.3 and 6.4 are, in fact, all examples of **RPharrPat** objects (in this instance representing an IRIS riometer beam).

Both **RAddPat** and **RPharrPat** use a single element pattern (again a **RRadPat**-derived object) representing the radiation pattern of one antenna element. This is sufficient, as antenna arrays usually consist of identical antenna elements.

It can be useful to additively combine two or more different radiation patterns, for example for deriving the radiation pattern of a crossed dipole (consisting of two linear dipoles at 90° of each other). This is implemented in the **RIndAddPat** (additive array pattern made up of individual elements), see section 6.3.7 below. Note that this will greatly increase the processing and memory requirements compared to an additive array pattern made up of identical elements, as each element pattern needs to be stored (and calculated) separately.

6.3.7 Additive Arrays of Individual Elements: RIndAddPat

This is the most generic form of an additive array radiation pattern. The array can be made up of an arbitrary number of individual radiation patterns (element patterns), with each one located at an arbitrary location and with an arbitrary phase delay. For any given direction of interest, the array sensitivity will be determined as the correctly phased sum of the contributions from all elements as described in chapter 2, section 2.3.3.

Compared to arrays consisting of identical elements (represented by **RAddPat** and **RPharrPat**, see section 6.3.6 above), **RIndAddPat** will have greatly increased processing and memory requirements, as each element pattern needs to be stored (and calculated) separately.

6.3.8 Multiplicative Arrays: RMulPat

Mills Cross type antenna arrays are multiplicative arrays (see chapter 2). The radiation patterns from individual arms (each arm being a linear phased array) are multiplied together to form pencil beams. **RMulPat** represents such multiplicative arrays. Each **RMulPat** object consists of two **RRadPat**-derived antenna patterns. For any given direction of interest, the array sensitivity will be determined as the product of the contributions from both elements. Figure 6.5, panel (b) is an example of a **RMulPat** radiation pattern. This is in fact an untapered ARIES beam 595 as returned by the beam factory function (see section 6.6.3).

Panel (c) in the same figure is an object diagram of this very **RMulPat** object. It can be seen how **RAddPat** (for the two additive arrays forming the arms of the Mills Cross), **RPharrPat** (as simplification due to the regular spacing), **RRotPat** (to take the sloping ground into account) and **RMulPat** work together to represent an ARIES pencil beam.

6.3.9 Rotated Patterns: RRotPat

Antenna patterns are not always perfectly aligned with the principal axes of the observer's coordinate system. Instead of re-implementing rotation algorithms for each individual class (type) of radiation pattern, a separate class **RRotPat** was developed. Objects of this class act as a container for any **RRadPat**-derived object, and simply return a rotated version of the original pattern, obtained by rotating the original pattern around the x, y and z-axes (in that order) by the specified angles.

Thus, ARIES fan and pencil beams can be properly rotated in accordance with the actual tilt of the ARIES site, and linear dipole patterns can be placed arbitrarily in free space, despite the fact that a **RLinDipPat** pattern is always aligned with the z-axis.

RRotPat uses a combination of matrix manipulations to correctly transform polarisation information, see for example [Owe]. For each direction, the following steps are taken to produce the correctly rotated version of the radiation pattern:

1. Rotate the required directions around z, y and x-axes (in that order) by the inverse of the specified angles. This is because the specified angles describe the rotation of the radiation pattern, whereas internally we perform the equivalent operation of rotating the observer in the inverse direction.

2. Also rotate the local XY coordinate system (in which the polarisation information is specified) along with the corresponding observer directions.

3. Map the retrieved field intensities Ex and Ey onto the rotated coordinate system. The resulting values are the polarisation information in the correct coordinate system.

6.3.10 FEM Simulated Radiation Patterns: RNECPat

Complex real-life antenna systems can often be simulated using FEM (Finite Element Method). A popular tool employing the NEC FEM engine [BP77] is miniNEC, and several simulations have been run using this tool.

Advantages of NEC simulations over theoretically derived radiation patterns are that they can easily take into account environmental properties such as (imperfect) ground planes, interfering metal surroundings and the material and dimensions of the actual antenna elements.

The disadvantage is that correctly modelling the antenna system under investigation is a labour-intensive, time-consuming task (involving manual steps for each individual beam) and FEM simulation tools have high processing requirements.

RNECPat loads NEC simulation results from a specified ASCII file. Any subsequent **getGain()** requests will return gain values interpolated from the simulation results. Figure 6.6, panel (b), is an example plot of a **RNECPat** object.

6.3.11 MIA Antenna Directivity Adaptor: RMIAPat

S. Marple's Multi-Instrument Analysis (MIA) toolkit ([Marc]) implements a **getdirectivity()** function for instruments of type riometer. Being initially very basic and only suitable for theoretical filled phased array radiation patterns, this function has somewhat improved since the inception of RIOSIM. Given the widespread use of MIA, RIOSIM includes an adaptor object to enable it to directly utilise MIA directivity patterns.

A **RMIAPat** is created simply by calling the **RMIAPat** constructor with two additional parameter-value pairs:

mia_instr to specify the MIA instrument object for which to create a radiation pattern.

mia_beamnr to specify the MIA beam number for which to create a radiation pattern, to be seen in the context of the MIA instrument specified with the **mia_instr** parameter.

As for all **RRadPat**-derived objects, these parameters can also be changed at run-time using **RRadPat**'s **get()** and **set()** methods.

Note that MIA's directivity function does not include support for antenna polarisation, so a **RMIAPat** object will always return the antenna radiation pattern as being entirely x-polarised. This is not a problem for most applications, as they usually deal with antenna (power) gain, i.e. the sum of the squared directivities in the x and y polarisation planes, see also the description of **RRadPat**'s **getGain()** function in section 6.3.1.

6.4 Sky Maps: CSkyMap and Descendants

The second basic class of objects in RIOSIM is the *cosmic sky background*. It is represented by class **CSkyMap** and its children, see figure 6.7. **CSkyMap** itself is able to return its current projection onto the celestial hemisphere as seen by an observer on Earth at any given moment in time. Three children of **CSkyMap** that were implemented initially will be described in this section:

CTaohSkyMap is a 'façade' class [GHJV95, Dea02] for the sky map originally implemented by Huiyu Tao [Tao04]. It uses all the original conversion functions used by Huiyu Tao, as well as the original digital sky map as digitised by Huiyu Tao from a sky map at 30MHz by Cane [Can78].

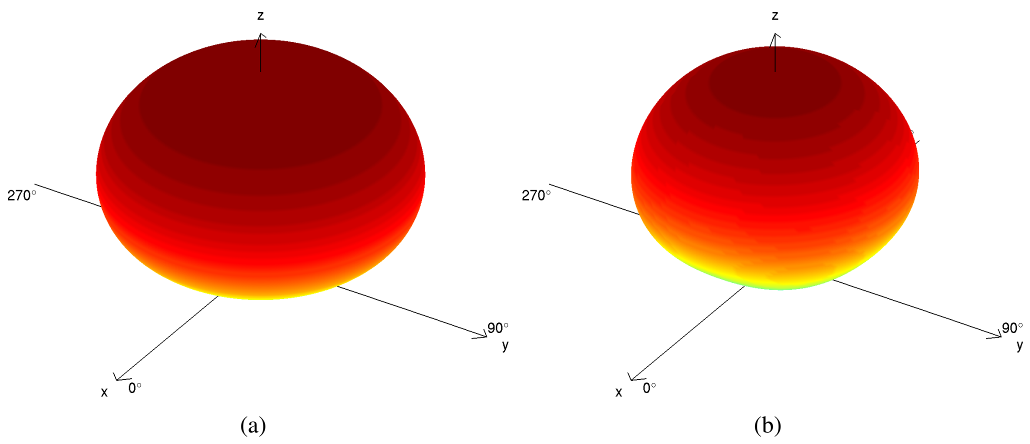


Figure 6.6: Radiation pattern of a crossed dipole above ground. (a) Simplified pattern according to equation 6.2; (b) FEM (miniNEC) simulation above perfect ground. The slightly non-symmetric shape of the FEM simulation is just about visible.

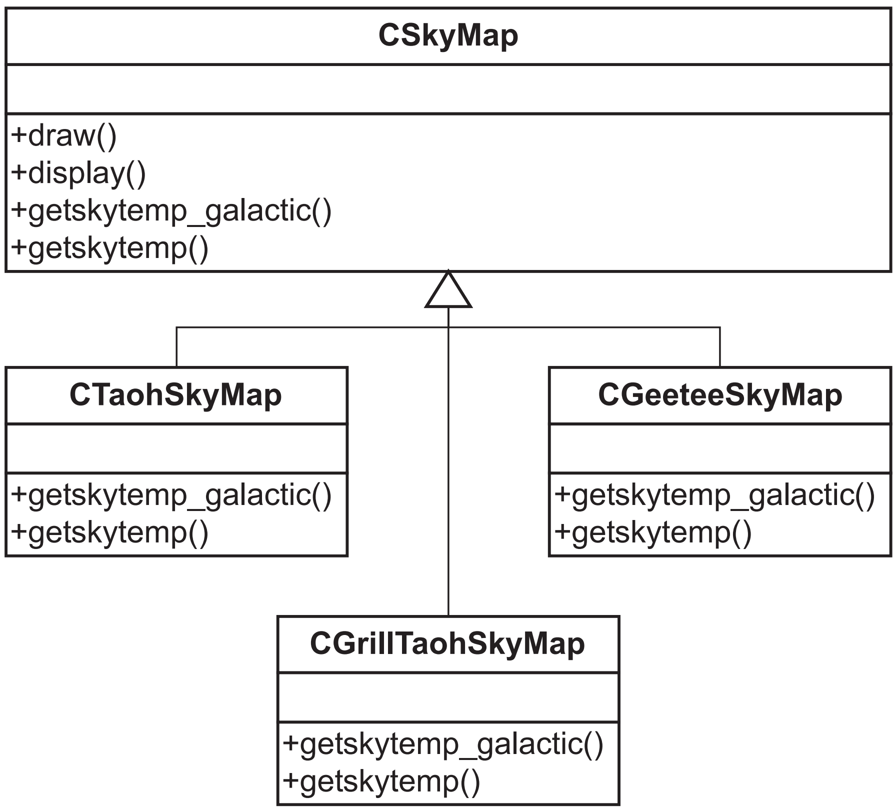


Figure 6.7: **CSkyMap** and descendants

CGrillTaohSkyMap (figure 3.12) still uses the original digital sky map as used by Huiyu Tao, but performs all coordinate conversions using function calls to the third-party SLALIB library [Walb]. Originally thought to be more accurate, it turns out that the gain in accuracy is only minor, as can be seen in figure 6.8, which shows the difference between the sky temperature for a given slice of sky as determined by **CTaohSkyMap** as opposed to **CGrillTaohSkyMap**.

However, using the well-documented SLALIB coordinate conversion functions makes the program flow much more transparent, significantly simplifying future changes to the code.

CSkyMap is designed to be extensible through new child classes, and the above implementations **CTaohSkyMap** and **CGrillTaohSkyMap** are only two possible implementations. In particular, Cane's sky map suffers from low resolution, which becomes very noticeable when simulating reception by antennas with narrow, pencil-shaped radiation patterns. For most ARIES-related simulations in this thesis, a higher-resolution sky map was used [DU90]. This sky map is implemented by **CGeeteeSkyMap** (figure 3.13). Generally, through this object-oriented design technique, sky maps can be swapped in and out as required and existing algorithms can be re-evaluated with higher resolution or better quality sky maps as these become available.

The following is a description of the main methods supported by **CSkyMap**-derived objects:

getskytemp_galactic Retrieves cosmic background temperature in Kelvins for directions as specified by (a set of) azimuth and elevation coordinates in galactic coordinates (see appendix B). For most sky maps, the galactic coordinate system will be the 'native' coordinate system, and this function will simply interpolate temperature values for the requested directions without any further coordinate transformations.

getskytemp Retrieves cosmic background temperature in Kelvins for directions as specified by (a set of) azimuth and elevation (horizontal) coordinates for a given universal time (UT) and location on Earth. This involves converting the observer's horizontal coordinates into the native coordinate system of the sky map. See appendix B for more details on coordinate systems and how to convert between them.

draw Plots an overview of the whole sky map, see figures 3.1, 3.11, 3.12 and 3.13 in chapter 3 for examples.

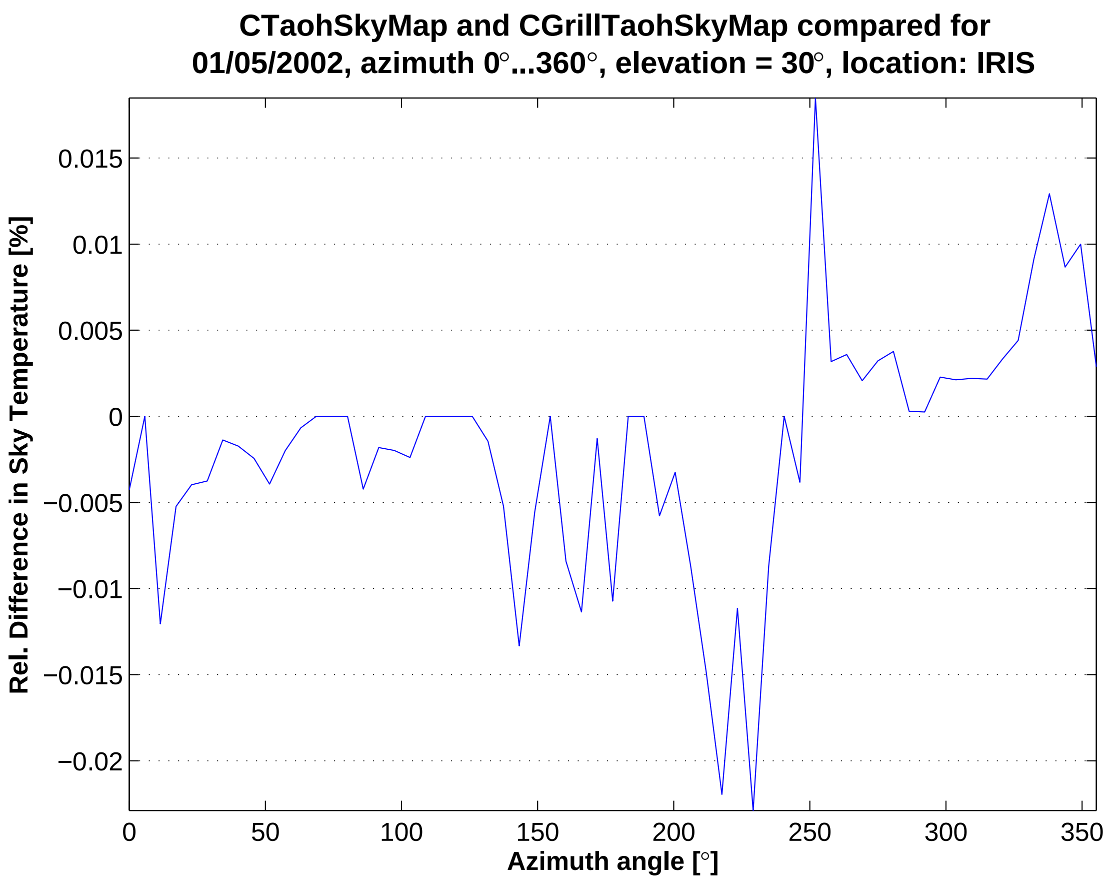


Figure 6.8: Difference in sky temperature as returned by **CTaohSkyMap** respectively **CGrill-TaohSkyMap**

6.5 Radio Stars: CRadioStar

Radio stars are in a way quite similar to sky maps in that their position on the celestial hemisphere depends on the time and place of the observer. Radio stars are represented by instances of the class **CRadioStar** (see figure 6.9), and their main ability is being able to determine their position in the observer's coordinate system at any given time. Given that each radio star can be uniquely identified by only its (galactic) coordinates and its flux density on Earth, an approach following the 'lightweight' design pattern [GHJV95, Dea02] was taken for its implementation. A **CRadioStar** object can be initialised to represent any arbitrary radio star, including fictitious stars such as the North Celestial Pole, using its constructor or the **catalogueLookup()** member function. This can conveniently be done at initialisation time by using code like "`cassiopeia = CRadioStar( 'Cass A' )`."

Each **CRadioStar** object supports the following method for querying its location:

getazel Return position of star in observer's Az-El-coordinate system for the given time and location of the observer.

CRadioStar objects will be used especially in the radio star tracker and scintillation predictor applications in chapter 7.

6.6 Elementary RIOSIM Functions

In addition to the classes described above, RIOSIM comprises several 'elementary' functions. These are not specific to any class, but instead perform general tasks related to the topic of riometer simulations. These are the functions referred to by the orange box on the right-hand side in figure 6.1. While many of these functions are trivial and will not be discussed further here, the following list describes some of the more useful helper functions, and separate sections below have been dedicated to describing the radiation pattern factory (section 6.6.3) and the more complex coordinate transformation functions (sections 6.6.1 and 6.6.2) .

All functions discussed in this section can serve as building blocks for larger-scale real-life applications, some of which will be described in the next chapter (7).

azeltriad Returns base vectors of a Cartesian horizontal coordinate system with the given origin, expressed as multiples of the base vectors of the underlying geographic coordinate system. Useful for establishing a Cartesian coordinate system for an observer located on the surface of the Earth.

pin3d Plots a 'pin' from $O(0 \mid 0 \mid 0)$ to the given spherical coordinates (az,el,radius). If no colour is specified, the default is used (red). Useful for pinpointing locations on a sphere, used for example in figures 4.6 onwards in chapter 4 (which is otherwise quite unrelated to RIOSIM).

scale Scales a set of values on a linear or dB scale, relative to the specified reference value.

stretch Scales a set of values to lie in the range $low \leq x \leq high$, with values exceeding the low and high marks being set to either NaN, exactly the limit or any other specified value.

myscand, myscand_narrow Useful maps: Map projections are useful for many different plots (for an example in this thesis see figure 9.12 in chapter 9) and are used extensively in publications. **myscand()** plots a map of the ARIES site using the M_Map mapping toolbox [Paw05] and MIA's **scand()** function [Marc, function SCAND]. This map can the be used for contour plots, etc., see also the applications in chapter 7, for example figures 7.1 and 7.2. The **myscand()** function is merely a shortcut to quickly arrive at a useful map around the ARIES and IRIS sites. So is **myscand_narrow()**, which produces a zoomed-in version of **myscand()**.

6.6.1 Projecting Rays onto the Spherical Ionosphere: 'projection1'

For a number of applications, it is useful to be able to plot the outline of the main beams (down to, say, the -3dB borderline) projected onto the surface of observation, in our case the ionosphere, which is for this case assumed to be a sphere with the centre of the Earth as origin and a radius $h + r_E$ greater than the radius of the Earth r_E, h being normally assumed to lie in the order of 90×10^3m as discussed in chapter 3.

Figure 6.10 shows the basic geometry. The coordinates of a given contour are in the horizontal coordinate system (see section B.1.5) of the radiation pattern, respectively in its underlying 'mathematical' spherical coordinate system (section B.1.2). The position of the observer is given in geographic coordinates (see section B.1.4), more exactly again in its underlying 'mathematical' spherical coordinate system.

After performing the operations described in B.2.1, the relative position of the two coordinate systems (I — red and II — green) involved are known, and now the vector $\overrightarrow{Q}$ can be found

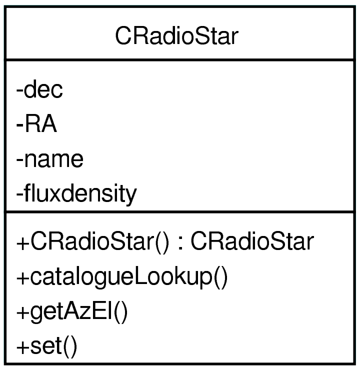


Figure 6.9: **CRadioStar**

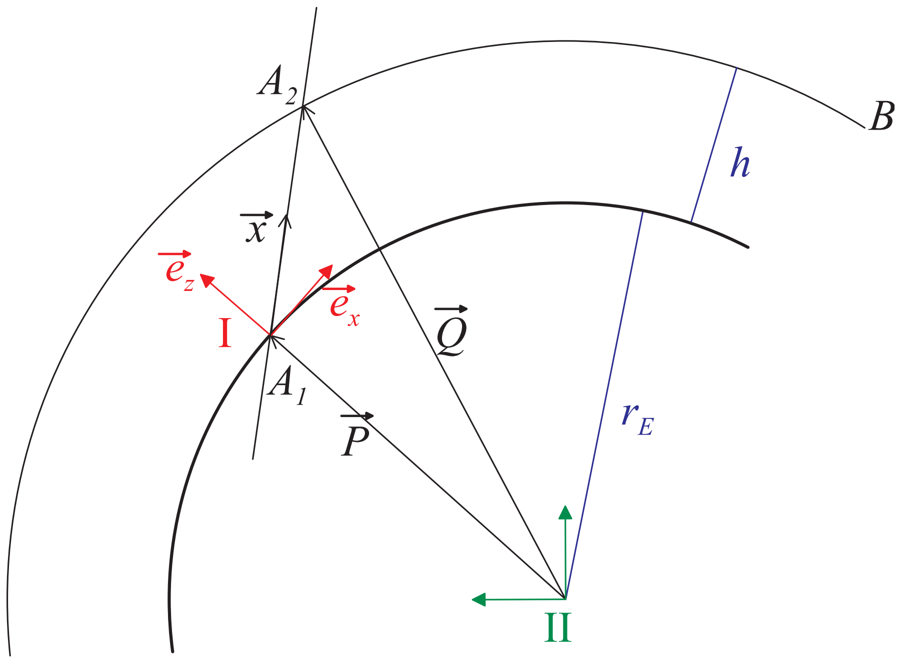


Figure 6.10: ‘Projection 1’: Projection onto the ionosphere

as the vector from the centre of the Earth to the intersection of line (A_1A_2) with the ionosphere as follows:

In coordinate system (I) we can describe the sphere B as follows:

$$x^2 + y^2 + (z + r_E)^2 = (r_E + h)^2 \tag{6.3}$$

(OX) can be expressed as

$$\overrightarrow{x} = \lambda \cdot \overrightarrow{c} \tag{6.4}$$

where $\overrightarrow{c}$ represents vector in direction of the point on the contour in question.

We can now calculate the intersection of line (OX) and sphere B, and with $\overrightarrow{c} = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}$ find that

$$\lambda = -c_z r_E + \frac{1}{2}\sqrt{(2c_z r_E)^2 + 4 \cdot 2 r_E h + 4h^2}. \tag{6.5}$$

Vector $\overrightarrow{Q}$ expressed in coordinate system II is therefore

$$\overrightarrow{Q} = \overrightarrow{P} + \lambda \cdot \overrightarrow{x} \tag{6.6}$$

$$= \overrightarrow{P} + \lambda \cdot (x_x \overrightarrow{e_x} + x_y \overrightarrow{e_y} + x_z \overrightarrow{e_z}), \tag{6.7}$$

where x_x, x_y and x_z specify the Cartesian coordinates of $\overrightarrow{x}$, and $\overrightarrow{e_x}$, $\overrightarrow{e_y}$ and $\overrightarrow{e_z}$ are the base vectors of coordinate system I expressed in coordinates of system II as calculated in section B.2.1.

The coordinates of vector $\overrightarrow{Q}$ can now be expressed in geographical coordinates, and these can be plotted onto a map.

This projection is being used in many applications. One example can be seen in figure 7.2, discussed in section 7.2, where it is used to project the the track of a radio star as well as IRIS beam contours onto the ionosphere at 90km height. Another example is the projection height evaluation in chapter 9, see figure 9.12.

6.6.2 Projecting a Spherical Cap onto a Flat Plane: 'FLATM Projection'

Antenna radiation patterns and sky brightness distributions are usually, and most intuitively, described in a spherical coordinate system. The mathematical spherical coordinate system (see B.1.2) in this thesis uses a spherical coordinate system made up from azimuth angle θ (in the XY plane, counterclockwise, starting from the positive x-axis) and elevation angle ϕ (between XY plane and direction of interest, positive ϕ denotes positive z). For visualisation purposes and further studies, a dataset on such a spherical grid often needs to be mapped onto a flat two-dimensional surface. For IRIS absorption images, a projection algorithm that we will henceforth call the 'FLATM' (for 'flat metres') projection is commonly used. This is a two-stage projection that first calculates the intersection between a ray in the given direction and the ionosphere (at a given height, 90km by default) and then maps the result onto a flat surface. The whole process is shown in figure 6.11. This projection offers relatively little distortion around the zenith, with distortion increasing with lower elevation angles. As the surface area of the ionosphere is much greater than the area usually covered by any given instrument, the FLATM projection gives a good (i.e. relatively undistorted) representation of the signal distribution at the height of the ionosphere (or any other sphere centred on the Earth's centre for that respect).

Note that of the interpolation algorithms discussed in chapter 10, only the original IRIS interpolation algorithm inherently uses (relies on) the FLATM projection. The GLEAM algorithm inherently interpolates in the coordinates of the underlying basis functions, only the results are generally mapped back to FLATM projections for visualisation and comparison purposes.

6.6.3 Riometer Beam Factories: getbeampat() and getspecialbeampattern()

Constructing a **RRadPat** from its basic components can be a tedious task. Depending on the complexity of the antenna, one needs to know about antenna spacings, phasing factors, physical locations of the aerials, slope of the ground, etc. It seems therefore desirable to put this knowledge into writing once, and then have the computer generate the appropriate **RRadPat** object itself. A function which does this is also known as a factory function [Dea05], or just factory. All the user needs to know is that it returns an object of type **RRadPat**. There is no need to care or even know about the exact procedures and parameters that were used to create this object, nor, indeed, the exact (**RRadPat**-derived) type of object returned.

The function **getbeampat()** is such a factory function. It takes the ID of a known riometer that we want to create a **RRadPat** for and the number of the beam that we require. It returns a

RRadPat object representing the radiation pattern of the beam in question.

At the moment, **getbeampat()** knows the following riometer IDs:

kil The IRIS riometer at Kilpisjärvi.

tro The untapered 32+32 ARIES system.

trofan The fan beams of the untapered 32+32 ARIES system.

tr2 Untapered ARIES 16+16 system.

tr2fan Untapered ARIES 16+16 fan beams.

ram Tapered ARIES 32+32 system.

ram2007 ARIES system with digital beamforming as used from 2007 onwards.

special Beam definition will be retrieved using **getspecialbeampattern()**, see below.

getspecialbeampattern()

Especially during the preparations for the ARIES October 2002 experiment, it became necessary to simulate a variety of custom configurations for the given 32+32 ARIES antenna array. Thus, the **getspecialbeampattern()** function was developed. Similar to the **getbeampat()** function described above, it takes a beam number and returns an appropriate **RRadPat** object representing the radiation pattern of that beam. However, the meaning of the given beam number is taken from a special ASCII file called `specialbeams.txt`. This file describes, how the radiation pattern for any given beam number can be obtained, and **getspecialbeampattern()** uses this information to create an appropriate **RRadPat** object.

Appendix F contains an example of such a beam pattern definition file. This is the file that was used for the simulations for the October 2002 experiment, many beam numbers mentioned in chapter 9 refer to the numbering scheme adopted in this file.

6.7 Summary

We introduced, and described the architecture of, a universal toolkit ‘RIOSIM’ for simulating and visualising the behaviour of real-life riometer systems. We introduced the major players (radiation patterns, radio stars and sky maps) and showed various example plots demonstrating the use of these objects.

The RIOSIM toolkit is the logical progression from earlier lower-level simulations presented in chapters 4 and 5. RIOSIM simulates system behaviour for arbitrary riometers or, in fact, antenna systems. The ability to do so is essential for the development of new instruments such as ARIES, enabling simulation of their behaviour long before any hardware has been developed or deployed, and later confirming that the real (prototype) system performs according to specification. In the following chapter we will discuss some more advanced applications making use of this toolkit.

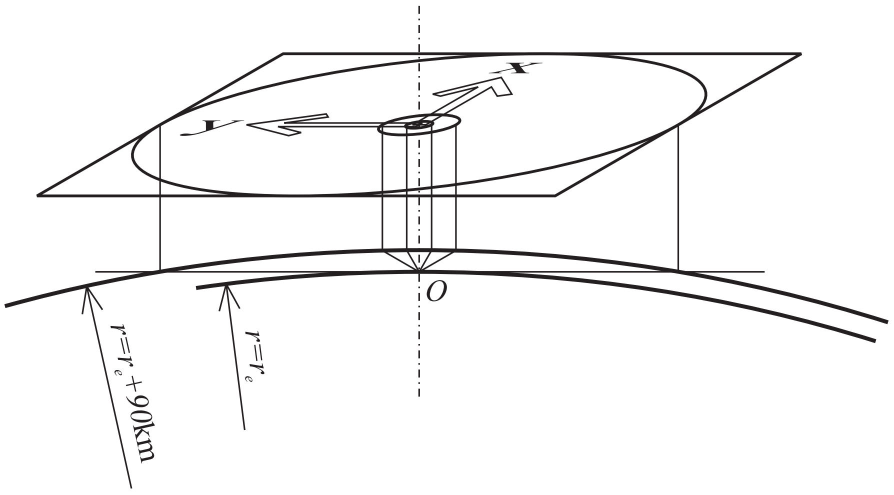


Figure 6.11: FLATM projection as used for IRIS image data. Drawn to scale. Observer (instrument) is situated at O. Published in [GSH05].

Chapter 7

Applications of the RIOSIM Toolkit

This chapter describes some advanced applications of the RIOSIM toolkit developed by the author. RIOSIM was designed with many of these applications in mind (see chapter 6), and this chapter aims to prove that RIOSIM fulfils these expectations and has in fact contributed to the successful deployment of ARIES in various ways, as many of the applications presented here have been used during initial investigations and deployment of the ARIES riometer. Some have also proven useful for existing riometers, first and foremost IRIS. It should be clear from the discussions in the previous chapters, that many of the applications presented here can easily be implemented for other riometers, usually by simply changing the location, time and beam pattern parameters appropriately.

7.1 Plotting Beam Contours onto the Ionosphere

One of the basic aims of *imaging* riometry is to spatially resolve the absorption information obtained by the riometer. As absorption occurs within a well-defined height region, the region illuminated by any given beam (as defined by the corresponding radiation pattern) can be determined by means of the 'projection1' algorithm described in section 6.6.1. However, all practical radiation patterns do not exhibit sharp corners, i.e. do not expose sharp beam boundaries. Instead, for practical purposes, the -3dB cut-off is often used, i.e. the principal beam shape is considered to be defined by the beam's -3dB contour. A signal received at this boundary will be attenuated by 50% compared to reception at the point of maximum sensitivity.

The -3dB beam contours projected onto the ionosphere can be thought of as the 'footprint' of the given beam in the ionosphere.

The code for plotting these footprints of a given radiation pattern is not included in the **RRadPat** class, for the reason that the processing steps required do not only involve **RRadPat** objects, but also a variety of other objects and algorithms. A basic axiom of object-oriented design is to keep the functionality of every single object limited (stick to the core competencies — high cohesion), and to make it know as little as need be about its surroundings (loose coupling) [Som04]. That is why, in this case, plotting beam contours onto the ionosphere is implemented as a separate program. It makes, of course, use of **RRadPat** to retrieve the direction vectors of the contour (this is core knowledge of the radiation pattern, internally implemented using the MATLAB **contour()** function [Matb, function CONTOUR] in current **RRadPat** implementations), but in addition to that it also uses

- a map to plot the contours onto (using the mapping toolbox M_Map [Paw05])
- algorithms for projecting these contours onto the ionosphere (provided by **projection1()** as described in section 6.6.1)

In particular, the following contour plotting functions are provided:

plotcontour()

The **plotcontour()** function simplifies the task of plotting a beam contour on a map using the M_Map mapping toolbox [Paw05]. It assumes that a map has already been created, for example by MIA's **scand()** function or the **myscand()** function described in chapter 6, section 6.6. Given a radiation pattern and a location on the Earth where this radiation pattern originates, **plotcontour()** then plots the specified contour. Additional parameters allow the user to specify the projection height (default 90km), the plotting colour, whether the contour should be labelled and additional text to be printed at the centre of the contour. Also, some parameters get passed down to **RRadPat**'s **getcontour()** function, namely the contour level (default -3dB) and the resolution of the grid to use for the contouring algorithm (defaults to the radiation pattern's default resolution).

No separate example for the output of **plotcontour()** is given here, the reader is referred to figure 7.1 below, demonstrating the output of the **plotmanybeams()** function that relies on **plotcontour()** to plot each individual beam. Also, **plotcontour()** and its relatives are used extensively throughout this thesis, other figures that use **plotcontour()** in various configurations are, for example, figures 3.1, 9.8 and 9.12.

plotmanycontours()

For studies involving the sidelobes of beams, a plot with many different contour levels of a single radiation pattern is often useful, and this can easily be obtained with the **plotmanycontours()** function. It behaves very similarly to the **plotcontour()** function described above. In addition to all parameters supported by **plotcontour()**, **plotmanycontours()** also supports parameters for specifying the number of contours to be plotted, the stepsize from contour to contour, and whether to plot a legend (colourbar).

For an example plot, the reader is referred to the following section 7.2. Its figure 7.2 shows a many-contoured plot of IRIS beam 38, together with the traces of Cassiopeia and Cygnus during the course of one day. Figure 7.3 is a plot of the actual received data for the day in question (a relatively quiet day). The time when Cygnus passes through the main lobe (around 12:00UT) can be clearly seen in the data. The reason for the observed increase in scintillation around 16:00UT would be less obvious if only the -3dB beam contour were plotted. By using **plotmanycontours()**, the increase in scintillation is seen to be directly associated with Cassiopeia passing through a sidelobe at this time.

plotmanybeams()

It is often desired to plot the contours of several beams onto one map, for example for visualising the field-of-view of a multi-beam (imaging) instrument. This can easily be achieved with the **plotmanybeams()** function which itself calls **plotcontour()** for every beam it wants to plot. **plotmanybeams()** retrieves the required radiation patterns using the **getbeampat()** beam pattern factory, see section 6.6.3.

To illustrate the output of **plotmanybeams()**, and therefore also the output of the underlying **plotcontour()** function, figure 7.1 shows the -3dB beam outlines of some of the central ARIES beams and the 45 'good' IRIS beams at 90km height, plotted onto an appropriate map of Scandinavia.

Non-standard Contour Plotting

Note that beam contouring is not limited to the applications described above. The contour information returned by **getcontour()** can also be used for other types of plots. Figure 3.1 in chapter 3, for example, is a map of the whole sky (produced by M_Map [Paw05]), with the beam outline of IRIS beam 31 for every hour during one 24h cycle.

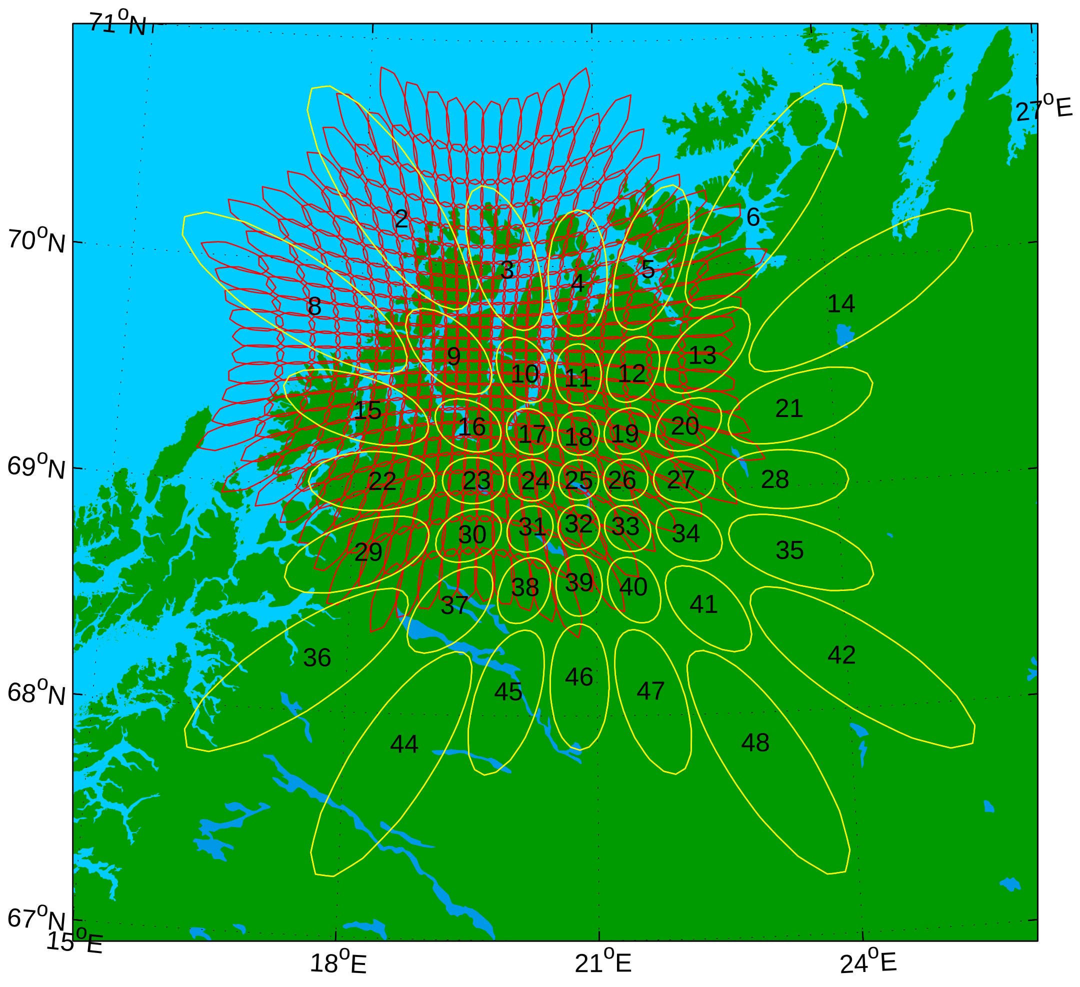


Figure 7.1: ARIES central beams (red, unnumbered) and IRIS 'good beams' (yellow, numbered) beam contours as produced by **plotmanybeams**(). Projection height 90km.

7.2 Radio Star Tracker

For calibration and validation purposes, the movement of a radio star with time is often of primary interest. For example, the passing of Cassiopeia through the main lobe of a beam was used in the 2002 experiment to validate the fact that we were actually forming a pencil-shaped beam of the predicted size, and to verify the exact pointing direction of the beam. See chapter 9 for more details on the October 2002 experiment.

Given an instance of **CRadioStar**, with its inherent capability to calculate its apparent location for any given moment in time, and combining this with the 'projection1' method as used above for contour plots, we can now implement a program to plot the diurnal path of the particular radio star represented by the **CRadioStar** object. If the path of the radio star is projected onto the ionosphere together with a radiation pattern of a given instrument, the resulting graph will give in a graphical form information as to when the given radio star passes through the given beam of the instrument. This particular fact can be used as a form of scintillation prediction, an application discussed in more detail in section 7.7.

A MATLAB program **traceradiostar()** was developed to perform the task of plotting radio star traces. Given a specific date and a **CRadioStar** object, it plots the trace of that radio star for the given date. Details about the necessary coordinate transformations (projection onto the ionosphere) can be found in section 6.6.1.

Figure 7.2 is one example output of the radio star tracker. Included are the projection of the radiation pattern of IRIS's beam 38 onto the ionosphere at 90km height (see section 7.1), a map of the relevant part of Scandinavia and the traces of the two major radio star Cassiopeia and Cygnus for 20 January 2001. The time ticks are in UT (Universal Time, see section C.3).

Figure 7.3 is actual received (power) data for the day in question (a relatively quiet day). The time when Cygnus passes through the main lobe (around 12:00UT) can be clearly seen in the data. The reason for the observed increase in scintillation around 16:00UT can be seen to be associated with Cassiopeia passing through a sidelobe at this time, an observation that is simplified by the multiple contour levels in figure 7.2.

7.3 Simulated Reception: rxskymap() and Relatives

To simulate reception of cosmic noise through an antenna-receiver-system, not only does one have to know the radiation pattern of the antenna and the brightness distribution of the sky at

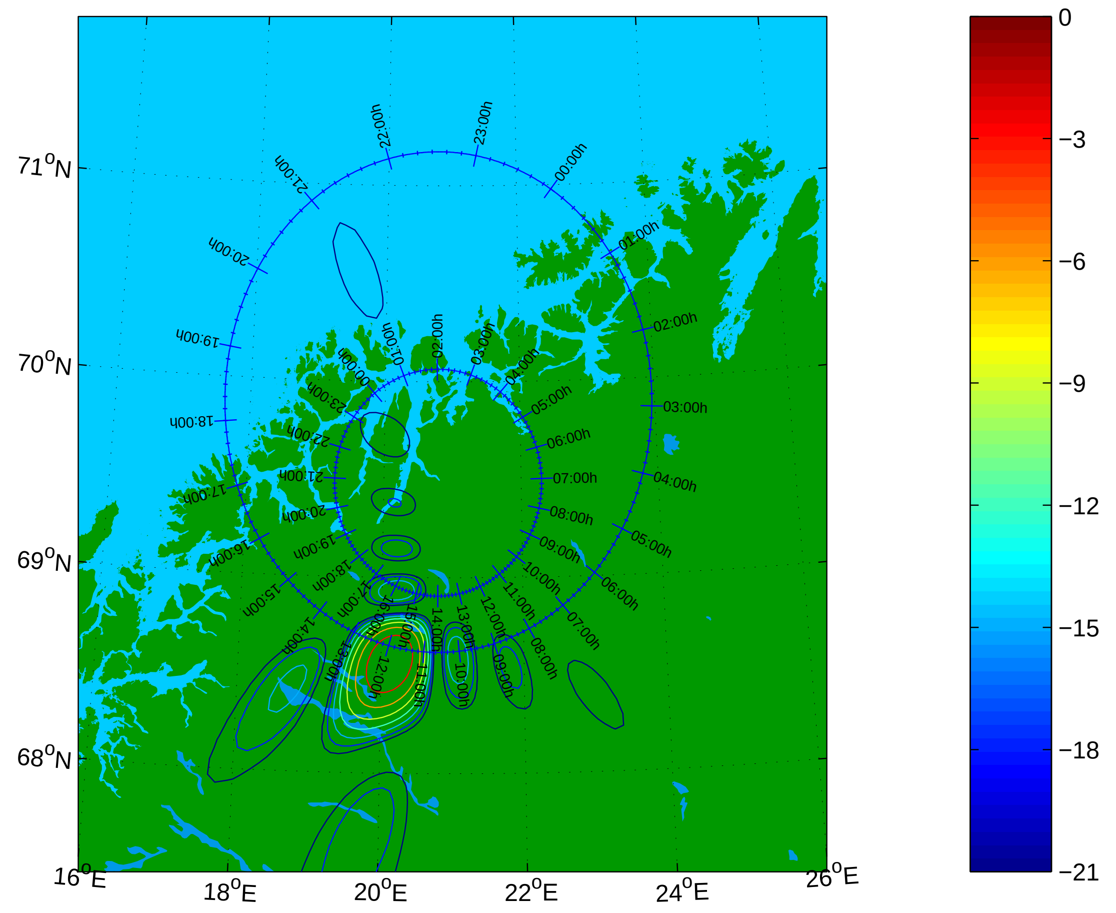


Figure 7.2: Multiple contour levels for one radiation pattern (output from **plotmanycontours()** for IRIS beam 38) together with radio star traces for 2001-01-20 (output from **traceradiostar()** for Cygnus — outer trace and Cassiopeia — inner trace). See figure 7.3 for a plot of actual IRIS power data for beam 38 on the day in question.

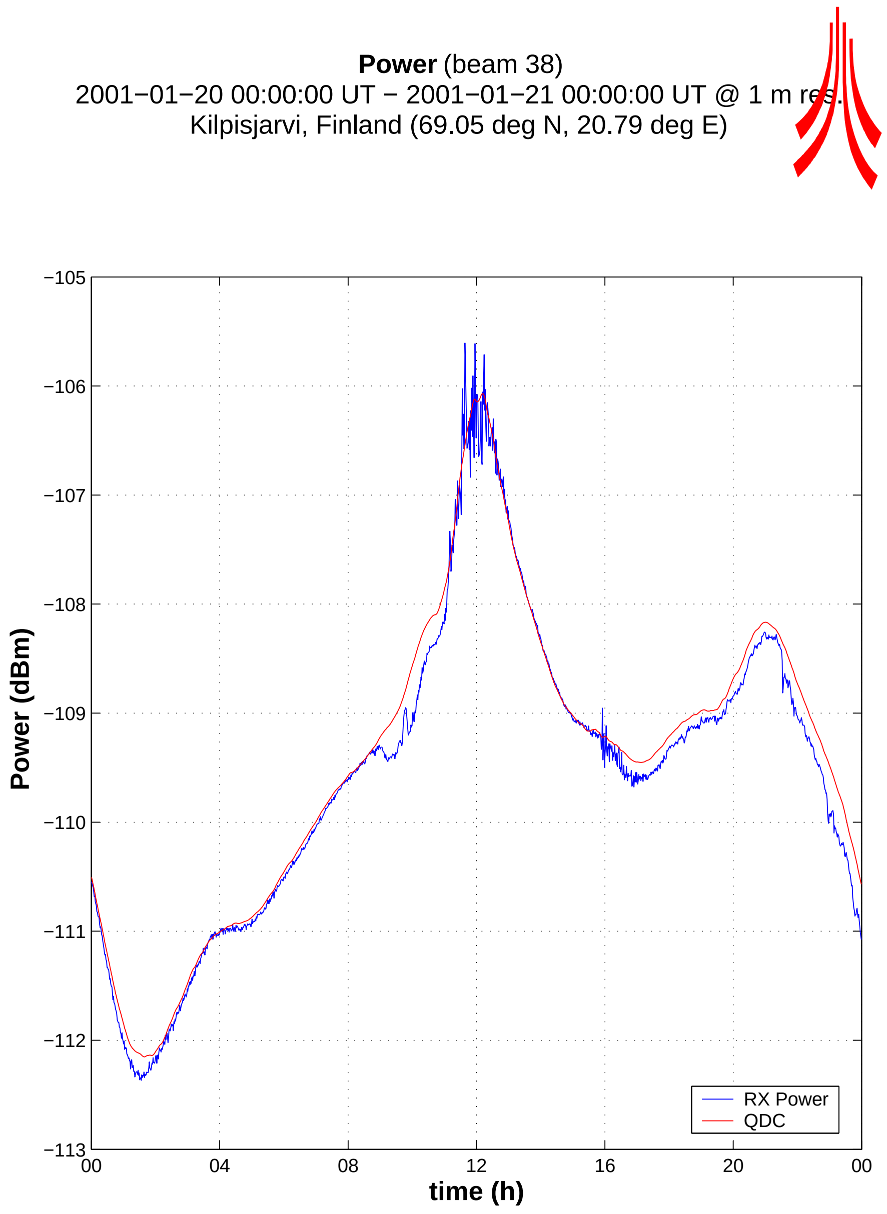


Figure 7.3: IRIS power data for beam 38 on 2001-01-20

the given frequency, but receiver bandwidth and current time and location also play a role. The **rxskymap()** function takes all these parameters, simulates reception by numerically integrating the convolution of radiation pattern and sky brightness distribution and returns the received power in Watts.

Note that **rxskymap()** and related functions (below) have been superseded by the QDC generator described in section 7.4 below. We therefore keep the description deliberately short.

rxskymap_day_fast()

Similar to the **rxskymap()** function, **rxskymap_day_fast()** simulates reception for one whole UT day at a given time resolution. For a riometer beam pattern and a sky map representing cosmic noise background, this will give the theoretical QDC for the specified day. The suffix _fast indicates that this function does not simply call **rxskymap()** repeatedly. Instead, it uses a more efficient algorithm, retrieving the radiation pattern only once. This, however, means, that **rxskymap()** and **rxskymap_day_fast()** have no program code in common. Therefore, any algorithm changes to **rxskymap()** need to be applied to **rxskymap_day_fast()** separately.

rxskymap_day_fast_all_beams()

This function is built around **rxskymap_day_fast()**. Instead of receiving the QDC for one specific beam, it retrieves the QDCs for all beams specified. It retrieves the respective radiation patterns using the **getbeampat()** factory function.

7.4 Quiet-Day Curve Generator

7.4.1 Introduction

The fact that we can have detailed information about the radiation pattern of a given (proposed) instrument as well as about the sky brightness distribution implies that we can simulate the power received by an antenna with that given radiation pattern, located at a known position on the Earth.

If we do this for one complete (sidereal) day, we end up with a theoretical quiet-day curve (QDC, see chapter 3). This curve will, of course, not contain any absorption effects, since it is based on a static map of the sky brightness distribution and does not take the ionosphere into account at all.

The theoretically derived QDC is likely to deviate from the real recorded signal, even on a perfectly quiet day, due to inaccuracies of the sky map in use and the inherent inaccuracies in any theoretically derived radiation pattern. For example, most sky maps do not cover the entire celestial sphere, so some areas consist of interpolated data. Some sky maps where originally recorded at different frequencies, and all sky maps will contain to some extent sidelobe effects from the original instrument that was used to record the map. Refer to section 6.4 for a list of the sky maps that are currently integrated into RIOSIM.

Note that due to the object-oriented structure of RIOSIM, new sky maps can be added as they become available, and they will seamlessly integrate with all the existing functions. Of course, we can also use several sky maps simultaneously. This enables us to compare results obtained from different sky maps to indirectly compare the suitability of the various sky maps for the task at hand.

7.4.2 Mathematical Background

To generate a quiet-day curve, one simply loops through the time span in question. For each moment in time, the convolution of antenna radiation pattern and radio background noise gives the (simulated) received power. Plotting the resulting power values over time gives the QDC for the period of time in question.

The exact formula that needs to be evaluated is given in, for example, [Tao04] as

$$P_r = k \cdot T_A \cdot \Delta f \tag{7.1}$$

where P_r is the received power, k is Boltzmann's constant and Δf is the bandwidth of the receiver.

T_A is the antenna temperature in Kelvin, calculated as

$$T_A = \frac{\int T_B(\theta,\phi) \cdot G(\theta,\phi) \cdot \sin\theta \, d\theta d\phi}{\int G(\theta,\phi) \cdot \sin\theta \, d\theta d\phi} \tag{7.2}$$

where $T_B(\theta,\phi)$ is the sky background temperature in Kelvin at the given direction as returned by **CSkyMap::getSkyTemp()** and $G(\theta,\phi)$ is the antenna gain in the given direction as returned by **RRadPat::getGain()**.

To evaluate 7.2 numerically, the integral needs to be evaluated as a sum. Care must be taken when evaluating this sum. For infinitely small steps $\Delta\theta, \Delta\phi \rightarrow 0$, the discrete sum and the integral become the same. However, as the resolution within the simulation is not quite infinite — in fact, only around 200 elevation angles are normally used for the sake of processing speed — care must be taken to match the internally used resolution to the requirements of the radiation pattern under investigation. The default resolution of the desired **RRadPat** object is usually a good starting point.

7.4.3 RIOSIM Implementation: maketheoreticalqdc()

Several versions of quiet-day curve generators have been implemented since the inception of the RIOSIM toolkit. The latest version, **maketheoreticalqdc()**, is the most versatile one, and interfaces well with S. Marple's MIA toolkit [Marc] in that it directly outputs **mia_qdc** objects. Previous development versions were optimised for certain processing patterns, for example by relying on an externally-instantiated sky map when deriving multiple QDCs. While this approach did show advantages in terms of the processing time required for certain simulations, **maketheoreticalqdc()** sacrifices speed for versatility, enabling the user to create QDCs for arbitrary radiation patterns and using arbitrary sky maps without having to know anything about the internal workings of the quiet-day curve generator.

maketheoreticalqdc() can take the following parameters, but will use reasonable default values for most parameters if omitted:

res_az, res_el Resolution for internal grid used during the discrete summation (equation 7.2).

instrument Instrument ID for which to calculate the QDC, this gets passed on to the radiation pattern beam factory (see section 6.6.3) to create the actual radiation pattern objects for the requested beams.
This can also be a MIA instrument object, in which case a **RMIAPat** adaptor (see section 6.3.11) is used to utilise MIA's directivity information instead of a native RIOSIM radiation pattern for the QDC reception process.

beams The beam numbers for which to generate QDCs.

skymap The **CSkyMap** object to be used for simulating the reception process.

time Date around which the QDC will be calculated. The QDC will be generated for one sidereal day, starting with sidereal midnight closest to the specified date. This parameter is only of very limited use, as theoretical QDCs will be identical for all sidereal days. Real

QDCs, however, potentially vary with seasonal parameters such as snow depth, hence a MIA QDC object knows about date and time.

resolution Time resolution for the QDC. The lower the time resolution (higher time span values), the quicker the simulation.

location Instrument location. (Take care when specifying existing MIA instruments with the instrument parameter, in which case the location parameter will overwrite the instrument's inherent location.)

bandwidth Instrument bandwidth (overwrites default bandwidth) .

showtimebar Show a graphical progress bar using Chad English's **timebar()** function [Eng02] (turn off for non-interactive use) .

offset Arbitrary offset for post-calibration in dBm, used to shift the generated QDC up or down.

QDCs are returned in a way consistent with the existing MIA toolkit [Marc], namely **rio_qdc** objects. Therefore, theoretical QDCs can be substituted in all the places where real QDCs would otherwise be used.

Figure 7.4 shows a set of theoretical QDCs (red) for each IRIS beam, plotted using the standard MIA toolkit functions. Underlayed are real QDCs as measured by IRIS (blue). An offset has been introduced to match the absolute power of theoretical and measured QDCs.

The QDCs in figure 7.4 were generated using a **CTaohSkyMap** object and a **RPharrPat** radiation pattern object, consisting of **CXDipNielsenPat** element patterns (the **RRadPat** object returned by the riometer beam pattern factory for the 'kil' instrument). As can be seen, the theoretical QDCs fit the real measurements very well. The somewhat flattened peaks in the theoretically derived QDCs as compared to real data stem from the fact that the used sky map contains only the continuous background noise, not the bright radio stars (see chapter 3 section 3.6).

7.4.4 Predicted ARIES QDCs for the 2002 Experiment

Armed with the quiet-day curve generator from the previous section, it is now possible to derive simulated quiet-day curves for the beams of the (at the time non-existent) ARIES riometer. At the time, no real data was available for comparison purposes, as ARIES pencil beams were of

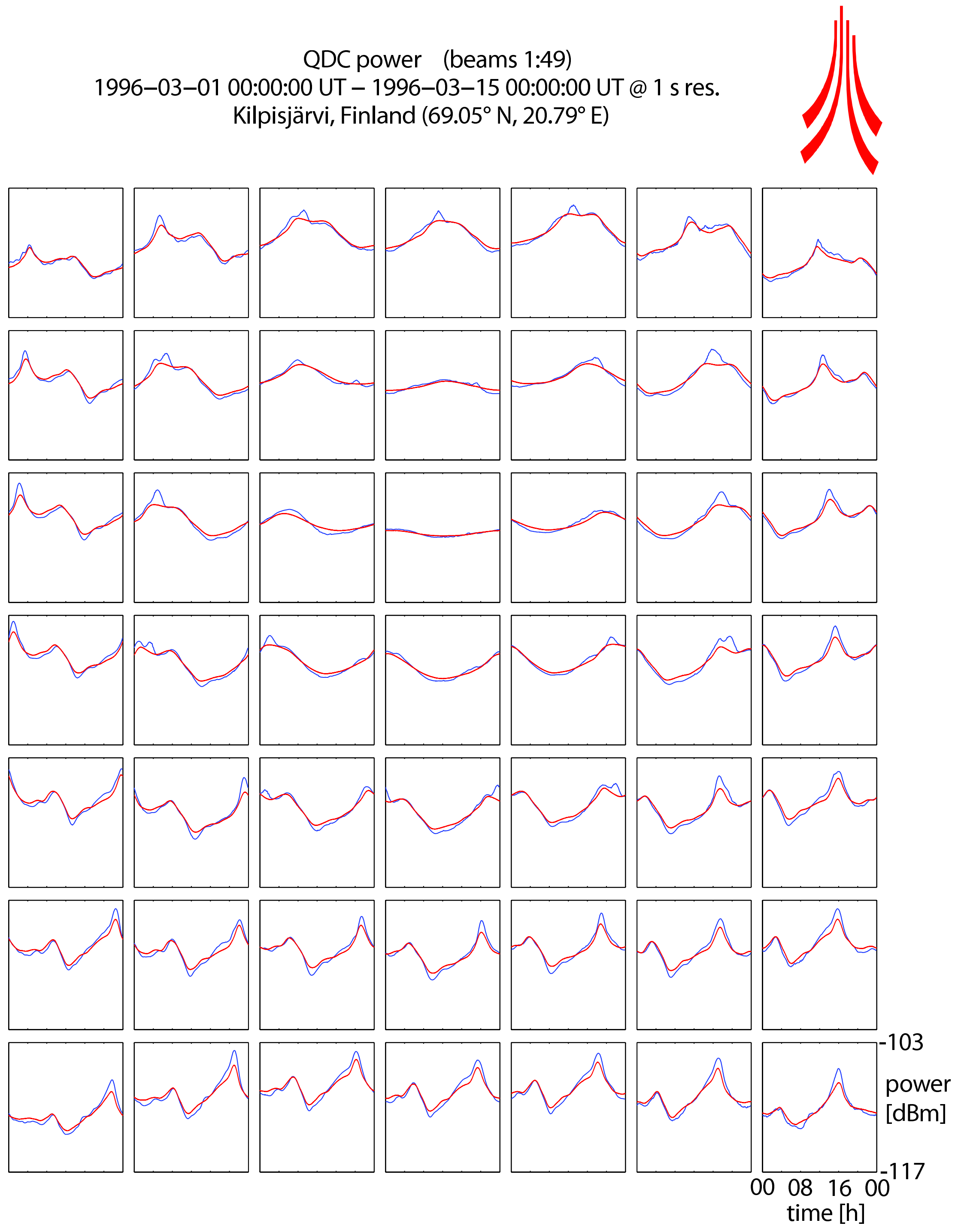


Figure 7.4: Theoretical QDCs for IRIS (red) compared to real QDCs (blue). An offset has been introduced to align the curves vertically.

yet unprecedented thinness, and these simulations were an important step towards understanding what sort of fluctuations to expect from the new instrument.

Quiet-day curves were derived for all the central pencil beams, as well as for all the special beams as described by `specialbeams.txt` (see the description of the riometer beam radiation pattern factory in section 6.6.3).

These curves were used for validating the received signal from the actual array during the 2002 experiment, and we will present a number of comparisons and results in chapter 9.

At this stage, we will simply present an overview plot of all QDCs for all central ARIES pencil beams, see figure 7.5. This figure provides an insight into how much signal variation to expect in any given beam. We can clearly see the relatively low variation in beams that point near the celestial pole (around beams 304 and 305). Also, large variations due to beams passing through the bright galactic plane and radio stars can be seen very clearly.

This is a good starting point for trying to identify a 'worst-case' beam for real-life investigation during initial experiment setups. We will expand on this topic in the next section.

7.5 Determining the Worst-Case ARIES Beam

For the first tests of the working principle of a cross-correlation riometer in the field, only a limited amount of prototype receiver hardware was available, and only a limited amount of beams could be formed simultaneously (see chapter 9). It is therefore a good idea to come up with some means of determining sensible beams to be investigated during such initial experiments.

One obvious choice is the zenithal beam due to the fact that it requires no phasing offsets and is formed simply by adding together all the signals from the respective fan beam before performing the cross-correlation (see chapter 2 for details on phased arrays and the working principle of a cross-correlation array).

However, the zenithal beam is unlikely to be the worst-case beam in terms of system performance. In fact, a quick glance at some simulated quiet-day curves in figure 7.5 shows that the zenithal beam does not appear to be particular 'special' in that it does not show a lot of diurnal variation.

The discussion and simulations in chapters 4 and 5 clearly indicate that required integration time and noise level are directly related to the ratio of power received from the fan beams compared to the power received in the overlapping area (the pencil beam). With respect to this, we

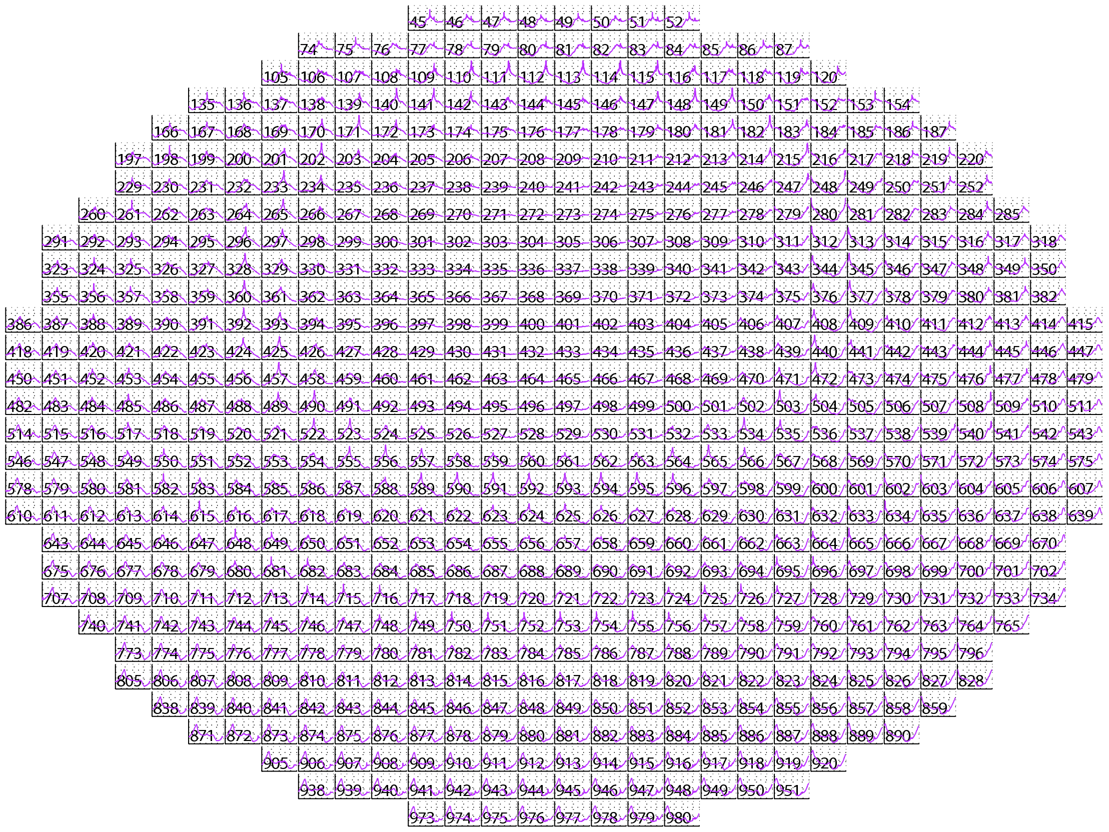


Figure 7.5: Simulated QDCs for the 716 ‘existing’ ARIES pencil beams for one day (x-axis). Y-axis in logarithmic power units, ranging from -102dB to -92dB.

can define the worst-case situation to be one with maximum power in one or both of the fan beams, but low power in the actual overlapping area (the pencil beam).

To visualise how these ‘bad’ situations develop during the course of a sidereal day, a program was developed that generates a time-lapse movie for one complete rotation of the Earth. Figure 7.6 shows one frame of this movie for the ARIES 32+32 antenna array.

The left panel shows simulated received power for all pencil beams (a snapshot of the simulated QDCs for the given moment in time). The right panel shows the power ratio

$$r = \frac{\text{power in both fan beams}}{\text{power in the corresponding pencil beam}}.$$

Thus, high r values represent ‘bad’ situations.

As can be expected, the worst situations occur when radio stars pass through the pencil or fan beams. Out of several likely candidates, beam 595 was selected as the ‘primary worst-case beam’ for the following reasons:

- At some stage, Cassiopeia passes directly through the centre of the beam.
- At other times, the beam points at relatively quiet parts of the sky.
- During these times, both Cassiopeia and Cygnus pass through the fan beams, resulting in a high fan beam to pencil beam power ratio.
- This beam is relatively close to zenith, well within the primary area of interest (the anticipated field of view) for the final instrument.
- Due to its relatively zenithal pointing direction, the beam can also be expected not to suffer from radiation pattern distortions and reflections that might appear closer to the horizon.
- Lastly, although this is of course true for most beams, it is worth mentioning that the beam overlaps well with existing IRIS beams, enabling comparisons between recordings from the prototype instrument (ARIES) and from a long-running, reliable existing instrument (IRIS).

The discussion of the 2002 experiment (chapter 9) contains various comparisons between simulated and actual received data for this beam, amongst others.

7.6 Radiation Pattern Explorer RP

While the plotting capabilities of a **RRadPat** object are quite advanced, they lack interactivity inasmuch as a new plot command needs to be issued for every plot parameter change. The Radiation Pattern Explorer (RP) fills this gap by providing an interactive way to view (explore) **RRadPat** objects, thereby helping the user to quickly get acquainted with the properties of any given radiation pattern.

Figure 7.7 shows the main window of the Radiation Pattern Explorer as it appears when invoked from the MATLAB command line with the `RP` command.

The Radiation Pattern Explorer window is divided into the following main areas:

1. The top line of the window is the radiation pattern selector. It is automatically populated with all **RRadPat** objects that exist in the current MATLAB workspace. These will have been created manually or through the use of some factory tool, for example the **getbeampat()** function described in section 6.6.3 or the **make_sample_radpats()** function that creates some example **RRadPat** objects.

2. The 3D view of the selected radiation pattern. This is essentially a plot as produced by the **plot3()** function (see section 6.3.2). Checkboxes at the bottom of the plot allow the user to switch on additional supporting elements like a semi-transparent sphere of radius 1 and two different versions of three-dimensional coordinate axes. There are also controls for rotating the plot in azimuth and elevation directions.

3. The polarisation explorer. To investigate the antenna polarisation for arbitrary directions, this section provides controls for moving an 'observer' (the red arrow in the 3D view) to any direction of interest. There is also a button to couple observer and camera movement. The polarisation plot to the left of the controls will show antenna polarisation (in the Ex-Ey-plane) at the given observer direction. For example. if the 'observer' is moved from zenith towards the horizon of a linear array of (FET simulated) crossed-dipole radiation patterns, one can clearly observe how the circular polarisation deteriorates more and more into linear polarisation for lower elevation angles.

4. The plot parameters section. This enables the user to change a variety of plotting parameters without having to issue new **plot3()** commands each time. The terms used for the colour and radius properties can be selected from a list of predefined options. Scaling for

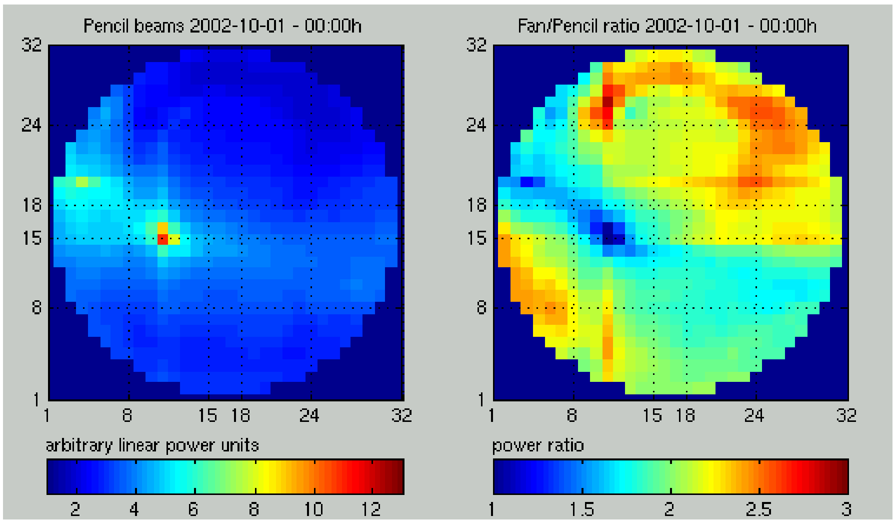
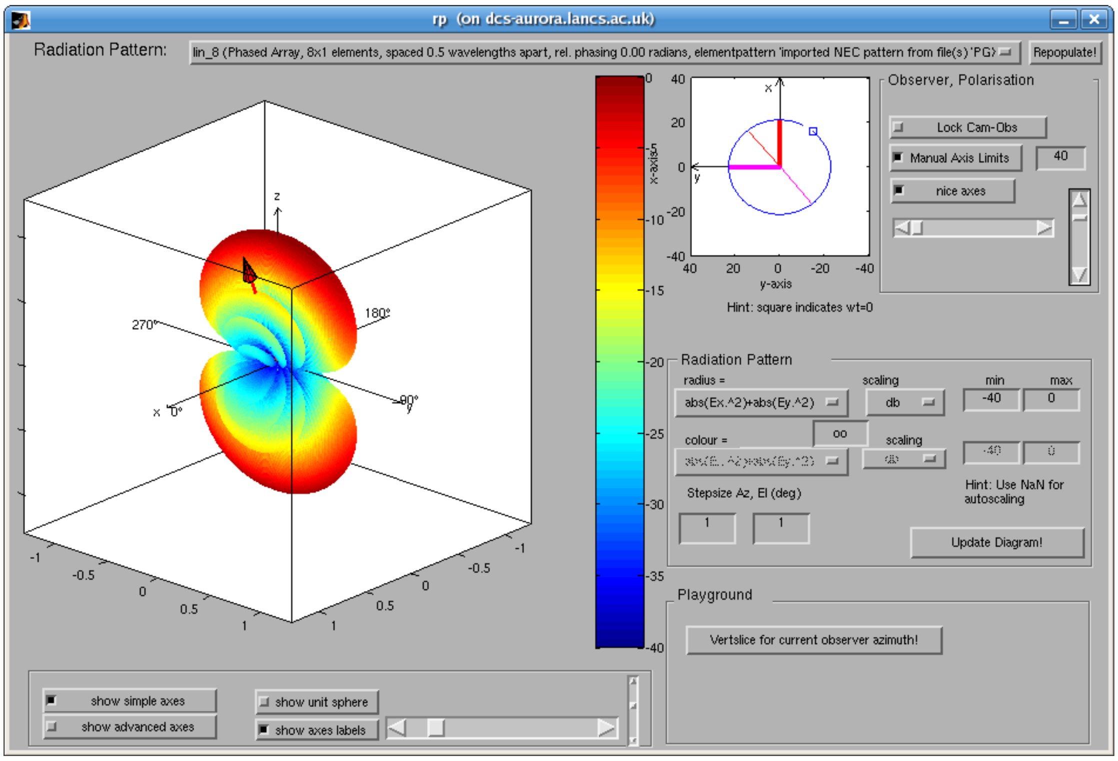


Figure 7.6: One frame of the ARIES worst-case beam determination movie


rp (on dcs-aurora.lancs.ac.uk)

Radiation Pattern: lin_8 (Phased Array, 8x1 elements, spaced 0.5 wavelengths apart, rel. phasing 0.00 radians, elementpattern 'imported NEC pattern from file(s) 'PG›

Manual Axis Limits: 40

Radiation Pattern

radius = abs(Ex.^2)+abs(Ey.^2), scaling db, min -40, max 0

colour = abs(Ex.^2)+abs(Ey.^2), scaling db, min -40, max 0

oo

Stepsize Az, El (deg): 1, 1

Hint: square indicates wt=0

Hint: Use NaN for autoscaling

Figure 7.7: RP, the radiation pattern explorer

both colour and radius axes can be manually specified or set to auto scaling. The step sizes for the plot can be specified, for example to use quick low-resolution plotting initially, and then produce higher-resolution plots once the scaling parameters have been adjusted to the user's liking.

5. There is also a 'playground' section, analogous to the 'sandbox' or 'playground' sections that can commonly be found in Wikis [LC01]. This serves as a placeholder for experimental additional commands, and at the time of this writing contains only one button, 'Vertslice for current observer azimuth.' This button creates a popup window containing a vertical slice through the currently explored radiation pattern at the azimuth position of the polarisation explorer's observer. **RRadPat**'s **vertslice()** plotting function (see section 6.3.2) is used to create this plot.

Whenever the radiation pattern plot is updated, the Radiation Pattern Explorer will print the full **plot3()** command that was used to the MATLAB console. Thus, this command can then be used in other scripts to plot the radiation pattern with exactly the same parameters outside of Radiation Pattern Explorer.

7.7 Scintillation Prediction: scint_calc_mia()

Given a specific radiation pattern (for a specific beam of a specific instrument) as represented by a **RRadPat** object, and a specific radio star as represented by a **CRadioStar** object, we can combine the abilities of those two objects to determine how much power from a radio star is received by the radiation pattern for any given relative position. Now, scintillation effects can predominantly be observed when a radio star passes through highly-sensitive parts of the beam pattern. Being able to calculate when this happens, and to what extent, together with knowledge of the major radio stars that exist, enables us to predict when scintillation can be observed. Furthermore, scintillation in existing data can be identified.

Predicting the presence of scintillation is scientifically important, as scintillation degrades the quality of the data. Sensitive feature detection algorithms might not work reliably for periods of scintillation. The quality of quiet-day curves might be degraded.

Note that in this approach we use our *knowledge* about the radiation pattern of the instrument to determine the influence that a radio star has on the received signal. In section 11.1 we propose a way of *deriving* an a-priori unknown radiation pattern by *measuring* the influence that a particular radio star has on the received signal.

The following sections will firstly describe the MATLAB function **scint_calc_mia()** that was developed by the author to predict scintillation effects for arbitrary MIA instruments. This will be followed by a description of how this scintillation calculator can be integrated into a web environment and therefore made available to external users using a standard web browser. This application of using MATLAB programs as the underlying calculation engine for interactive web pages is potentially useful in a variety of contexts. A similar application, though based around different mechanics, now exists in form of the SPEARS Data Access Facilities. Appendix H contains a description of this existing backend and a comparison between the two approaches, along with some suggestions for future improvements.

scint_calc_mia() calculates when a given radio star enters and leaves which beam(s) of a given MIA instrument. It returns a list of all these 'events' for one specified day.

Syntax

```
events = scint_calc_mia( ['parameter-name', 'parameter-value' [, ...]]  );
```

Input parameters

date Start of 1-day period for which to calculate scintillation effects.

mia_instr A MIA instrument object, scintillation events will be calculated for imaging beams of this instrument, employing the **RMIAPat** adaptor class (see section 6.3.11).

star **CRadioStar** object. This radio star will be used for the calculations.

res **timespan** specifying the resolution of the simulation, defaults to one UT minute.

threshold Threshold in negative dBs when to consider the received signal as relevant for scintillation.

silent Do not print any feedback to `stdout`.

Return value

events An $n \times 3$ cell array containing the n events that were found. events{i,1} is the date when this event takes place in the format YYYY-MM-DD HH:MM:SS (UT), events{i,2} is 1 if the star enters the beam or 2 if the star leaves the beam and events{i,3} is the beam number.

Figure 7.8 shows the output of **scint_calc_mia()** for 20 January 2001 for the IRIS instrument. Compare this to the radio star traces and recorded data for the same day, for example in figures 7.2 and 7.3.

Description

Internally, **scint_calc_mia()** queries **mia_instrument**'s **info()** function for the beam numbers of all imaging beams of this instrument. It then creates a RIOSIM **RMIAPat** object for each of these beams. Then the location of the radio star is calculated for all times during the specified day. This is done at a time resolution as specified with the **res** parameter, which defaults to one UT minute. Since the power flux of the radio star is known, the power received by each beam at each moment in time can be calculated by passing all the calculated radio star locations to the **RRadPat:getGain()** function.

The results are then normalised to their maximum value and scaled to dBs. An edge detection algorithm is now used to find all the 'enter beam' and 'leave beam' events. The list of events is sorted by time and returned.

7.8 Running the Scintillation Calculator Remotely and Asynchronously

Tools like the scintillation calculator from section 7.7 above are ideal candidates for remote access through a web browser. That way, potential users will not require detailed knowledge about any underlying (MIA, RIOSIM, etc.) toolkits. Neither would they require user accounts on the computer that is performing the calculation. The following sections describe how to turn **scint_calc_mia()** into a remotely accessible web-based tool.

We will use generic methods that will work not only for **scint_calc_mia()**, but for any MATLAB function, using **scint_calc_mia()** merely as an example. Figure 7.9 is a sequence of screenshots of the end result: A scintillation calculator for all instruments known to MIA, accessible through the web.

Note that although the chosen approach is generic in nature, it has some serious drawbacks, and we will draw a short conclusion in section 7.8.5 below. More advanced remote invocation

```
>> events = scint_calc_mia('date',timestamp([2001 01 20 0 0
                                                    0]))
Using radiostar      : Radio Star "Cassiopeia A (3C 461)"
Using date           : 00:00:00 20 January 2001.
Using instrument     : Kilpisjarvi [IRIS] (KIL #1).
Using time resolution: 1 m (UT).
Using threshold      : -6dB.
Found 49 imaging beams.
Receiving from beams.....................................
edge detection in progress......................................
events =
'2001-01-20 01:04:00'    [2]    [ 3]
'2001-01-20 01:12:00'    [1]    [ 4]
'2001-01-20 02:55:00'    [2]    [ 4]
'2001-01-20 03:03:00'    [1]    [ 5]
'2001-01-20 04:53:00'    [2]    [ 5]
'2001-01-20 05:31:00'    [1]    [13]
'2001-01-20 07:20:00'    [1]    [20]
'2001-01-20 07:40:00'    [2]    [13]
'2001-01-20 09:16:00'    [1]    [27]
'2001-01-20 09:45:00'    [2]    [20]
'2001-01-20 10:34:00'    [1]    [26]
'2001-01-20 11:14:00'    [2]    [27]
'2001-01-20 11:44:00'    [1]    [33]
'2001-01-20 12:43:00'    [2]    [26]
'2001-01-20 12:58:00'    [1]    [32]
'2001-01-20 13:16:00'    [2]    [33]
'2001-01-20 14:47:00'    [1]    [31]
'2001-01-20 15:05:00'    [2]    [32]
'2001-01-20 15:20:00'    [1]    [24]
'2001-01-20 16:19:00'    [2]    [31]
'2001-01-20 16:49:00'    [1]    [23]
'2001-01-20 17:29:00'    [2]    [24]
'2001-01-20 18:19:00'    [1]    [16]
'2001-01-20 18:47:00'    [2]    [23]
'2001-01-20 20:24:00'    [1]    [ 9]
'2001-01-20 20:43:00'    [2]    [16]
'2001-01-20 22:32:00'    [2]    [ 9]
'2001-01-20 23:10:00'    [1]    [ 3]
>>
```

Figure 7.8: Output of **scint_calc_mia()** for 20 January 2001

and job queuing systems exist, one of which is the custom framework used by MIA. Appendix H contains a brief description and evaluation of the MIA approach. Commercial frameworks are also available, for example the MATLAB Distributed Processing Toolbox [Mata].

7.8.1 XML Wrapper for scint_calc_mia(): SCINT_CALC_MIA_XML_WRAPPER.M

To make **scint_calc_mia()** accessible from outside MATLAB, some standard way of passing parameters to and returning results from this function needs to be implemented that is independent of MATLAB's internal way of parameter passing. This is because MATLAB's own way only works from within the MATLAB environment, whereas we want to call **scint_calc_mia()** directly from the 'outside world.'

The Extensible Markup Language (XML) has evolved as a standard for describing arbitrary data together with its metadata (data describing the data) [BPSM$^+$06]. Well-written XML files can easily be understood by both humans and computers alike, so XML seems an obvious choice for data exchange between **scint_calc_mia()** and the outside world.

A wrapper function **scint_calc_mia_xml_wrapper()** has therefore been implemented. This function takes only two parameters: The name of an XML file containing the input parameters and the name of an XML file that will take the results. **scint_calc_mia_xml_wrapper()** will parse the input XML file, translate the parameters contained in this XML file into the name-value pairs required by MATLAB and pass them on to **scint_calc_mia()**. Once **scint_calc_mia()** returns, **scint_calc_mia_xml_wrapper()** will take the results (in this particular case in the MATLAB specific format of a cell array) and translate them back into XML. This XML data will get written to the output file as specified by the second parameter.

Note that the functionality of marshalling and unmarshalling parameters could potentially be integrated into the original **scint_calc_mia()** function. However, the aim of this section is to show how to turn existing MATLAB functions into remotely invokable web-based services *without* modifying the existing code.

7.8.2 Remote-Access Wrapper for SCINT_CALC_MIA_XML_WRAPPER.M: scint_calc_mia_xml_wrapper.sh

While **scint_calc_mia_xml_wrapper()** is no longer dependent on MATLAB's way of passing parameters and results between functions, it still relies on access to the local file system since, as described above, the parameters need to be written to an XML file which is then read by **scint_calc_mia_xml_wrapper**(). Similarly, if some external process wants to read the results, it has to be able to access the XML file that was written by **scint_calc_mia_xml_wrapper**().

A simple and secure way of executing processes remotely without having to rely on any additional frameworks such as CORBA [Obj05], Gridservices (e.g. [Fos06], [Uni07]) or a home-made framework such as the one used in MIA, is to use the Secure Shell (SSH) protocol [BSB05]. However, SSH does not give direct access to the remote filesystem. Instead, SSH will execute a given command on a remote machine, redirecting its standard input (`stdin`) and standard output (`stdout`) to the local machine. It is through these two 'tunnels' that data can be passed between the local and the remote machine.

A secondary wrapper in the form of a BASH [New05] shell script has therefore been developed. This wrapper takes arbitrary input through `stdin` (the standard input as tunnelled from the invoking machine by the SSH protocol) and transfers this data to a temporary file. It then calls the MATLAB **scint_calc_mia_xml_wrapper()** function with the name of that file and the name of another (so far empty) temporary file. **scint_calc_mia_xml_wrapper()** will now read its input parameters from the temporary file which has in fact been transferred from the remote machine through the '`stdin`-tunnel.' It will write the results to the second temporary file as described in section 7.8.1 above. The BASH wrapper will then take the contents of this secondary file (the results) and pass them through its '`stdout`-tunnel' back to the remote machine. The whole process of creating temporary files is completely transparent to the user of the remote function call. All the user needs to do is to pass the input parameters in XML format to SSH through `stdin`, and to process the XML output as it appears on `stdout`.

7.8.3 Asynchronous Remote Execution: run_scint_calc

In theory, the wrapper shell script described in section 7.8.2 could be invoked directly by the webserver in response to a user request. However, it might take anywhere from a few seconds up to several minutes to process the request, depending on the complexity of the task. Most browsers time out after about 30 seconds, meaning that they would never get to see the results of such a function call. There are also security issues, since CGI programs run with the effective user ID of the webserver. This user ID would have to be granted remote execution rights for the given remote machine. This would in turn mean that every CGI program would from then on have remote execution rights on that machine.

These considerations led to the development of yet another wrapper mechanism, this time on the webserver, to avoid these drawbacks. It aims to achieve asynchronous execution and fine-grained execution permissions. Since this wrapper will be run asynchronously, it also needs some mechanism of indicating when the remote processing has finished. A convenient way of doing this is through temporary files.

The wrapper that was developed is called **run_scint_calc**. This shell script takes two parameters, the first one specifying the file name of the temporary file to which the results of the remote execution should be written. The second parameter specifies the name of the file containing the input parameters. This file also serves the aforementioned secondary purpose: the content of this file will be deleted by **run_scint_calc** to indicate that the remote processing has finished and that the specified output file now contains valid data. This provides a way for the webserver calling **run_scint_calc** to check whether remote execution has finished or not.

A final layer of indirection has been added to support fine-grained execution permissions. Instead of calling **run_scint_calc** directly, the web server will call a **suid_wrapper** which in turn will call **run_scint_calc**. Therefore, **suid_wrapper** is the only executable ever called directly by the webserver. Through the suid mechanism, it will execute **run_scint_calc** under the effective user ID of the owner of **suid_wrapper**. Therefore, only this user account needs to be granted remote execution rights.

7.8.4 Summary: How to Asynchronously Invoke a MATLAB Function on a Remote Machine from a Webserver

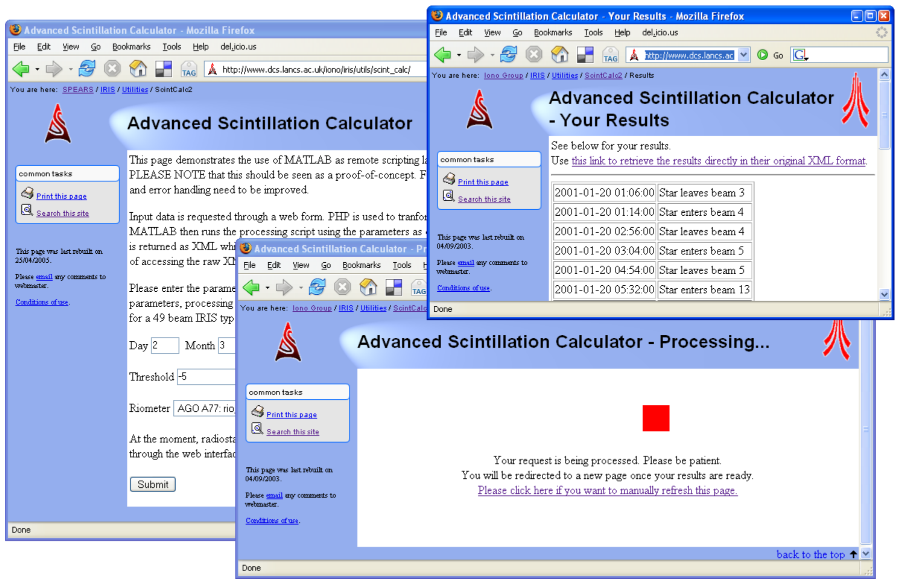
The UML sequence diagrams in figures 7.10 and 7.11 summarise the sequence of events described in the preceding sections. Figure 7.10 shows the client (user and webserver) side[1], figure 7.11 shows the activities on the server (MATLAB) side. The following is a description of events in chronological order:

Client side (figure 7.10)

1. User requests the scintillation calculator tool through his web browser. The web server returns the start page.

2. User submits a request.

[1]Note that we refer to both the end user and the webserver as being on the client side in this description, although these two entities can of course be located on two physically separate remote nodes.


Advanced Scintillation Calculator - Mozilla Firefox

http://www.dcs.lancs.ac.uk/iono/iris/utils/scint_calc/

You are here: SPEARS / IRIS / Utilities / ScintCalc2

Advanced Scintillation Calculator

This page demonstrates the use of MATLAB as remote scripting la
PLEASE NOTE that this should be seen as a proof-of-concept. F
and error handling need to be improved.

Input data is requested through a web form. PHP is used to tranfor
MATLAB then runs the processing script using the parameters as
is returned as XML whi
of accessing the raw XI

Please enter the parame
parameters, processing
for a 49 beam IRIS typ

Day 2 Month 3

Threshold -5

Riometer AGO A77: rio

At the moment, radiosta
through the web interfac

Submit

This page was last rebuilt on 25/04/2005.

Please email any comments to webmaster.

Conditions of use.

Done

Advanced Scintillation Calculator - Pr

You are here: Iono Group / IRIS / Utilities / ScintCalc

Advanced Scintillation Calculator - Processing...

Your request is being processed. Please be patient.
You will be redirected to a new page once your results are ready.
Please click here if you want to manually refresh this page.

This page was last rebuilt on 04/09/2003.

Please email any comments to webmaster.

Conditions of use.

back to the top

Done

Advanced Scintillation Calculator - Your Results - Mozilla Firefox

http://www.dcs.lancs.ac

You are here: Iono Group / IRIS / Utilities / ScintCalc2 / Results

Advanced Scintillation Calculator - Your Results

See below for your results.
Use this link to retrieve the results directly in their original XML format.

2001-01-20 01:06:00	Star leaves beam 3
2001-01-20 01:14:00	Star enters beam 4
2001-01-20 02:56:00	Star leaves beam 4
2001-01-20 03:04:00	Star enters beam 5
2001-01-20 04:54:00	Star leaves beam 5
2001-01-20 05:32:00	Star enters beam 13

This page was last rebuilt on 04/09/2003.

Please email any comments to webmaster.

Conditions of use.

Done

Figure 7.9: Three screenshots of the scintillation calculator as accessed through its web interface

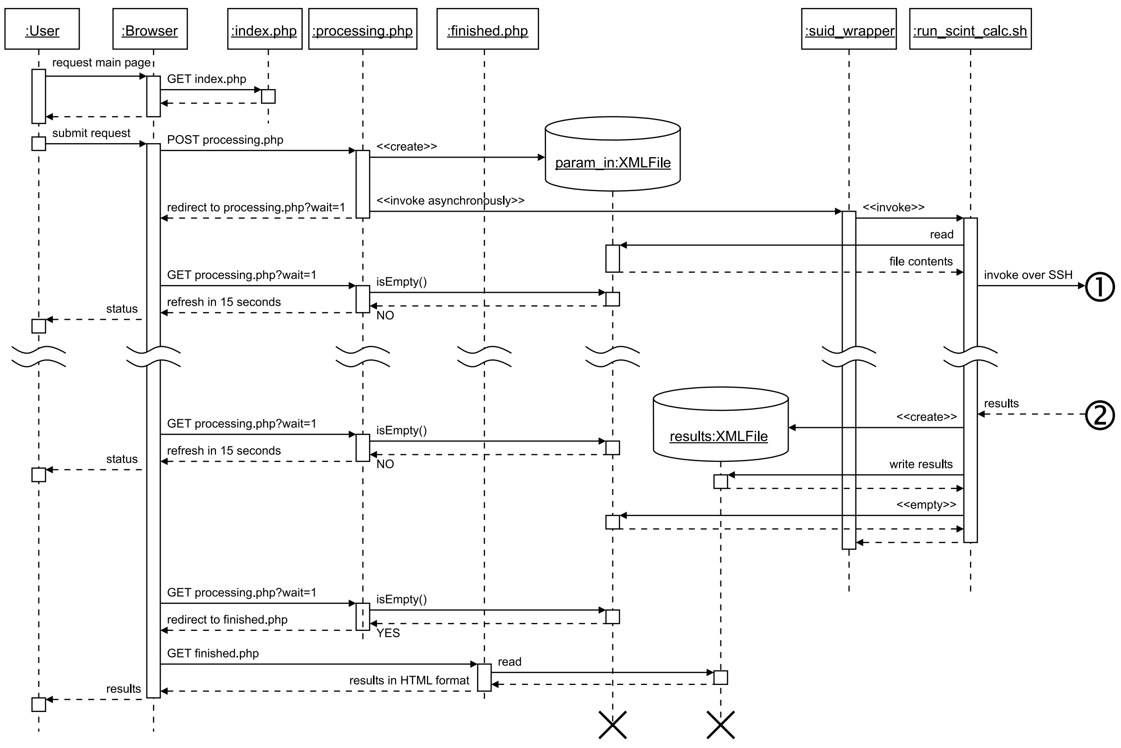


Figure 7.10: Remote execution the hard way. Client side: From user request to remote application invocation through SSH. For a description of this UML sequence diagram see section 7.8.4.

3. Browser passes request parameters to the web server, which in turn passes them on to `processing.php`.
4. `processing.php` writes the parameters to a temporary file “`param_in`” and runs the **suid_wrapper**.
5. **suid_wrapper** runs **run_scint_calc** with required user privileges.
6. **run_scint_calc** reads the processing parameters from the “`param_in`” file and passes them through the SSH tunnel to the remote machine, while at the same time invoking the **run_scint_calc_wrapper** script on the remote machine.
7. The remote machine does the processing (see figure 7.11 and description below) and returns the results through the `stdout` SSH tunnel.
8. **run_scint_calc** reads the results from the `stdout` SSH tunnel and writes them to a temporary file “`results`”.
9. **run_scint_calc** signals that processing has finished by setting the file size of the “`param_in`” file to 0.
10. The periodically invoked `processing.php` script on the web server realises that processing has finished and redirects the user’s browser to the “finished” web page.
11. The browser requests the “finished” web page, causing the `finished.php` script to be executed.
12. `finished.php` reads the “`results`” file and returns the result in HTML format.

Server side (figure 7.11)

1. **scint_calc_xml_wrapper.sh** gets invoked from the SSH daemon.
2. **scint_calc_xml_wrapper.sh** reads all XML input parameters from `stdin` and writes them to a temporary file.
3. **scint_calc_xml_wrapper.sh** runs **scint_calc_mia_xml_wrapper()** inside MATLAB.
4. **scint_calc_mia_xml_wrapper()** reads the input parameters from the XML file and invokes the original **scint_calc_mia()** function.

5. When **scint_calc_mia()** returns, **scint_calc_mia_xml_wrapper()** writes the results to a temporary file “`output`” and terminates.
6. Control goes back to **scint_calc_xml_wrapper.sh** which reads the “`output`” file and passes it back to the client through its `stdout` SSH tunnel.

7.8.5 Conclusion (Remote Asynchronous Execution)

The framework presented in the above subsections does achieve asynchronous remote execution of MATLAB code triggered by a remote user through a web-based interface. It does not require any modifications to the MATLAB function that is to be invoked.

However, the presented solution also has several drawbacks, namely:

- Complex interactions, as illustrated by the sequence diagrams. This makes maintenance difficult and error-prone.
- No queuing for multiple requests (instead, all requests will try to execute simultaneously)
- No load limiting (even when the server is busy, it will still accept new requests, therefore slowing down all currently running processes)

There are a number of potential solutions to these issues:

- Decoupling of client and server through a remotely accessible queuing system, e.g. a database. This approach is taken by MIA, see appendix H.
- MATLAB Distributed Processing toolbox [Mata]. MATLAB itself has support for remote invocation. This solution is expensive in terms of licensing costs.
- Other remote object frameworks (Webservices, Gridservices). These are also complex but scale well to complex problems.

In conclusion, the presented framework helps to understand all issues related to remote invocation of existing MATLAB tools. It does perform well for this particular example. For large-scale and more robust applications, a less hand-made solution is preferable.

7.9 Summary

Several advanced applications of the RIOSIM riometer simulation toolkit were presented. These take the basic functionality as provided by the various components of RIOSIM and combine them with other RIOSIM components and/or external tools such as MIA or the mapping toolbox M_Map to derive new data products. Many of these data products have been used in the development, deployment and testing of the ARIES riometer.

A ‘do it yourself’ way of making such data products available through the World Wide Web (WWW) was also discussed, and the limitations of such a hand-made asynchronous execution engine were pointed out.

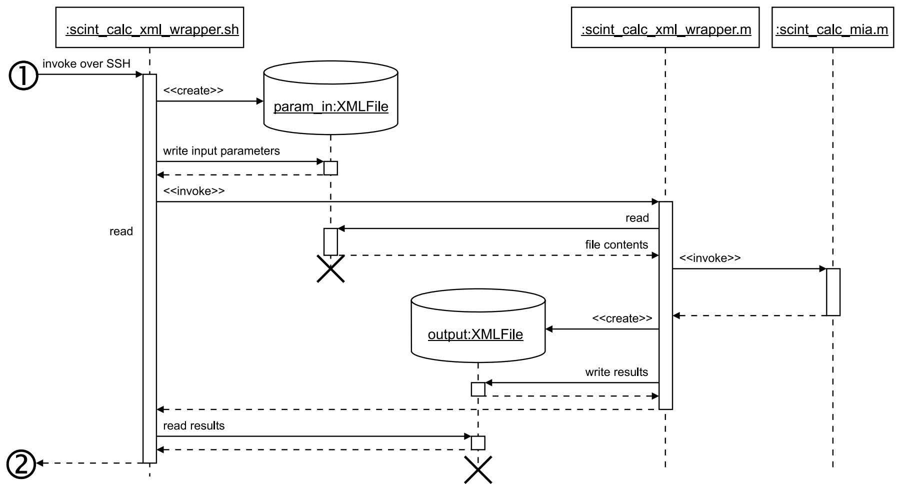


Figure 7.11: Remote execution the hard way. Server side: From remote invocation to actual execution of the original MATLAB function.

Chapter 8

Advanced Riometer Components: ARCOM

This chapter defines a framework for a generic operating software for advanced riometer systems. It is roughly structured along the process activities of the software engineering cycle [IEE07]. We will start off by stating the basic requirements, and then show how the ARCOM (Advanced Riometer COMponents) architecture implements these requirements. This chapter will not go into implementation details of every single function call. Instead it aims at providing a general — although more abstract — description of the working principles involved. Detailed information is contained in the ARCOM documentation [Gri06a], which is automatically generated and updated from the appropriately documented source code by Doxygen, a source code documentation generator tool [vH06].

As most technical systems, the ARIES advanced riometer, which prompted the development of the ARCOM framework, consists of a number of hardware and software parts. The system design of ARIES exhibits many characteristics of a 'Wicked System' [Som04], in that the design parameters and requirements are not well-defined at the outset, and are likely to change as the system evolves from prototype to prototype and results from test campaigns feed back into the system design.

The design of operating software for such a system presents unique challenges to the software engineer seeking to support system evolution to the most flexible extent possible. This chapter will look at what these challenges are, how they influenced the design of the ARIES operating software, and how the implemented software architecture solves the 'Wicked Problem.' The ARCOM operating software is the result of a structured approach of defining goals, deriving the overall architecture and finally designing, implementing, integrating, deploying and testing the individual (software) building blocks that make up the overall system.

The software engineering process revolves around the basic activities of specification, design, implementation and testing (verification and validation) [Som04]. For complex systems, an iterative (evolutionary) approach is usually taken in order to counteract some of the uncertainties inherent to 'Wicked Systems.' This chapter will present the initially defined goals of the software and how the developed ARCOM software meets these goals. Major design decisions and, where appropriate, their superiority over alternative techniques will be discussed. Concepts relating to the discipline of Software Engineering will be introduced and illustrated as necessary.

8.1 Design Goals

Due to the 'wicked' nature of the problem, the initial specification stage of the software design process produced not so much requirements but goals, i.e. very general statements describing what the software should and should not do. What distinguishes goals from requirements is that they are sufficiently vague so that they will not unnecessarily narrow down the number of design choices. Goals are not objectively verifiable ('testable') as they do not contain enough detail to do so [Som04]. This is in line with the fact that at the initial design stage, those details were not known, and the software to be developed would itself be used to elicit likely operating requirements.

This section lists the basic requirements (goals) that the ARIES control software needs to take into account. The following sections will go into details of how the implemented architecture ARCOM (Advanced Riometer COMponents) fulfils these requirements.

Speed

On average, the system needs to keep up with the incoming data stream from the receivers. The phrasing 'on average' is appropriate, because even though we must not lose any data, we need not guarantee that all data is processed immediately. In engineering terms, this system can be referred to as a 'soft real-time' system. Several tens of megabytes will have to be processed per second when dealing with raw data streams. A system with 64 individual receiver channels, each sampled at, say, 1MHz and 16bit complex sampling, will have a raw data rate of $64 \times 2 \times 16\text{bit} \times 1\text{MHz} = 2\text{GBit/s}$!

Run-time Reconfigurability

To a certain extent, we want to be able to reconfigure the system while it is running. For example we want to continuously integrate an incoming signal, and then later on add functionality for logging the incoming signal to a file. In this example, the logging should not affect the operation of the integrator.

Expandability

We want to be able to develop new system functionality in the future. This new functionality should integrate seamlessly with existing parts of the system.

On-line Status Information

We want to be able to query the current internal status of the system in sufficient detail at any time, and without affecting the operation of the system.

Remote Control

Due to the physical remoteness of the site, all system functionality should be accessible from off-site and through potentially slow communication links.

8.2 Basic ARCOM Structure

Figure 8.1 gives a layer-oriented overview of how ARCOM fits in with the other parts of an advanced riometer system. This section discusses major aspects of the ARCOM architecture (structure) and how these aspects contribute to fulfilling the design goals specified in section 8.1.

8.2.1 Component-based

One of the basic decisions that needs to be made during the very early stages of software design is whether to implement the desired functionality in one monolithic block, or whether to split the design into components, see figure 8.2. All but the most basic pieces of software will exhibit some trace of component-based design. Yet even software designed around a component-like structure can still result in a monolithic executable, with all pieces of the software being glued together at compile-time. Based on the basic goals formulated in section 8.1, it follows naturally that an advanced component-based design approach needs to be taken. Such an approach will naturally allow for easy insertion and removal of components at run-time.

The decision towards a component-based design is a major step towards designing the overall software structure. This decision alone is sufficient for the more detailed design outlined in the following sections. However, for lack of a better location, some more specific implementation details will be outlined here. As William Grosso points out in [Gro02], it is hardly ever necessary to write one's own componentry framework. In fact, there are compelling reasons not to do so: Large parts of such a framework consist of code that deals with communication between components. There is usually little point in re-inventing the wheel, and code that tries do do so is likely to contain more errors than well-established existing frameworks. Excellent frameworks exist, and table 8.1 shows three exemplary ones that are commonly used for applications such as this one. The decision for CORBA was made mainly because it is well-supported on our development platform of choice (Linux) and bindings for our implementation language (C++) are readily available. However, it should be clear that the actual framework used has little impact on the design. Also, the implementation described in this chapter makes use of only a small portion of the functionality offered by the CORBA framework (figure 8.3), abusing it as merely a way of simple (strictly client-server-type) inter-object communication and not using any of the higher level CORBA services that provide common business logic such as transaction processing, security management, asynchronous notification, etc.

8.2.2 Pipeline Architecture

From a technical point of view, it soon becomes clear that the basic functional model of any operating software for instruments such as ARIES is that of pipelining. A simple example of a pipelining architecture is shown in figure 8.4. A pipeline architecture follows the natural notion of data flowing through the system, being processed as it does so. Data enters the system, gets processed in various ways, and finally leaves the system. A pipelining architecture supports reuse of transformations, evolving the system by means of adding new transformations is straightforward and concurrent systems (many processing paths processed simultaneously) are readily supported. This model is obviously not limited to one input and one output, a fact that is of major importance for the ARCOM architecture. As will be seen further on, several streams of data may enter the system simultaneously, to be processed independently or combined together. Results may be written to external disk straight away, or processed by components further down

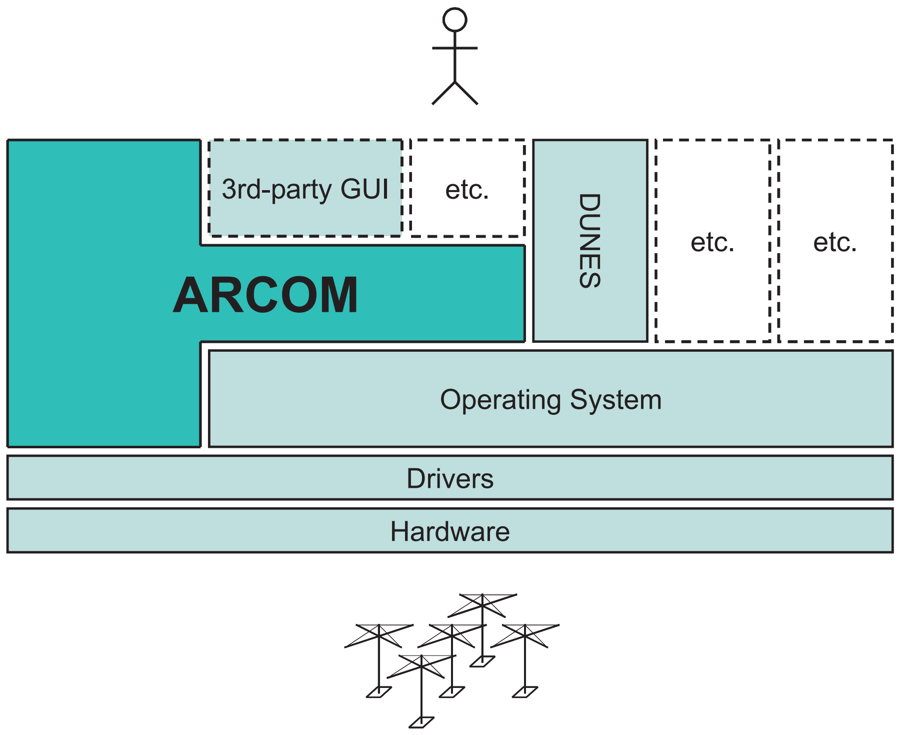


Figure 8.1: Multi-layer view of ARCOM and its operating environment. DUNES is an independent dial-up networking module implemented by the author and is not discussed further in this chapter. Appendix I contains an overview of the functionality provided by DUNES. For more details see the DUNES requirements document [Gri06b].

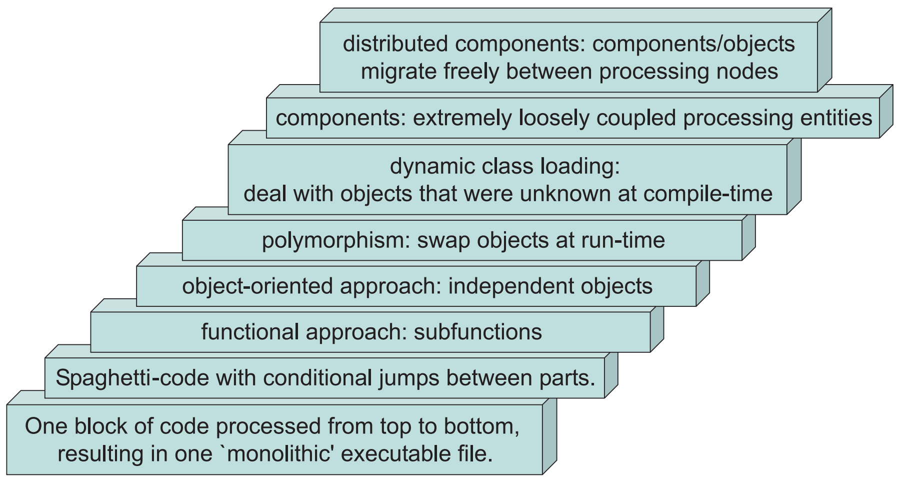

distributed components: components/objects migrate freely between processing nodes
components: extremely loosely coupled processing entities
dynamic class loading: deal with objects that were unknown at compile-time
polymorphism: swap objects at run-time
object-oriented approach: independent objects
functional approach: subfunctions
Spaghetti-code with conditional jumps between parts.
One block of code processed from top to bottom, resulting in one `monolithic' executable file.

Figure 8.2: From monolithic spaghetti-code to fully component-based distributed applications. An attempt was made to order common design practices according to how ‘component-based’ they are. Although this is in many senses comparing apples and oranges, this figure attempts to give the reader a general idea as to which techniques are employed in which context and how far up they are on the ladder of today’s programming concepts.

	Java RMI (Remote Method Invocation)	COM (Component Object Model)	CORBA (Common Object Request Broker Architecture)
scope	Specific to Java programming language.	In practice specific to Microsoft Windows operating systems.	Designed to be system-independent.
support for distributed systems	Yes (requires Java)	Yes (DCOM)	Yes (Inter-ORB communication)
cost	Free with Java Software Development Kit	Free to use, various levels of development suites at different costs.	Many free implementations and language bindings readily available for many different operating systems.

Table 8.1: Comparison of three common componentry frameworks

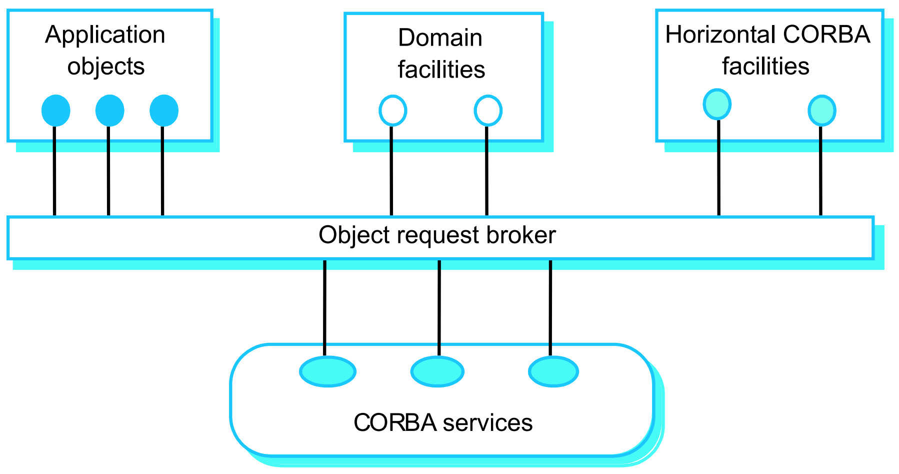
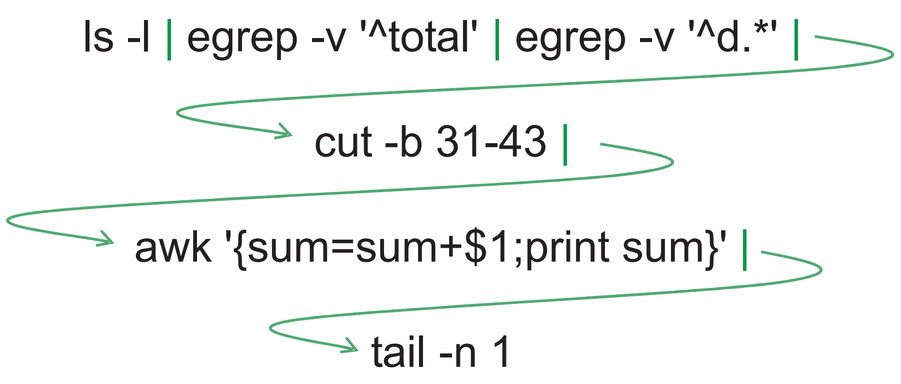


Figure 8.3: Structure of a CORBA-based application, from [Som04]. ARCOM uses CORBA only for client-server-style communication between application objects (top left).



```
ls -l | egrep -v '^total' | egrep -v '^d.*' |
cut -b 31-43 |
awk '{sum=sum+$1;print sum}' |
tail -n 1
```

Figure 8.4: An example of pipelining: Pipes on the UNIX command line. This example calculates the total size of all files in the current directory, starting off by outputting a list of all files and their sizes with “ls” and using various other standard UNIX tools along the pipeline.

the pipeline. A pipelining architecture inherently offers the following advantages (taken with modifications from [Som04]), all of which are desirable for ARCOM and tie in with the goals specified in section 8.1:

- It supports reuse of transformations.
- It is intuitive to think of the work done by the software in terms of input and output processing.
- Evolving the system by means of adding new transformations is straightforward.
- Simple to implement as concurrent system (many processing paths processed simultaneously).

The major disadvantage of a pipelining model is that each transformation needs to agree on a common input and output format in order to be able to tie in with the other transformations along the processing pipeline. Section 8.6 below describes how this issue is overcome and in fact turned into an advantage in ARCOM through the use of a versatile streaming data format.

Following on from the considerations above, and keeping in mind the goals as defined in 8.1, a set of three principal (meta-)components was designed (see figure 8.5). Every component in the ARCOM software behaves like (is derived from) one of these principal components. The core ARIES control software is therefore made up of only three structurally different components, and even those have strong commonalities as far as inter-component communication is concerned. We will first describe the three meta-components, and how real-life instantiations of these components can interact in sections 8.2.4 onwards below. The discussion of specific components as implemented in ARCOM is left for the later section 8.8 below. During the discussion, it will become clear how the structure presented addresses each of the requirements in section 8.1. The three basic components are merely templates for real components that will be derived from the basic components. Section 8.8 will describe the real components that have so far been implemented.

8.2.3 High-speed Component Interconnect

In addition to the CORBA interfaces, which are being used for infrequent control tasks such as starting and stopping components, all components that can take part in pipeline-based processing are glued together through blocks of shared memory with strictly unidirectional data flow.

Establishing a shared memory interface for data flow between components in the processing pipeline allows for maximum speed as data is transferred between components, essentially only limited by memory throughput of the processing hardware. All these shared memory blocks share common characteristics, and these will be described in section 8.3 after having introduced the basic ARCOM components. See figure 8.6 for an example of how several components work together, this was in fact the software configuration for the October 2002 experiment (chapter 9). Figure 8.7 represents a more recent configuration for the current FPGA-based system design (see chapter 3, section 3.3 and appendix D).

8.2.4 Recorders

Recorders stand at the beginning of a processing chain (pipeline). Their responsibility is to collect data from some (hardware) device and transfer it to the standardised shared memory interface. Therefore, a recorder component needs detailed knowledge about the particular hardware device it is designed to get data from. This could be an A/D converter, a GPS clock, environmental sensors for measurements such as temperature and humidity, a camera, etc. The recorder communicates with this device through a specialised Application Programming Interface (API) specific to the respective hardware device. The API function calls normally get translated into commands to an underlying device driver, and this driver will communicate directly with the hardware device. Driver and API are usually third-party modules supplied with the respective hardware.

A recorder component will typically only take input data from one specific (type of) hardware device. Where data from several devices is required, multiple recorders will be used. This helps to keep each recorder component highly cohesive (doing one task, but properly) and loosely coupled (not depending on multiple other pieces of software), both very desirous properties for software components [Dea05].

A recorder also knows how to access an ARCOM shared memory interface. Therefore, it can store data coming from the hardware into the shared memory interface, ready to be picked up by (processor) components further down the pipeline.

Finally, like all ARCOM components, a recorder component understands the common ARCOM CORBA commands as defined in the **AComponent** interface. See section 8.7 for more information on the common CORBA interface.

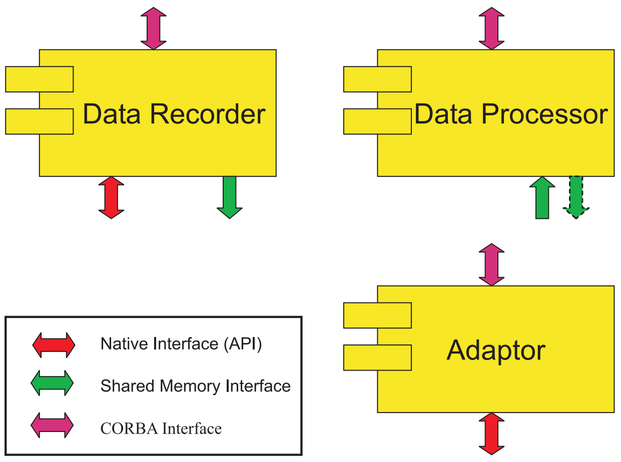


Figure 8.5: The three basic ARCOM (meta-)components

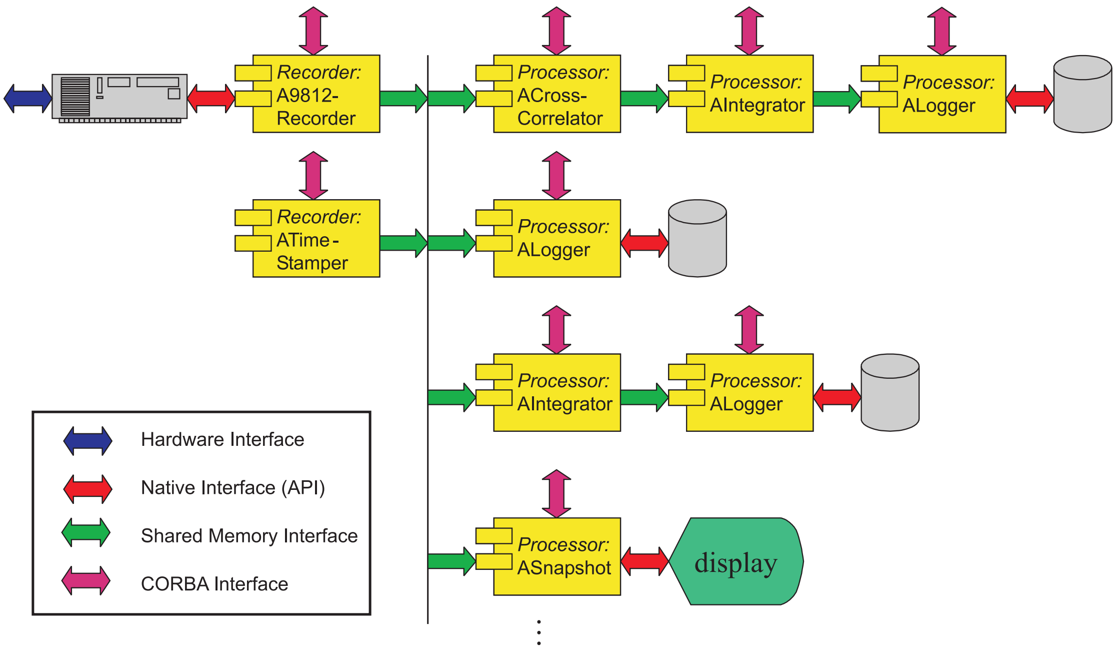


Figure 8.6: ARCOM example configuration as used during initial experiments. One **ALogger** component logs all packets passing through the main shared memory interface, other **ALogger** components only log the output of preceding processing stages. Additional components can be added and removed at run-time. Command line tools or additional components can be used to tap into any of the shared memory interfaces.

8.2.5 Processors

Processor components are responsible for processing data. They get this data from a shared memory interface (it will have been put there by other recorder or processor components further up the processing pipeline). Therefore, they do not have to know anything about any specific hardware, i.e. they do not need to know how the data got into the shared memory in the first place. This distribution of tasks between recorder components and processor components keeps both types of components well-focused (cohesive). Also, timing issues are no longer as critical as in the case of a recorder component, because even though the processor component will have to keep up with the incoming data flow on average (see our initial goals in section 8.1), the shared memory interface inherently has the ability to transparently buffer data, thus decoupling recorders and processors in the time domain. Section 8.3 gives more information about the shared memory interface.

8.2.6 Adaptors

The purpose of an adaptor component is to provide a consistent CORBA interface to different hard- and software components supplied by third parties. In this respect, an adaptor follows the ‘façade’ design pattern [GHJV95, Dea02]. Any given adaptor component introduces an additional (software) layer between the original (ARCOM incompatible) interface of the given third-party software component and the rest of the ARCOM components. That way, third-party components like, for example, the control software for an uninterruptible power supply (UPS), can be accessed through a standard CORBA interface, just like any other ARCOM component. In particular, they can be initialised with the **init()** command and their current status can be retrieved with the **getStatus()** command (see section 8.7). Adaptors do not access shared memory interfaces.

8.3 Shared Memory Interface

Figure 8.8, panel (a), shows a basic data flow example from the hardware to the final result, in this case a file on a hard disk. Data enters the system through the hardware device, shown as the data acquisition card on the left. An ARCOM recorder component, **A9812Recorder**, reads the data from the hardware by means of proprietary API function calls to the driver software that comes with the respective hardware. This data is then encapsulated into ARCOM streaming

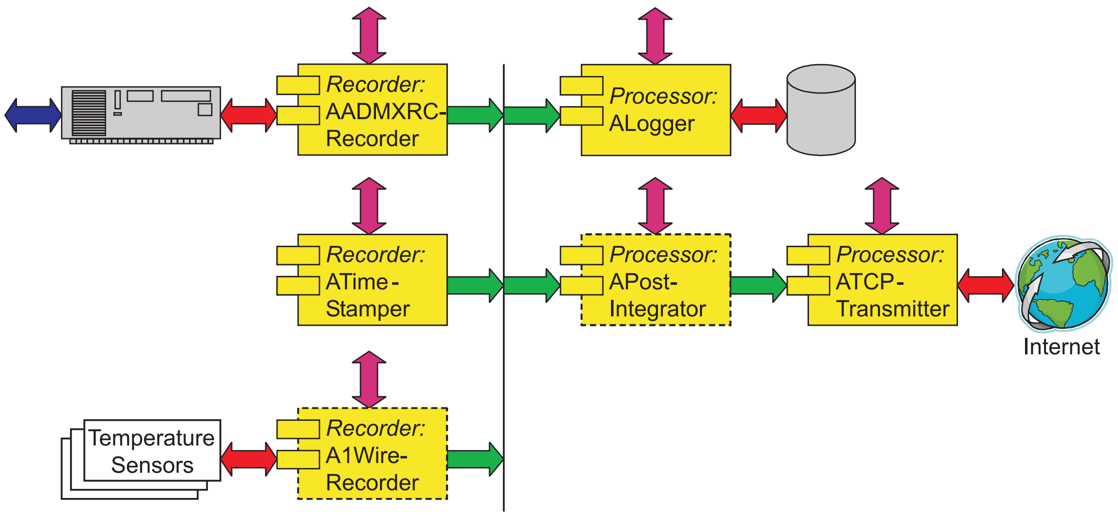
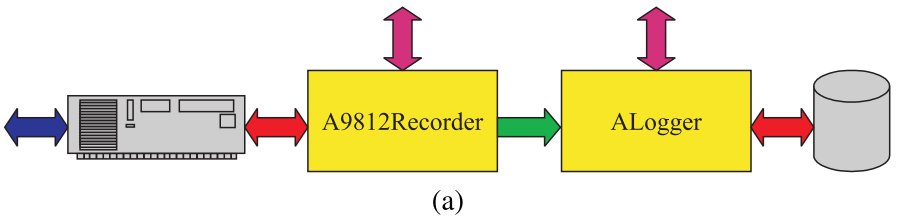
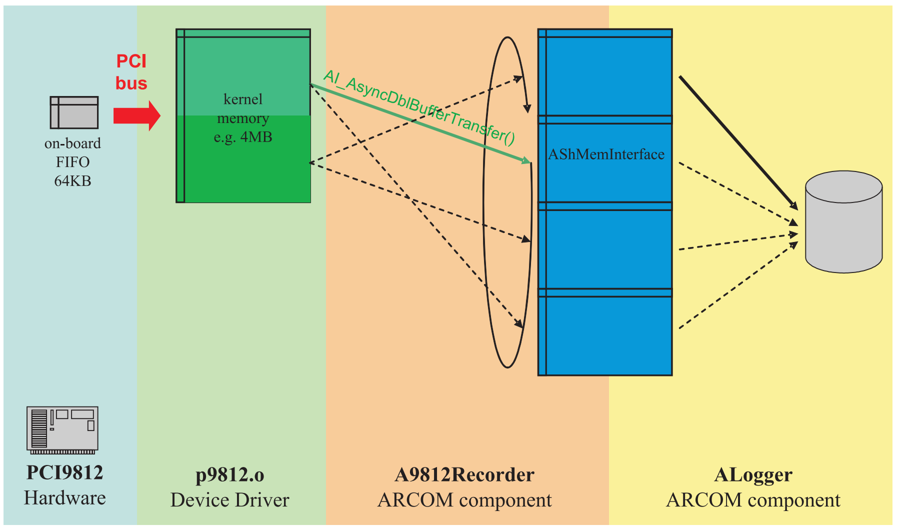


Figure 8.7: ARCOM example configuration for current FPGA-based design. Dashed components are still under development. All recorder components feed into the same shared memory interface. The resulting data stream gets stored to disk by the **ALogger** component. Selected packets can be post-integrated and transmitted over the network in real-time. As in the previous example, additional components can be added and removed at run-time and command line tools or additional components can be used to tap into any of the shared memory interfaces.





(b)

Figure 8.8: Example of ARCOM data flow through a shared memory interface. (a) Component configuration, (b) Internal data flow

packets and output to an ARCOM shared memory interface.

An ARCOM processor component, **ALogger**, connects to the output side of the shared memory interface. The component retrieves data as it becomes available and, in this example, simply stores it to disk.

Panel (b) shows how the overview diagram translates into more detailed internal processes. The exact data flow shown in this panel is only valid for this particular combination of components, whereas the more generic description in panel (a) can readily be used to describe a variety of different component combinations. In the case shown, data flows from a hardware FIFO (First-In, First-Out buffer) on board the data acquisition card to some reserved memory area within the random access memory (RAM) of the processing computer. It is then the responsibility of the **A9812Recorder** component to continuously cause the device driver to transfer data from this internal buffer (that is inaccessible to all other programs) to the shared memory interface.

The **ALogger** component finally reads data from the shared memory interface and writes it to a file. It is important to understand that the **ALogger** component itself does not need to have any knowledge whatsoever about how the data got into the shared memory interface. This means that the **ALogger** component can be used to log data to files no matter where the data originates from, or what information it contains.

Whilst we will leave the detailed description of how to use ARCOM shared memory interfaces to the ARCOM software documentation [Gri06a], we will describe the novel concepts of this interface compared to simple blocks of shared memory [Ste98] here:

8.3.1 Multi-client

In all but the simplest pipeline configurations, the same stream of data will have to flow from one source (recorder) to several destination (processor) components simultaneously. For example, we might want to store raw incoming data to a file, and at the same time process (e.g. post-integrate) that very same data. According to the principles of high cohesion and low coupling mentioned previously, these tasks will be handled by two entirely separate processor components. Both these components will connect to the same incoming shared memory interface. ARCOM's **AShMemInterface** implementation supports a virtually unlimited number of simultaneously connected clients. A maximum number of simultaneous clients needs to be specified when creating any particular instance of a shared memory interface, though, as each client 'slot' takes up space in **AShMemInterface**'s internal data structures. An **AShMemInterface** maintains separate head and tail pointers for each client, and only frees up memory blocks that have been processed by all clients. See section 8.4 for details on the internal workings of an **AShMemInterface**. Should a client request to read more data than currently available, the function call will block (i.e. the client component will go to sleep) until enough data becomes available. This is analogous to normal synchronous I/O system calls.

8.3.2 Multi-master

Not only will more advanced pipeline configurations require the 'forking' of one data stream into several separate branches ('multi-client' above), the opposite will also be required: data produced by several different (recorder) components will have to be funnelled (multiplexed) into one data stream. For example, one might want to combine raw incoming data, processed (e.g. post-integrated) data and current readings from environmental sensors into one data stream that then gets written to disk. In this scenario, each recorder component acts as a master to the same shared memory interface. The ARCOM **AShMemInterface** shared memory interface implementation supports an unlimited number of masters for any given instance of a shared memory interface. Similarly to the maximum number of clients, a maximum number of masters needs to be specified when creating any particular instance of a shared memory interface, as each master 'slot' takes up space in **AShMemInterface**'s internal data structures. One curio about multi-master capability is the existence of what we term a 'ghost-master.' As a shared memory interface can exist without any connected masters (for example during system shutdown, or when components are swapped in and out during run-time), an always present (but invisible to clients) 'ghost-master' helps to ensure data consistency in these situations. See section 8.4 for details on these internal data structures.

8.3.3 Block-based

Reads and writes from and to the shared memory interface happen in blocks of arbitrary sizes. This makes the ARCOM shared memory interface ideal for streams of data packets, where each packet consists of a well-defined, fixed-size header followed by a payload of arbitrary length. Although the shared memory interface itself knows nothing about the specifics of ARCOM packet streams (see section 8.6), the knowledge that incoming and outgoing data is packet-based (block-based) allows the ARCOM shared memory interface to ensure consistency, e.g. when dealing with buffer overflow situations. At the same time, since detailed knowledge about the packet format is still the user's (client's/master's) responsibility, loose coupling is maintained.

8.3.4 Simultaneous Read

As a separate 'slot' is allocated for each client, many clients can read data from the interface quasi-simultaneously. Memory will only be freed up (reused) once the last client has read the data.

8.3.5 Simultaneous Write

Even more importantly, our implementation of a streaming shared memory interface allows multiple masters to *write* to the interface quasi-simultaneously. A master opens a memory block of a certain size, and can then take as much time as needed to write data to the open block of memory. Meanwhile, other masters can in turn open memory blocks as required. Memory blocks will only be made readable to clients once the master has finished writing to the open block of memory and closed it. Note that all 'open' requests are serviced in the order in which they arrive, the interface acts as a time multiplexer for all incoming data streams.

8.3.6 Diagnosis

An **AShMemInterface** comes with a set of diagnostic functions to retrieve and visualise current allocation information. An example of automatically generated graphical 'snapshots' of the state of an example shared memory interface can be found in figure 8.11 as used during the discussion of the internal data structures (section 8.4).

8.4 Shared Memory Interface Internals

Access to ARCOM shared memory interfaces is encapsulated into the **AShMemInterface** class and its subclasses (figure 8.9). **AShMemInterface** encapsulates all functionality common to both read (**AProcShMemInterface**) and write (**ARecShMemInterface**) access to the interface. **AShMemInterface** is itself a valid class (as opposed to an abstract base class – ABC) and can be instantiated directly if neither read nor write access is desired, e.g. for diagnostic purposes, aka 'spy mode.' In this case, only the state transitions on the left side of the diagram in figure 8.10 are possible. As it is being used by an ARCOM component to access an ARCOM shared memory

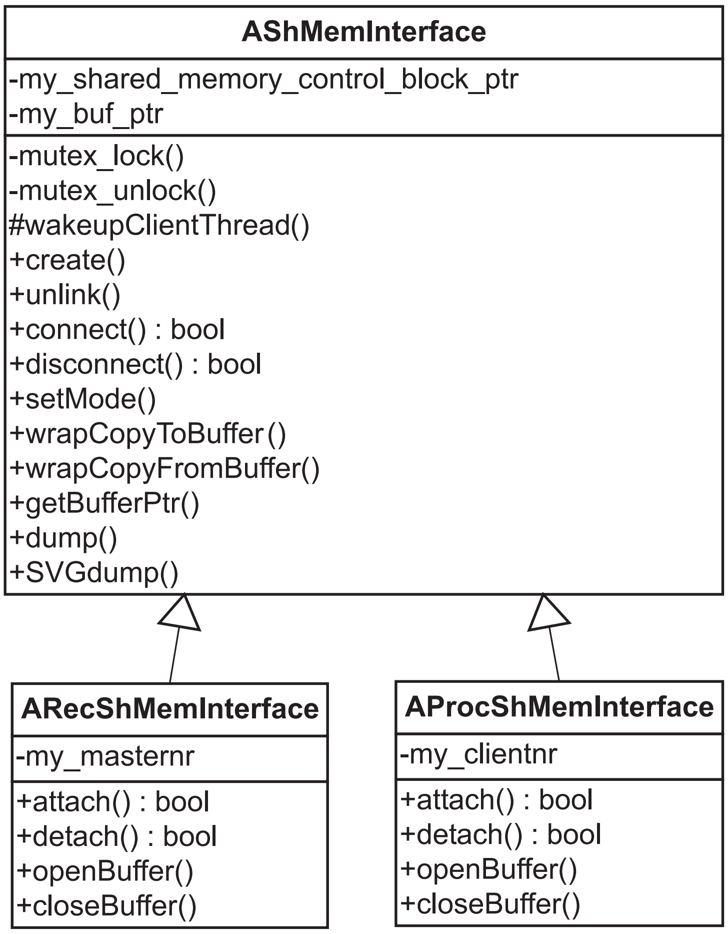


Figure 8.9: **AShMemInterface** class hierarchy including the major class attributes and methods of interest to users of shared memory interfaces.

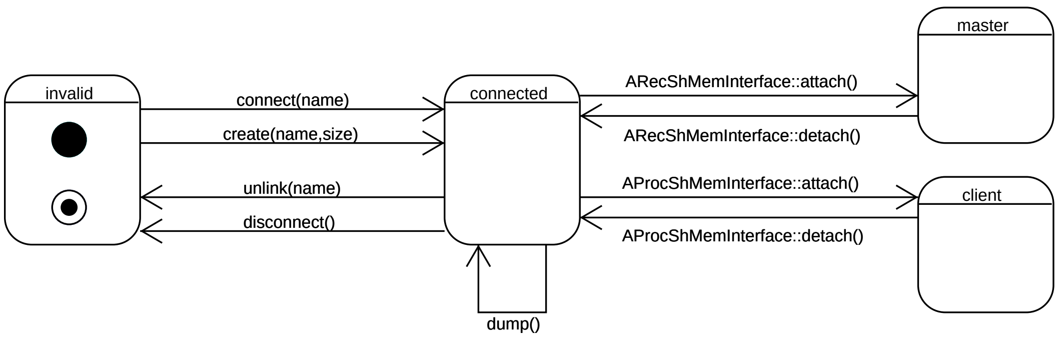

invalid
connect(name)
create(name,size)
unlink(name)
disconnect()
connected
dump()
ARecShMemInterface::attach()
ARecShMemInterface::detach()
AProcShMemInterface::attach()
AProcShMemInterface::detach()
master
client

Figure 8.10: **AShMemInterface** states

interface, any given instance of an **AShMemInterface** object will go through the states shown in figure 8.10.

Internally, each shared memory interface consists of a block of shared memory organised as a ring buffer. A separate control block maintains head and tail positions for each connected master and client. The **AShMemInterface** class and its descendants encapsulate access to memory and control block and sequentialise access in order to keep both blocks in a consistent state at all times.

Both clients and masters attach and detach to/from a shared memory interface using the **attach()** and **detach()** methods. They then request and relinquish permission to access a block of memory within the ring buffer of the shared memory interface by calling **openBuffer()** and **closeBuffer()**. This is roughly analogous to 'normal' file input/output operations. The success or failure of **openBuffer()** also depends on the operating 'mode' that has been specified for the interface in question. In 'blocking' mode, the call to **openBuffer()** blocks until the requested amount of memory becomes available. In 'gentle' mode, an **AExWouldBlock** exception is thrown. For master access, a third mode option exists: in 'force' mode, a block of memory is forced open, and any clashing clients are forcibly disconnected. This ensures that high-priority masters (for example an **A9812Recorder** component recording data from an A/D converter) will never have to wait for slow clients further down the pipeline. This mode is invalid for client access, as one cannot 'force' data to appear out of thin air.

Figure 8.11 shows some example transactions reading and writing data from/to an example shared memory interface object 'DemoShMem.' The individual plots were generated by calling the **SVGdump()** diagnostic function of the **AShMemInterface** instance for the interface in question, resulting in an SVG (Scalable Vector Graphics) format file depicting graphically the current internal state of the shared memory interface. The example itself was generated using the command line tools in `tools/shmemplayground/` to manually create, read from, write to, spy on, etc. ARCOM shared memory interfaces. These tools are described in section 8.9.4 below.

The following is a description of the individual panels in figure 8.11, representing specific successive moments in time. This description will serve to explain the data flow and internal data structures working 'behind the scenes' inside an ARCOM **AShMemInterface**.

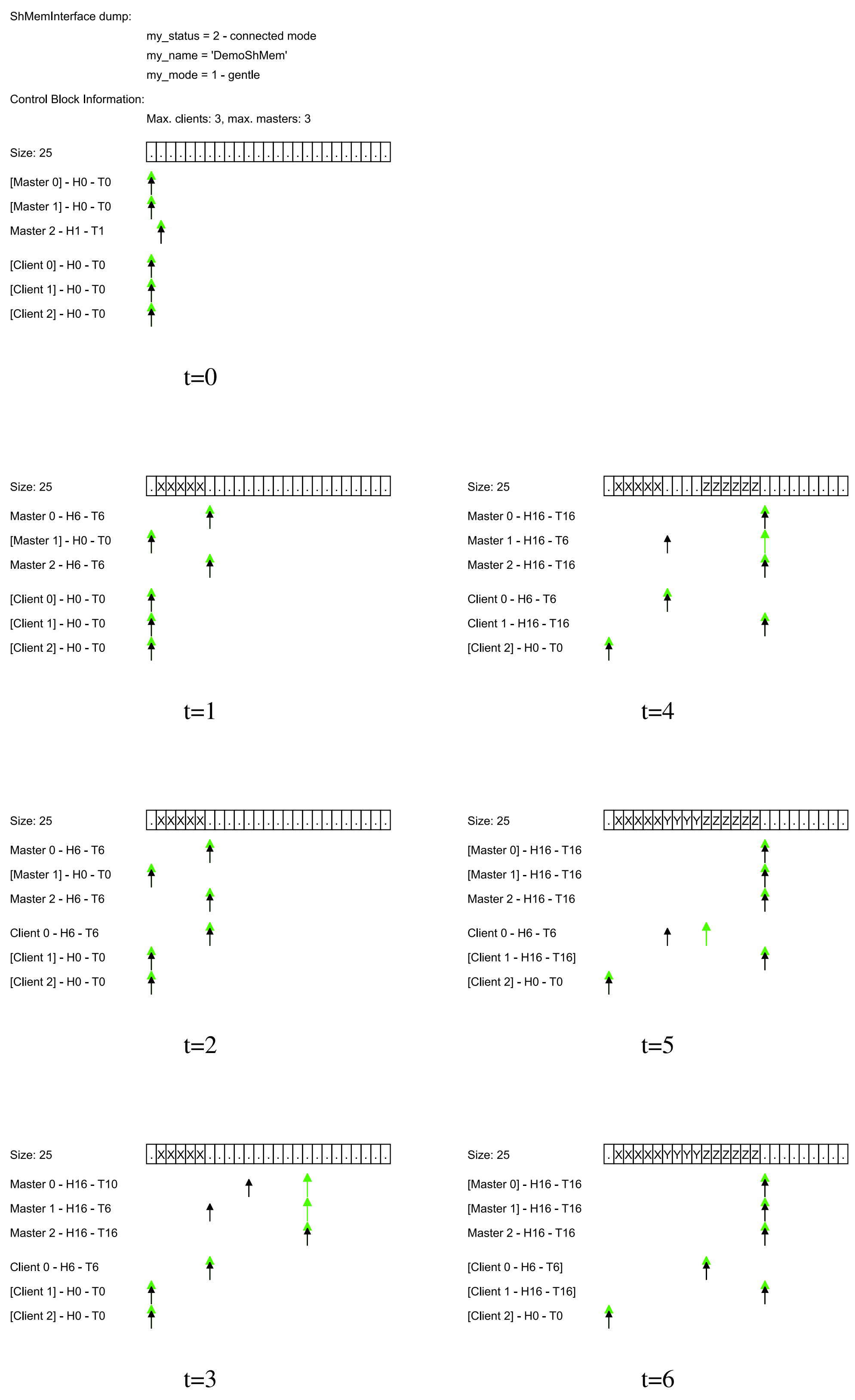


Figure 8.11: An ARCOM shared memory interface in use. Tall (green) arrows indicate head pointer position, short (black) arrows indicate tail pointer positions. For further description see section 8.4.

t=0

The interface has been created but no clients or masters are yet connected, indicated by the square brackets around all client and master slots. Note that there seems to be one master connected already (master 2). This is the ‘ghost-master’ that is always present and whose role in maintaining a consistent dataset will become clear towards the end of this discussion. Head and tail of the ‘ghost-master’ point at the same location, indicating an empty interface.

t=1

A master has connected to the interface. This master has written a block of five ‘X’ bytes to the interface. Note how head and tail for both the real master (slot 0) and the ‘ghost-master’ (slot 2) are incremented.

t=2

A client has connected but not yet read any data. Newly connected clients automatically start off at the ghost-master’s head position, i.e. they will not see any data that is still in the process of being written by a master at the time they connect.

t=3

A second master (master 1) has connected and opened a block of four bytes for output to the interface. No data has been written yet. The first master (master 0) has then also requested to write a block of 6 bytes by calling **openBuffer(6)**. Both buffers are still open, indicated by the lagging tail pointers for the masters. Note how all master head pointers point at the next free position in the ring buffer.

t=4

Master 0 has finished writing to its data block and called **closeBuffer()**: head and tail point at the same location. Master 1 is still writing data to its part of the ring buffer. Another client has connected (Client 1) but cannot yet read any data.

t=5

Master 1 has finished writing to its data block. Both masters have disconnected (indicated by the square brackets around the master slots). Note that the ‘ghost-master’ still exists, ensuring that client heads and tails are kept in a consistent state and all clients can read all the remaining data, even when no (external) master is connected to the interface. In fact, client 0 is in the process of reading a block of four bytes. Client 1 still cannot read any data because no masters are connected that would be capable of supplying new data.

t=6

Both clients have disconnected. The interface is empty again. Client head and tail pointers show the last status just before the **detach()**. They will be re-initialised to the ghost-master's head position when a new client connects.

8.5 Log File Handling

Although not explicitly mentioned as a goal for riometer operating software, the fact that riometers are instruments designed for long-term observations implicitly acknowledges that data storage for later retrieval is an essential part of their functionality. From the structural discussions above it has already become clear, that the task of writing data to files on disk ('log files') will primarily be the responsibility of one appropriately designed 'processor' component providing a data sink at the end of some processing pipeline.

Even so, various other parts of the overall system will have to deal with these data files later on, for the purposes of transferring, translating, post-integrating or generally reading and analysing them.

For this reason, all functionality related to the handling of log files in the widest sense was encapsulated in the class **ALogFile** and its children, see figure 8.12. Apart from the obvious 'write data to a file' functionality, this set of classes incorporates additional functionality stemming from experiences with previous instruments and the knowledge of problems that can and do arise with these.

In particular, **ALogFile** provides the following 'above standard' functionality:

8.5.1 Date/Time Awareness

Although **ALogFile** itself cannot actually determine the current time (a task that is left to the client), **ALogFile** is aware of the concept of time and can use this knowledge when it comes to automatically determining appropriate name sequences for log files (section 8.5.3) and for reading data from several successive log files (section 8.5.6).

8.5.2 Automatic Intelligent Splitting

Depending on the type of data that is being recorded, log files can quickly reach rather unwieldy file sizes in relatively short time spans. For example, when logging data that has an incoming data rate of, say, 12MB/s (as during the October 2002 ARIES experiment, for example), a 30 minute recording will reach a size of 22GB, not including any overheads. It is usually considered good practice to limit the size of any given log file. Files above a certain size tend to be awkward to process, transfer and analyse. Some filesystems can only store files up to a certain size limit (for example 4GB on FAT filesystems [Mic07]).

An **ALogFileWriter** will ensure that any given log file will never exceed a certain size limit. It does this by creating a new file with increased sequence number whenever the current file is about to reach the size limit. Any given log file might well end up being (slightly) smaller than this limit, as **ALogFileWriter** makes sure that any particular chunk of data it is asked to write to the file will always end up as a whole, complete chunk in the file. This means that, even though **ALogFile** and relatives do not know anything about ARCOM packets or, in fact, any other logical units contained in the data stream, clients (first and foremost an **ALogger** component in our case) can ensure that packets are never truncated/corrupted, simply by always passing the whole packet on to **ALogFileWriter** in one go.

8.5.3 Flexible Automatic Naming (Timestamping)

Log files need to be organised in some clear manner to enable later retrieval of datasets. For long-term observations, some way of timestamping files is the obvious and widely used method of choice. Once again sticking with the object-oriented (OO) principle of high cohesion, **ALogFile** itself is not aware of current time, but instead relies on its caller to inform it about time and time changes. It will then, however, automatically create appropriate file- and directory names based on a freely configurable file name template.

Separation of time handling and file naming means that, for example, **ALogFile** can easily be used to read and write files with timestamps in the past or future, for example for replaying previously recorded events. Time information can come directly from the data written to file, or from other sources like the local system clock or even an external GPS receiver, without any changes being required to **ALogFile**.

8.5.4 Overwrite/Out-of-order Protection

In the past, instruments have had to cope with the effects of unreliable time sources. Often, this would result in the instrument overwriting existing data files (if the clock erroneously jumped backwards) or creating incorrectly timestamped data files that would later on either get lost or overwritten (if the clock erroneously jumped forwards).

ALogFile has a concept of ‘open’ versus ‘finalised’ files and directories. Files marked as ‘open’ (as in ‘open for subsequent write requests’) will allow the client to append data to them. Once **ALogFileWriter** determines that the a new log file should be created, most likely because of reaching the size limit or because of a change of time that results in a new file name, the file will be closed and marked as ‘finalised,’ from which moment onwards **ALogFileWriter** will refuse to open that file ever again in write mode.

A similar concept applies to directories, therefore implementing a 2-tier safety net: If the file template is set up to, say, store files in subdirectories according to the current day of the month, and a transition from one day to the next takes place, **ALogFileWriter** will not only finalise (mark as ‘finalised’) the file itself, but also the subdirectory that contains it. All attempts to write to that subdirectory in future will get rejected, causing a switchover to ‘emergency mode’ (see below).

8.5.5 Fallback (Emergency) Mode

In addition to problems with inaccurate clocks, other error conditions can also arise, first and foremost an out-of-diskspace error. If at all possible, we want an instrument not to stop recording even in the presence of certain error conditions. Like the out-of-order protection mechanism described above, these conditions will trigger the transition into what we term ‘emergency mode.’ Specifically, ‘emergency mode’ gets activated automatically on encountering any of the following (error) conditions:

- Attempt to write to a ‘finalised’ file.
- Attempt to write to a ‘finalised’ directory.
- Disk full error while writing to a log file.
- Manual ‘switch to emergency mode’ command.

In ‘emergency mode,’ files will get written to a different path as specified using **setEmergencyTemplate()**. It is a good idea to point this template to a physically separate disk or at least partition.

In ‘emergency mode,’ files will no longer be timestamped (as the time information is potentially unreliable anyway), but will simply bear increasing sequence numbers. As before, once a file has been closed (‘finalised’), it will never be written to again.

In the case of ARCOM, files that have been written in ‘emergency mode’ can be further processed using an appropriate pipeline of ARCOM packet tools (see section 8.9.5 below). In this way, they can, for example, be reshaped appropriately for inclusion in the proper data archive, once the error condition has been rectified.

8.5.6 Fuzzy Search

When it comes to reading log files, the exact name of the required file might not always be known, for example because of clock inaccuracies. Also, a data archive might contain gaps due to, for example, instrument failures.

The **findFileForReading()** function has the ability to automatically scan for the next available file starting from a given time. It does this by looping through a number of potential file names relating to the specified as well as future dates — up to a maximum specified by **setRetries()**. Both ‘open’ and ‘finalised’ files (see section 8.5.4 above) are considered in this search. This functionality comes in very useful in, for example, the `readfromlogfile` ARCOM packet tool (described in section 8.9.5 below), which will usually be able to locate the required data file without too much fuss. (Note that `readfromlogfile` also implements some more advanced backtracking capability, but the ability to do so requires knowledge of the ARCOM streaming data format, which is beyond the knowledge horizon for **ALogFile**.)

8.5.7 Conclusion and Evaluation

The advanced logging concept as presented in this section has generally been found to perform well. The fact that file and even directory names can change not only as they are being recorded but also at a later stage (see the section on overwrite protection above), has, however, caused issues with near-real-time file transfers via simple `rsync` or `scp` commands. It is not necessarily straightforward to predict the appropriate file name to use in the transfer script, and even more severely, the archive that is built up at the remote end will either not match the source archive, because of the file name changes, or the renamed files will be transferred twice or have to be renamed a-posteriori at the client end, the latter again not being straightforward.

This is compensated by the fact that errors and data corruption due to timing issues are now greatly reduced and error conditions can much more easily be detected. If nothing else, the fact alone that there exist ‘open’ files in an archive is a strong indication that something went wrong at some stage.

Also, ARCOM does of course provide its own built-in mechanism for real-time data feeds, see section 8.8.5, thereby potentially eliminating the need for real-time file-level transfers.

Implementation-wise, it has to be said that in this instance the parent/child relationship between **ALogFile** and **ALogFileReader/ALogFileWriter** is abusing the pure teaching of object-oriented programming in that the relationship between reader/writer and **ALogFile** is more of a ‘makes use of’ relationship than an ‘is a’ relationship, i.e. inheritance is (ab)used as a practical way of sharing common pieces of code although it does not actually accurately describe the relationship between the two classes of objects. This design decision is acceptable in this case, though, as **ALogFileReader** and **ALogFileWriter** essentially provide a certain functionality to their clients, and the knowledge that they are both derived from a common class is unnecessary/irrelevant from a client’s perspective.

A final important property of **ALogFile** worth noting is that, despite all the intelligence relating to **ALogFile** and relatives, the log files that end up on disk are still 100% pure ARCOM streaming data (see section 8.6) files, i.e. a stream of valid ARCOM packets. This is because **ALogFile** does not add or take away any of the information it is asked to write to files, so as long as it is being fed with valid ARCOM packets, the result will always be an archive of valid ARCOM packets (even though **ALogFile** itself does not even know about packets as such). This means that existing operating system tools such as `cat` can be used to perform simple tasks such as concatenating several files. For more advanced processing, the (ARCOM-specific) tools described in section 8.9.5 would generally be used.

8.6 The ARCOM Streaming Data Format

As already noted in section 8.2, any pipelining architecture requires its individual processing stages (components) to agree on a common data format for passing data down the pipeline. The flexibility requirements faced by ARCOM compared to existing riometer systems are unique in that the data format should accommodate both existing well-defined datasets and data produced by yet-to-be-defined-and-implemented future components.

Generic multi-purpose data formats have come a long way, with XML (eXtensible Markup Language) being the most prominent of these formats in recent years [BPSM+06]. XML is uniquely flexible in that it is an inherently extensible format that can be customised to essentially represent any possible dataset.

In fact, XML is widely used in inter-component communications, being the foundation of such standards as SOAP (Simple Object Access Protocol) used in today's Web- and Gridservices [GHM+07]. The major drawback of XML when it comes to inter-component communication between, in this sense closely coupled, components in a processing pipeline, is its text-based format. Parsing XML data, which allows for both variable-length tags and variable-length content is processing intensive and therefore slow.

Several attempts are underway that try to resolve these issues by introducing a 'binary' format based around similar ideas. These attempts include "Fast Infoset" that encodes XML into a compressed binary format [CT04] and many other proprietary solutions. However, none of these formats were readily available and sufficiently defined at the design stage of ARCOM. Also, they all still suffer from an unnecessarily flexible approach as far as ARCOM data processing is concerned. However, it turns out that very similar requirements exist in quite a different area, namely digital television. Equipment compliant with the Digital Video Broadcast (DVB) standard uses pipeline architectures to multiplex, transmit, receive and de-multiplex streams of images, audio, subtitles and many additional, quite unrelated, datasets such as teletext or IP (Internet Protocol) packets. The underlying data format is the MPEG transport stream as described in [ETS97] and the many references therein, for a concrete case see, for example, [ETS03].

Inspired by this, ARCOM defines a common ARCOM streaming data format, which, on the highest level, consists of 'packets,' each packet in turn containing one or many 'descriptors.' This format combines all the advantages with respect to flexibility mentioned above, while still being fast to process and access:

- Flexible: Define packet types as required.
- Future-proof: Unknown packet/descriptor types are simply ignored.
- Generic: A basic set of functionality can deal with arbitrary data streams.
- Hardware-friendly: External (hardware) data sources can be made to 'speak' ARCOM

simply by encapsulating their data output into appropriate ARCOM packets, an approach very similar to 'tunnelling' in, for example, TCP/IP networking [BSB05]. This removes the need for translation between incoming data and the common ARCOM streaming data format.

- Fast to process and access: The binary format based on fixed-length headers is fast to process in hard- and software.

A very similar approach has since also successfully been used for the SPARKLE high-speed photometer [Gri06c], although in this instance memory bus and speed limitations of the micro-controller hardware required some simplification.

Tables 8.2 and 8.3 list all currently defined ARCOM packet and descriptor types, respectively. Figure 8.13 shows the definition of an ARCOMPACKET_FPGAPACKET packet as an example of a complex ARCOM streaming data packet. Note that although the figure shows the packet exactly as it is issued by a specific version of the FPGA firmware, ARCOM does at no stage rely on the detailed knowledge of at which offset to find what data. Instead, the packet is scanned for descriptors of interest, and the relevant data is extracted/processed, while unknown/uninteresting descriptors or packets that do not contain descriptors of interest are simply skipped over (i.e. ignored). See the ARCOM source-level documentation for details on all defined packet and descriptor types [Gri06a].

8.7 The CORBA Interfaces

The reason for designing the ARCOM architecture around a basic set of meta-components (adaptor, recorder, processor, see section 8.2) that all 'speak' CORBA on their control interface was so that each component could be controlled using the same set of CORBA commands. This is essentially the idea of polymorphism on a component level: a user can initialise, run, stop, etc. any given component without having to know about any internal details. This is heavily used by automatic startup and shutdown scripts such as the `executor.pl` script described in section 8.9.3. Figure 8.14 is a UML diagram of the CORBA interface hierarchy for ARCOM components. All components inherit the common commands **init()**, **terminate()** and **getStatus()** from the basic **AComponent** interface. In addition, all recorder and processor components inherit the **run()** and **stop()** commands.

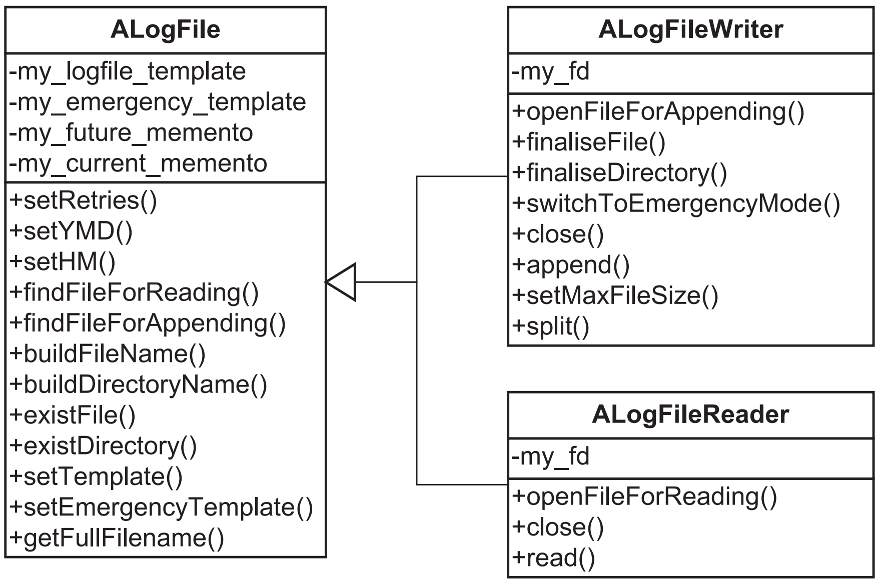


Figure 8.12: **ALogFile** and descendants

id	Packet Type
0x01	ARCOMPACKET_TEXT
0x02	ARCOMPACKET_GPS_NMEA
0x05	ARCOMPACKET_TIMESTAMP
0x06	ARCOMPACKET_BASICDATA
0x07	ARCOMPACKET_RAWFPGADATA
0x11	ARCOMPACKET_FPGAPACKET
0x12	ARCOMPACKET_SHMEMSTATUS
0x13	ARCOMPACKET_TEMPDATA

Table 8.2: ARCOM packet types

id	Descriptor Type
0x01	DESCRIPTOR_UTC_TIME
0x02	DESCRIPTOR_RAW_DATA_16BIT
0x03	DESCRIPTOR_CMPLX_DATA_64BIT
0x04	DESCRIPTOR_STREAM_ID
0x05	DESCRIPTOR_GPS_TIME
0x07	DESCRIPTOR_64_CPLX_POW
0x08	DESCRIPTOR_HOPF_GPS_INFO
0x09	DESCRIPTOR_RX_INFO
0x0a	DESCRIPTOR_ARIES_CPLX_DATA
0x10	DESCRIPTOR_BITSTREAM_VERSION
0x11	DESCRIPTOR_SHMEMNAME
0x12	DESCRIPTOR_SHMEMCTLBLOCK
0x20	DESCRIPTOR_8BIT_CPLX_DATA
0x21	DESCRIPTOR_16BIT_CPLX_DATA
0x22	DESCRIPTOR_24BIT_CPLX_DATA
0x23	DESCRIPTOR_32BIT_CPLX_DATA
0x24	DESCRIPTOR_FFT_TAPER_INFO
0x25	DESCRIPTOR_POST_INT_INFO
0x26	DESCRIPTOR_RX_DC_OFFSETS
0x27	DESCRIPTOR_UNIXTIME
0x28	DESCRIPTOR_TEMP_READING

Table 8.3: ARCOM descriptor types

getStatus() return value	meaning
error	component does not exist
0	component not yet initialised (‘newborn’)
1	component initialised but not running
≥ 2	component is running, exact meaning of status value is component-specific.

Table 8.4: ARCOM component status values

Word Index	Packet Description				
0	magic (0xABDE)			packet type 0x0011	
1	total packet size in bytes – 0x0000_403C				
2	checksum (ignored) 0x0000_0000				
3	Xcorr Data Descriptor	Descriptor Header	Did 0x0a	0x4004	0x00
4 - 67		Real Outputs	Y0*X(0 to 31)	LSBs	
				0x0000 & MSBs	
68 - 131			Y1*X(0 to 31)	LSBs	
				0x0000 & MSBs	
132 - 195			Y2*X(0 to 31)	LSBs	
				0x0000 & MSBs	
: :			: :	: :	
1988-2051			Y31*X(0 to 31)	LSBs	
				0x0000 & MSBs	
2052-2115		Imaginary Outputs	Y0*X(0 to 31)	LSBs	
				0x0000 & MSBs	
2116-2179			Y1*X(0 to 31)	LSBs	
				0x0000 & MSBs	
2180-2243			Y2*X(0 to 31)	LSBs	
				0x0000 & MSBs	
: :			: :	: :	
4035-4099			Y31*X(0 to 31)	LSBs	
				0x0000 & MSBs	
4100	Timestamp Descriptor	Descriptor Header	Did 0x08	0x0014	0x00
4101		Hopf GPS Time	Hours(tens) & Hours(units) & Week day & status		
4102			Secs(tens) & Secs(units) & Mins(tens) & Mins(units)		
4103			Month(tens) & Month(units) & Day(tens) & Day(units)		
4104			0x0000 & Year(tens) & Year(units)		
4105	Information Descriptor	Descriptor Header	Did 0x09	0x0014	0x00
4106		Info 1	Number of Integrated Values for this packet		
4107		Info 2	Number of ADC conversions between PPS (Rising Edges)		
4108		Info 3	Upper 16 bits are noise word1; lowest bit is noise mode; bit 12 is the actual noise_en fed to Host		
4109		Info 4	Noise word 2		
4110	Version Descriptor	Descriptor Header	Did 0x10	0x0004	Version 0x??

Figure 8.13: ARCOMPACKET_FPGAPACKET. Note how the packet is made up of 'descriptor' entities, allowing the packet structure to adapt to new requirements without breaking existing code. Diagram adapted from [Bar07].

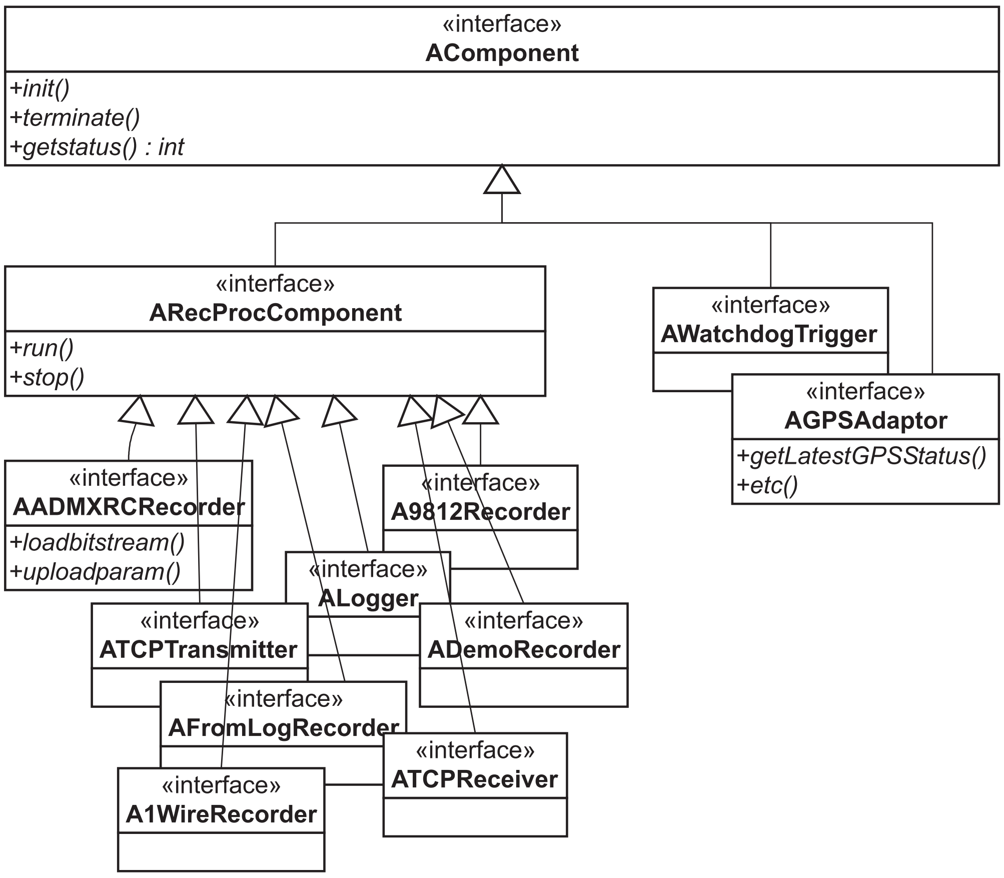
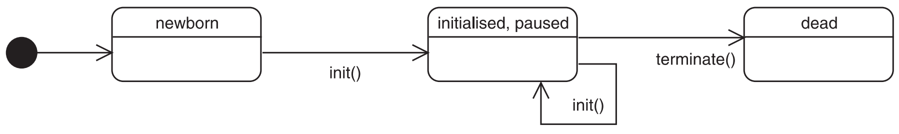
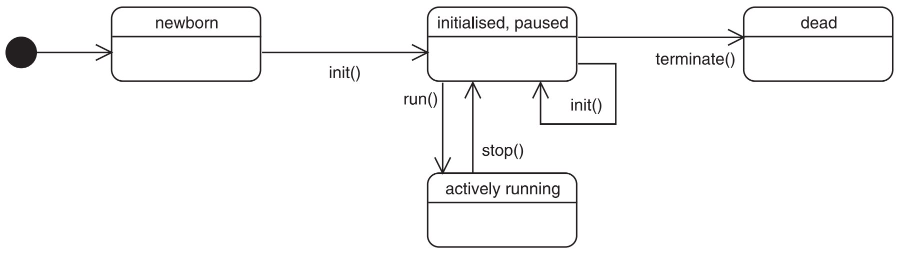


Figure 8.14: ARCOM CORBA interfaces



(a)



(b)

Figure 8.15: Life cycle of an ARCOM component. (a) Adaptor, (b) Recorder/Processor

These four basic commands (init, run, stop and terminate) are sufficient to take any ARCOM component through all stages of its life cycle, see figure 8.15. The **getStatus()** command is a generic diagnostic command to enquire about the current state of any component, the returned status value is defined as summarised in table 8.4.

Note that only the interfaces at the bottom of figure 8.14 do actually get implemented by real ARCOM components (although for simple components the interface might well be an unchanged copy of its ancestor). In this sense, interfaces in the top two levels of the ARCOM CORBA interface hierarchy can be described as 'virtual' interfaces, i.e. interfaces that are never directly implemented by any component. They only serve as a common base for derived interfaces and enable component-level polymorphism as described above.

Figure 8.14 shows that each individual ARCOM component can add to the common interface by defining its own component-specific messages. This should only be used sparingly, as it obviously breaks the uniformness of all ARCOM components and requires special user knowledge of the particular component in question. As can be seen, as of yet only the **AADMXRCRecorder** component implements additional messages, in this case for enabling dynamic reconfiguration of the FPGA hardware at run-time.

8.8 Selected ARCOM Components

All ARCOM components are configured through a central XML configuration file. Each component supports the following set of basic configuration options:

IORFilename Name of the file to store this component's CORBA Interoperable Object Reference (IOR) in.

id Unique integer ID to identify this component in log file messages.

In addition, each component may define additional configuration options.

8.8.1 AADMXRCRecorder

The ARIES riometer system employs an FPGA-based interface board to feed the signals from 64 digital receivers into the processing PC, see chapter 3, section 3.3 and appendix D. The **AADMXRCRecorder** ARCOM component is responsible for retrieving this data stream from the hardware and feeding it into the ARCOM infrastructure. This is a typical ARCOM recorder component (see section 8.2.4).

AADMXRCRecorder encapsulates all knowledge about this particular type of hardware. A number of configuration options enable flexible configuration based on the exact type of hard- and firmware. On top of the common options, the following configuration options are supported:

ShMemInterfaceName The name of the shared memory interface that we should connect to as a master.

cardindex The index of the ADMXRC card to use, in case there are several cards installed in the system.

bitstream The file name of the binary FPGA bitstream file to use.

bufsize Receive buffer size: The size of one DMA transfer.

lclk LCLK frequency in Hz.

vclk VCLK frequency in Hz.

DumpRawData Transmit ARCOMPACKET_RAWFPGADATA packets with the raw data as obtained from **DoDMA**()?

CookData Parse raw data and turn it into ARCOMPACKET_FPGAPACKET packets?

RawPacketFormat The type of packet to expect. Non-conforming packets are ignored.

AADMXRCRecorder also makes use of the ability to extend the existing ARCOM **ARecProc-Component** CORBA interface (figure 8.14), providing two additional methods **loadbitstream()** and **uploadparam()** for run-time firmware and parameter upload to the FPGA hardware, respectively.

8.8.2 A9812Recorder

The **A9812Recorder** component reads 1-, 2- or 4-channel digital data from a PCI-based AD-LINK PCI9812 A/D converter board [adl07] as used during the initial ARIES investigations (see chapter 9). **A8912Recorder** encapsulates all knowledge about this particular type of hardware. A number of configuration options enable flexible configuration based on desired mode of operation. On top of the common options, the following configuration options are supported:

ShMemInterfaceName The name of the shared memory interface that the component should connect to as a master.

samplingrate Requested sampling rate for synchronous sampling of all channels in Hz.

channels Number of channels to convert (0=only channel 0; 1=channels 0 and 1; 3=channels 0,1,2,3).

8.8.3 ADemoRecorder

The **ADemoRecorder** is an ARCOM recorder component that sends (test) data to a shared memory interface. It can generate different kinds and sizes of test data at user configurable time intervals. Some of the data produced can also be useful in production systems, for example the timestamp packets containing local time information.

On top of the common options, the following configuration options are supported:

ShMemInterfaceName The name of the shared memory interface that the component should connect to as a master.

interval How often the component sends a data packet to the shared memory interface (specified in seconds).

mode What type of packets to send to the shared memory interface. One of the following operating modes:

- random_small: send random ‘small’ data packets.
- random_big: send larger random data packets.
- random_both: send a mixture of ‘small’ and ‘big’ packets.
- timestamps: send an ARCOMPACKET_TIMESTAMP packet containing the current local system time.

8.8.4 ALogger

The **ALogger** component is a versatile logging component based around the **ALogFile** family of classes (see section 8.5 and also [Gri06a]). **ALogger** writes data streams to files on disk. Courtesy of **ALogFile** and derivatives, files are automatically named and split according to user preference. Section 8.5 outlines the novel functionality and basic design decisions behind the **ALogFile** family of classes together with a short evaluation and comparison to existing log file handling concepts. See the detailed source documentation for the **ALogFile** class for an implementation-level description of the concepts behind **ALogFile**.

On top of the common options, the following configuration options are supported by **ALogger** components:

ShMemInterfaceName The name of the shared memory interface that the ALogger component should read incoming data from.

maxfilesize The maximum file size of each individual log file in bytes. This size will not be exceeded in any case.

logfiletemplate Path to the log files. May contain placeholders for time and date information and sequence numbers. For a detailed description of the syntax of this entry see **ALogFile** in [Gri06a].

emergencytemplate Emergency path for log files. **ALogFile** will automatically switch into 'emergency mode' and start storing files at this location if it cannot write to the standard log file path, e.g. because of a 'disk full' error condition.

8.8.5 ATCPTransmitter and ATCPReceiver

The **ATCPTransmitter** component interfaces ARCOM shared memory interfaces to TCP/IP-based networks. An **ATCPTransmitter** component transmits (forwards) all (or selected) packets from an ARCOM shared memory interface to one or multiple TCP connections. Like all ARCOM components, **ATCPTransmitter** assumes that the shared memory interface contains valid ARCOM packets (section 8.6). From a TCP/IP point of view, the **ATCPTransmitter** is a server, as it simply waits for incoming connections, and on acceptance of a new connection starts transmitting data over this connection. From an ARCOM point of view, the **ATCPTransmitter** is a processor, since it processes data coming from the shared memory interface.

The **ATCPReceiver** is the counterpart to the **ATCPTransmitter**. On receiving a **run()** command through its CORBA interface, it establishes a TCP/IP connection to a (remote) **ATCPTransmitter** component. It then listens for incoming packets and outputs all valid ARCOM packets to its shared memory interface, therefore acting as an ARCOM recorder component.

These two components together allow for the operation of distributed ARCOM systems sharing common data streams. Note, however, that one of the design goals for the ARCOM shared memory interfaces was high-speed communication. As network communication is inevitably much slower and more unreliable than direct shared memory access, these components can only be used for moderate (on-site) or low (off-site) data throughput. They find their use in remote monitoring of non-critical summary data produced by summariser components (see also the suggestions for future system configurations in chapter 11) and in debugging and testing scenarios.

The following configuration options are supported by **ATCPTransmitter** and **ATCPReceiver**:

ShMemInterfaceName The name of the shared memory interface that the component should read data from (**ATCPTransmitter**) or write data to (**ATCPReceiver**).

tcpport Port to connect to (**ATCPReceiver**) or port to listen on for incoming connections (**ATCPTransmitter**).

server Only for **ATCPReceiver**: the IP address of the corresponding **ATCPTransmitter** component to connect to.

8.8.6 AFromLogRecorder

The **AFromLogRecorder** is an ARCOM recorder component that takes a stream of ARCOM data packets from a (set of) log file(s) and transmits them onto its shared memory interface. Thus, this component is ideally suited for replaying situations after they have happened. Equally well, it can be used to simulate and evaluate system behaviour with synthetically generated or simulated data, with the **AFromLogRecorder** replacing a ‘real’ (as in ‘recording from real hardware’) recorder component, but all the processor components further down the pipeline being the same as in an actual ‘real’ system. This component had already been used to stream (converted) data from old pre-ARCOM riometers through various ARCOM components before any physical hardware was available.

Similar to **ALogger**, the **AFromLogRecorder** component is based around the functionality of the **ALogFile** family of classes (see section 8.5 and also [Gri06a]).

The following configuration options are supported by **AFromLogRecorder**:

ShMemInterfaceName The name of the shared memory interface that this instance of **AFromLogRecorder** will connect to as a master.

interval Time interval at which to send out (bursts of) packets to the interface (time in seconds).

mode One of the following operating modes:

- pause_always: pause after every packet for the time given in <interval>.
- pause_after_data: send out packets in bursts, only pausing after sending a packet that is known to contain 'data.'

startdate The date and time for which to start transmitting data from the log file. ISO format (YYYY-MM-DD hh:mm:ss).

logfiletemplate Path to the log files. For a detailed description of the syntax of this entry see **ALogFile** in [Gri06a].

8.8.7 Future Components

During the initial evolution of the ARCOM framework, some components were 'left behind' and therefore only exist in old versions, and some only conceptionally (i.e. have never been implemented). Both these categories are included in the following list of future components. These components will fit into the existing ARCOM framework and can be implemented as required for future projects.

- **AIntegrator** to post-integrate incoming data to higher time resolutions (as used with **A9812Recorder** in figure 8.6).
- **ACrossCorrelator** to cross-correlate data streams (as used with **A9812Recorder** in figure 8.6).
- **AWatchdogTrigger** to re-trigger a hardware watchdog as long as all watched components and interfaces are alive and active.
- **ASummariser** to produce summary information about a data stream.
- **ACompressor** and **ADecompressor** to compress and decompress incoming ARCOM packet streams.

- **A1WireRecorder** to read temperature (and potentially other environmental data) from Dallas Semiconductor's 1-Wire temperature sensors [Max07].
- **AUDPTransmitter** to transmit real-time summary information to remote clients via the User Datagram Protocol (UDP).

8.9 Low-level Support Tools (ARCOM Tools)

Eventually, every architecture has to manifest itself in real-life applications to fulfil its purpose and prove its usefulness. This section describes the tools that were implemented for dealing with ARCOM components and their data inputs and outputs. These tools enable day-to-day operation and maintenance of ARCOM-based systems and are part of the ARCOM distribution. The following section (8.10) will outline some even lower-level implementation specifics, serving as a starting point for those who want to implement new ARCOM components.

8.9.1 ARCOM CORBA Message Dispatcher (`sendcmd`)

The CORBA standard itself defines generic methods for interacting with unknown CORBA components through dynamic interface discovery at run-time. However, as an easy way of interacting with running ARCOM components through their CORBA interface, a command line-based tool was implemented.

This tool already knows about the ARCOM specific CORBA interfaces, and can be used to manually request the status of individual components and change their current state. It can also be used internally by higher-level scripts to programmatically initialise, start, stop and check the status of any currently active component. Currently, these scripts include the `executor.pl` (section 8.9.3), the simple graphical user interface `gui1.tcl` (section 8.9.2) as well as the `init.d/arcom` startup script [Gri06a].

Usage examples

`sendcmd` takes the CORBA Interoperable Object Reference (IOR) of the desired component as the first parameter and the name of the method to invoke as the second parameter. Some messages require further parameters, and these are specified after the method name as required.

```
# ./sendcmd
syntax: ./sendcmd <ior> {init|run|stop|terminate|getstatus
```

```
                        | uploadparam|loadbitstream }
  This will invoke the respective method for the specified CORBA
  ARecProcComponent (including derived components).
```

As no parameters were specified, `sendcmd` prints a short help message that tells us how to use the `sendcmd` program. Note that only the first five commands are generic ARCOM commands, the remainder (uploadparam, loadbitstream) are specific to certain ARCOM components (the **AADMXRCRecorder** in this case).

```
# ./sendcmd `cat /arcom_running/aadmxrcrecorder-ior` getstatus
orb init - string to object - narrow - invoking getstatus()
status is: 2
```

This invokes the **getStatus()** method of the component described by the IOR in `/arcom_running/aadmxrcrecorder-ior`. Every ARCOM component stores its IOR in a file when it is started (the file name can be configured by one of the parameters in the component's section of the ARCOM configuration XML file, see section 8.10), and it is the contents of this file that we need to pass to the `sendcmd` tool using the UNIX 'backtick' mechanism [Coo06].

From the output of the command it can be seen that the **A9812Recorder** component in question is currently actively running (status ≥ 2, see table 8.4), i.e. transferring data from the receiver hardware to its shared memory interface, to be processed by other components further down the processing pipeline.

```
# ./sendcmd `cat /arcom_running/aadmxrcrecorder-ior` uploadparam
             param/param-1M4-noise-10s-on-3590s-off.param
orb init - string to object - narrow - invoking uploadparam()
transmitting the following data:
0000: ff ff ff ff  ....
0004: c0 5c 15 00  .\..
0008: 01 00 0a 00  ....
000c: 06 0e 00 00  ....
```

The above command will upload a new set of operating parameters as specified in the file `param/param-1M4-noise-10s-on-3590s-off.param` to the firmware running on the ADMXRC recording hardware. This is an **AADMXRCRecorder** component-specific command, and its invocation will fail if the addressed component is not of **AADMXRCRecorder** type. This particular example sets an integration time of 1.4 million samples and causes the ARIES receiver noise source to be switched on for 10 seconds every hour, but note that the exact format of the parameter set is application-specific and will not be discussed further here.

8.9.2 Graphical User Interface (`gui1.tcl`)

`gui1.tcl` (figure 8.16) is an easy-to-use graphical user interface (GUI) to control the state of ARCOM components. It builds on the functionality provided by `sendcmd` (section 8.9.1) for the actual component communication. For user convenience, the colour of the 'getstatus' button reflects the current status of the component in question: grey=no reply, red=newborn, orange=initialised, green=running. Note that this interface can only be used to send standard ARCOM messages to components, component-specific messages need to be sent through the `sendcmd` tool.

8.9.3 Automated Startup and Shutdown (`executor.pl`)

Manually starting up an ARCOM system with many components, and shared memory interfaces between them, can be an arduous and error-prone task. The `executor.pl` script automates this task. It can be used to start up and shut down a predefined ARCOM system configuration. `executor.pl` takes its commands from two special sections `<startup>` and `<shutdown>` in the system-wide ARCOM XML configuration file, see the following excerpt:

```
<arcomconfig>
 [...individual component configuration sections go here...]
 <startup>
  <option delay="5"/> <!-- default delay after each command in seconds -->
  <createshmem name="nice_little_shmem" size="1000000" masters="7" clients="8" />
  <launchinitrun type="alogger" name="default" />
  <launchinitrun type="ademorecorder" name="default" />
  <launchinitrun type="aadmxrcrecorder" name="default" />
 </startup>
 <shutdown>
  <option delay="2"/>
  <stopterminate type="alogger" name="default" />
  <stopterminate type="aadmxrcrecorder" name="default" />
  <stopterminate type="ademorecorder" name="default" />
  <unlinkshmem name="nice_little_shmem" />
 </shutdown>
</arcomconfig>
```

This example starts up a basic ARCOM system consisting of an **AADMXRCRecorder** (recorder) component, an **ADemoRecorder** (recorder) component and an **ALogger** (processor) component, connected through a one million byte shared memory interface. See [Gri06a] for details on all available commands and options.

8.9.4 Shared Memory Playground

For diagnostic and debugging purposes, it is useful to be able to interact not only with individual ARCOM components, but also with the shared memory interfaces that connect them. The various command line-based tools in the `tools/shmemplayground/` directory were developed for such low-level interaction. These tools allow the user to create and delete ARCOM shared memory interfaces, write raw data to them, read raw data from them, and produce diagnostic output of the current state of a given shared memory interface. In fact, the example in figure 8.11 (discussed in section 8.4) was created using these tools. Table 8.5 provides a brief summary of all available tools.

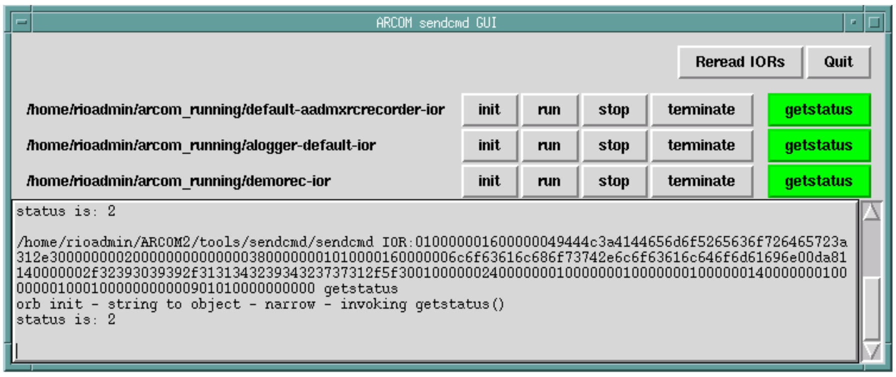
8.9.5 ARCOM Packet Tools

These tools allow the inspection and manipulation of ARCOM packet streams (section 8.6). Each tool can be invoked straight from the command line. Unless otherwise noted, these tools will take their input from `stdin` and output their processed result to `stdout`. This allows for them to be combined together by using standard UNIX pipes ('|'), very similar to the pipelining example given earlier (section 8.2 referring to figure 8.4). The following paragraphs will give a brief description of these tools, together with some commonly used examples. For more details on functionality and available parameters for any given tool, the reader is referred to the low-level documentation and the usage examples given therein [Gri06a]. All tools can be found in the `tools/arcom_packet_tools/` directory.

Note that each tool will also respond to being called with the '`-?`' parameter by outputting a help message containing all valid command line options with a brief explanation.

8.9.5.1 Generic ARCOM Packet Tools

`packetdumper.` Output a brief (1-line) or extensive clear-text dump of any incoming ARCOM packet. Can also be configured to only dump the payload-part of a packet, in either binary or hexdump formats.


ARCOM sendcmd GUI
Reread IORs
Quit
/home/rioadmin/arcom_running/default-aadmxrcrecorder-ior
/home/rioadmin/arcom_running/alogger-default-ior
/home/rioadmin/arcom_running/demorec-ior
init
run
stop
terminate
getstatus

```
status is: 2

/home/rioadmin/ARCOM2/tools/sendcmd/sendcmd IOR:010000001600000049444c3a4144656d6f5265636f726465723a
312e300000000200000000000000038000000010100001600000006c6f63616c686f73742e6c6f63616c646f6d61696e00da81
140000002f32393039392f313134323934323737312f5f30010000002400000001000000010000000100000014000000000100
000001000100000000000901010000000000 getstatus
orb init - string to object - narrow - invoking getstatus()
status is: 2
```

Figure 8.16: `gui1.tcl`: A simple GUI for controlling ARCOM components

tool	use to...
`createshmeminterface`	...create an ARCOM shared memory interface with the specified number of client and master slots.
`unlinkshmeminterface`	...delete (remove) a specified ARCOM shared memory interface.
`writetoshmem`	...output sequences of bytes to a specified ARCOM shared memory interface.
`writepacketstoshmem`	...output a selection of (hard-coded) ARCOM streaming data packets to a specified ARCOM shared memory interface. This is a predecessor of the more generic functionality available in the ARCOM packet tools (see section 8.9.5).
`readfromshmem`	...read sequences of bytes from a specified ARCOM shared memory interface as they become available.
`readpacketfromshmem`	...read ARCOM streaming data packets from a specified ARCOM shared memory interface. This is a predecessor of the more generic functionality available in the ARCOM packet tools (see section 8.9.5).
`spyshmeminterface`	...regularly print the current internal status of a specified ARCOM shared memory interface to the console. This will also create regular graphical representations in SVG format, just like the ones shown in figure 8.11.

Table 8.5: ARCOM low-level shared memory tools

packetplotter. Graphically plot the contents of incoming ARCOM packets. A variety of different packet types and plotting styles are supported. In combination with `readfromshmem` (below), this is useful for monitoring the contents of packets as they pass through a system's shared memory interfaces. The plotter is especially useful during the setup and calibration phases of instrument deployment.

packetfilter. Only packets of specified type(s) will be passed through, all other packets will be ignored.

readfromlogfile. Read ARCOM packets from a (set of) ARCOM log file(s) on disk. Start and end times of the period of interest can be specified, along with template strings to identify the set of log files to use. This tool automatically loops through all relevant ARCOM log files with the help of the **ALogFile** class (see section 8.5) and will also backtrack through earlier log files if the first file opened is found to already contain data newer than the specified start time.

readfromshmem. Connect to the specified shared memory interface and read packets as they pass through the interface. This is very useful for spying on traffic as it passes through an interface, for example in combination with `packetdumper` or `packetplotter`.

packetiser. Identify valid ARCOM packets in an incoming (garbled) data stream. This is useful to recover data from corrupted files, where parts of packets have been lost. The `packetiser` detects packet boundaries based on the 'magic' value in packet headers, and resynchronises in the case of broken packets. The output of the `packetiser` will always be a stream of valid ARCOM packets.

8.9.5.2 More Specialised ARCOM Packet Tools

The following tools are more specialised in that they are only useful for a small number of packet types. They will still deal with arbitrary ARCOM packet streams, of course, but they will only process the packets that they were designed for.

extractfpgadata. Export some of the data contained in FPGAPACKET ARCOM packets as lines of ASCII text. Useful for importing small amounts of data into third-party tools by means of a generic ASCII import filter.

`fpgapacketsanitychecker`. Look at all incoming FPGAPACKET packets and check if they contain continuously increasing (GPS) timestamps and a reasonable number of samples per integration period. Sudden jumps in time and number of samples will be detected and flagged up. Useful during instrument setup as well as during normal instrument operation.

`fpgapacketcropper`. Compress (crop) the data contained in FPGAPACKET packets by throwing away unneeded parts and reducing the resolution of integer numbers contained in FPGAPACKET packets. Which parts of the data are kept is determined by command line options and/or a separate configuration file. Useful for transferring selected datasets over low-speed network (modem) links.

`fpgapacketuncropper`. Reverse the cropping process done by `fpgapacketcropper` by expanding all data inside the FPGAPACKET packets back to its original size, filling any gaps with zeros.

8.9.5.3 Usage Examples

Hexdump of next arriving packet in shared memory interface:

```
# ./readfromshmem -n 1 -s nice_little_shmem -v | ./packetdumper -p
| od -t xC -A x
```

Summary info of all timestamp packets in log file starting 2005-11-17 00:00h:

```
# ./readfromlogfile -f '2005-11-17 0:0' -t '/arcom_log/%YYY/%YYY-%M
-%D-*/%YYY-%M-%D_%h_####_*' -v -d | ./packetfilter -t 5 -p
| ./packetdumper -aB
```

Read 300,000 packets from the log file, filter for only FPGAPACKET packets and extract ASCII data for offsets 1 and 33:

```
# ./readfromlogfile -f '2005-12-08 16:0' -n 300000  -t '/arcom_log/
%YYY/%YYY-%M-%D-*/%YYY-%M-%D_%h_####_*' -v -d | ./packetfilter
-t 17 -p | ./extractfpgadata -o 1 -o 33 > extractpower_2_200508.log
```

8.10 Component Implementation Details

This section explains how each individual ARCOM component is implemented by listing which files are involved and what they are used for. The **A9812Recorder** component will be used as an example, the structure being identical for all components.

The ultimate level of detail can be found in the extensive Doxygen-generated documentation [Gri06a], which can be found in the `doc/html/` directory of the ARCOM distribution.

Though all components are to some degree multithreaded [NBF96] through their use of the MICO CORBA implementation [PR00], especially recorder and processor components will usually implement (at least) a separate thread for activities performed during their 'running' state. For the exemplary component described here, this thread is contained inside the main implementation file (`aadmxrcrecorder_impl.cpp`), but could equally well be defined in a separate source file.

All files specific to a particular component are located in the component's directory, in this case `aadmxrcrecorder/`. Primarily, all that an ARCOM component does is implement its respective CORBA interface and the 'run' state behaviour. As figure 8.14 shows, **AADMXRCRecorder** implements two methods — **uploadparam()** and **loadbitstream()** — specific to this component. These are defined in **AADMXRCRecorder**'s CORBA Interface Definition Language (IDL) file `aadmxrcrecorder.idl`.

Table 8.6 shows the names of the important files in the component's directory together with their purpose. In addition to the files described in this table, each component will also make use of some files common to all ARCOM components, providing underlying ARCOM functionality. Except for the ARCOM configuration file (which can be located anywhere and is being read at run-time), these files are all located in the `common/` subdirectory, and table 8.7 lists the names and purposes of the more important ones. For details, the reader is referred to the extensive descriptions in [Gri06a].

8.11 Summary

The ARCOM software architecture provides a versatile run-time environment for a variety of scientific instruments and can readily be tailored to support a wide range of data acquisition and processing tasks. ARCOM is not limited to ARIES, or even riometers. The component-based approach, together with high-speed pipelining through dedicated shared memory interfaces, allows for unprecedented flexibility and run-time reconfigurability of an ARCOM-based instrument, thus successfully solving the 'Wicked Problem' of ever-evolving systems. The ARCOM architecture and associated data structures such as the ARCOM packet format are well-documented

file name	purpose
`aadmxrcrecorder.idl`	CORBA Interface definition for the **AADMXR-CRecorder** component, specified in IDL (Interface Definition Language). Note that every component's interface will inherit (at least indirectly) from the **AComponent** and, in case of a recorder or processor, the **ARecProcComponent** interfaces (figure 8.14). These two are purely 'virtual' interfaces that are not directly implemented by any component. They only serve as a common base for the derived interfaces.
`aadmxrcrecorder-_impl.cpp`	Implementation of the **AADMXRCRecorder_impl** class. This class defines the behaviour of the **AADMXR-CRecorder** component and needs to implement the CORBA **AADMXRCRecorder** interface as defined in `aadmxrcrecorder.idl`.
`aadmxrcrecorder-_impl.h`	Declaration of the **AADMXRCRecorder_impl** class to go with `aadmxrcrecorder_impl.cpp` (see above).
`main.cpp`	This file contains the **main()** function, i.e. the function that gets invoked by the operating system when the component gets invoked. **main()** simply instantiates an object of class **AADMXRCRecorder_impl**, registers it with the CORBA ORB and then waits until this object terminates itself (in response to a **terminate()** message). There is very little need to customise the content of `main.cpp`, as all component logic should be contained in the component implementation class **AADMXRCRecorder_impl**.
`aadmxrcrecorder-config.cpp`	Definition of the component's configuration class **AAD-MXRCRecorderConfig**. This class extends the generic **ARCOMComponentConfig** class (which in itself is derived from **XMLConfig**) and deals with parsing the AR-COM XML configuration file for component-specific configuration options. An instance of this object will be utilised during the initialisation phase (init). In case of the **AADMXRCRecorder** component, **AADMXRCRecorderConfig** is responsible for parsing all parameters outlined in section 8.8.1.
`aadmxrcrecorder-config.h`	Declaration of the **AADMXRCRecorderConfig** class to go with `aadmxrcrecorderconfig.cpp` (see above).
`makefile`	This file outlines how the binary executable for this component is built. It is read by `make` during the build process and usually requires little customisation. The make file also controls the (totally transparent) creation of CORBA stubs and skeletons for the component's CORBA interface.
`aadmxrcrecorder.cpp`, `aadmxrcrecorder.h`	These files are created automatically by the CORBA IDL compiler based on the CORBA interface definitions in `aadmxrcrecorder.idl`.

Table 8.6: Important ARCOM component files

and various tools support the user during setup and day-to-day operation.

Since its inception, ARCOM has also been deployed for the Advanced Imaging Riometer for Ionospheric Studies (AIRIS). Other instruments have also benefited from ARCOM concepts, for instance the new high-speed photometer for optical emission measurements (SPARKLE) developed by the author, which employs a packet-based streaming data format very similar to the one used by ARCOM.

file name	purpose
`logmacros.h`	Defines macros and helper functions for sending nicely formatted log messages to the `/var/log/messages` message log.
`arcompacket.h`	Declares and defines all data structures and helper functions related to the ARCOM packet data format as outlined in section 8.6. This file is the ultimate reference to all things **ARCOMPacket**.
`logfile/*`	These files define **ALogFile** and derived classes for writing to and reading from ARCOM streaming packet log files. See also the description of the **ALogger** and **AFromLogRecorder** components in sections 8.8.4 and 8.8.6 and the description of ARCOM log file handling in section 8.5.
`ashmeminterface/*`	The files in this subdirectory define the **AShMemInterface** and derived classes used to access ARCOM shared memory interfaces, see section 8.3.

Table 8.7: ARCOM files in the `ARCOM/common/` directory

Chapter 9

First Experiment Results

This chapter contains discussions of the different results obtained from the October 2002 ARIES experiment. During the October 2002 experiment, a variety of datasets has been recorded for different configurations of a preliminary ARIES system. The data recorded comprised several hundred gigabytes of raw input data as recorded from the A/D converters connected to the beam-forming network, as well as integrated data derived from the raw data in real-time. Integrated data is available for most of the time.

Also included is a short note on different ways of post-integrating data to justify the approach that was used during the experiment. This note also shows how our current way of post-integrating data from our IRIS riometer introduces (negligible) inaccuracies.

9.1 Experiment Setup

Figure 9.1 gives an overview of the hardware setup available during the experiment. Also, a widebeam riometer was set up using a separate crossed dipole antenna near the basement. This widebeam antenna has been recording power from 2002-10-20 until 2002-10-31 with only minor interruptions. In addition, IRIS data is available for the entire time span.

Receiving (and recording) capability was limited to two simultaneous channels. Due to the non-availability of Butler Matrices, phasing leads had been designed to allow additive beam-forming (see chapter 2, section 2.3.1) for the vertical case and a pre-determined 'worst-case' (beam 595) pointing direction.

The two single-channel receivers could be switched manually between the different configurations. Most recordings were taken with a 16+16 antenna configuration (ignoring the outermost

8 antenna elements on each side). A noise source and manual variable attenuator were also available for calibration purposes (section 9.6). Table 9.1 gives an overview of the major datasets that were recorded. In addition, raw input data was recorded for the following periods of time:

- 2002-10-28, 13:00h – 15:00h
- 2002-10-29, 07:00h – 09:00h
- 2002-10-29, 13:00h – 16:00h (alternating between sampling rates 1.1MHz and 2.2MHz)
- 2002-10-29, 17:00h – 19:00h

Output from the two prototype receivers was fed into an ADLINK 9812 A/D converter, which interfaced to the ARCOM software for logging, cross-correlation, post-integration and visualisation. Suitable software configurations as used during the experiment and data flow are presented in the discussion of ARCOM in chapter 8, see especially figures 8.6 and 8.8.

9.2 Note on Different Ways of Post-integrating Data

The current version of the MIA toolkit [Marc] post-integrates (IRIS) data by taking the mean of the dBm values. This appears to work fine so far. Nevertheless, the **setresolution**() method [Marb] has been designed with customisable post-integration functions in mind [Mard].

An alternative to the current approach is to take the mean of the linear power data in mW instead of the mean of the logarithmic data in dBm. It is worth investigating the different results one obtains with these different methods, this may also shed some light on why scintillation appears in IRIS data the way it does instead of, for example, averaging out [Mard].

Results of this comparison can be seen in figure 9.2. The figure shows data from one particular IRIS pencil beam (beam 9) during one particular day (2002-10-30). The yellow plusses are the original IRIS data as retrieved by the MIA **getdata()** function [Mara] for ‘one second’ resolution.[1]

The cyan line (shown with x-marks in the magnified insets) is averaged IRIS data as produced by MIA with the **setresolution()** function [Marb]. We can see that **setresolution()** integrates a certain amount of samples, returning the average of these samples. This reduces the time resolution of the resulting data, as can be seen clearly in the inset diagrams (cyan line).

[1]The term ‘resolution’ can be ambiguous, see section 9.3 for a more detailed discussion of the terms ‘resolution’ and ‘integration time’ as commonly used in MIA.

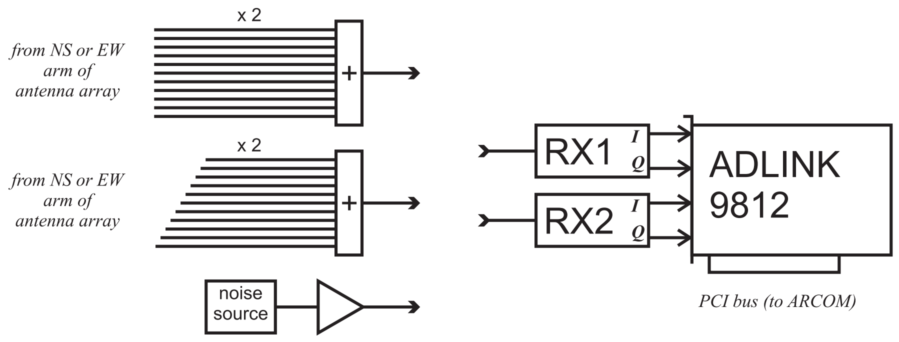

x 2
from NS or EW arm of antenna array
x 2
from NS or EW arm of antenna array
+
+
noise source
RX1
RX2
I
Q
I
Q
ADLINK 9812
PCI bus (to ARCOM)

Figure 9.1: Hardware available during the October 2002 experiment

date / time (UT)	configuration	available data
19/10/02 15:00 – 20/10/02 06:00	widebeam (X32) reception with 2 receivers, different gain settings	1.5s-integrated data sampled at 3.3MHz
20/10/02 17:00 – 21/10/02 04:00	ditto	1s-integrated data @ 3.3MHz
21/10/02 16:00 – 24/10/02 09:00	untapered 32+32 zenithal fan beams	1s-integrated data @ 3.3MHz, with gaps, a little cross-correlated data, partly wrongly calculated
24/10/02 21:00 – 25/10/02 12:00	ditto	1s-integrated data and 2s-cross-correlated data
25/10/02 21:00 – 26/10/02 18:00	incorrectly wired 16+16 beam	1s-integrated data and 0.5s-cross-correlated data
26/10/02 21:00 – 27/10/02 22:00	zenithal 16+16 fan beams	ditto
27/10/02 23:00 – 04/11/02 10:00	worst-case 16+16 fan beams	ditto
04/11/02 13:00 – 05/11/02 13:00	worst-case 32+32 fan beams	ditto

Table 9.1: Available datasets as recorded during October 2002 experiment

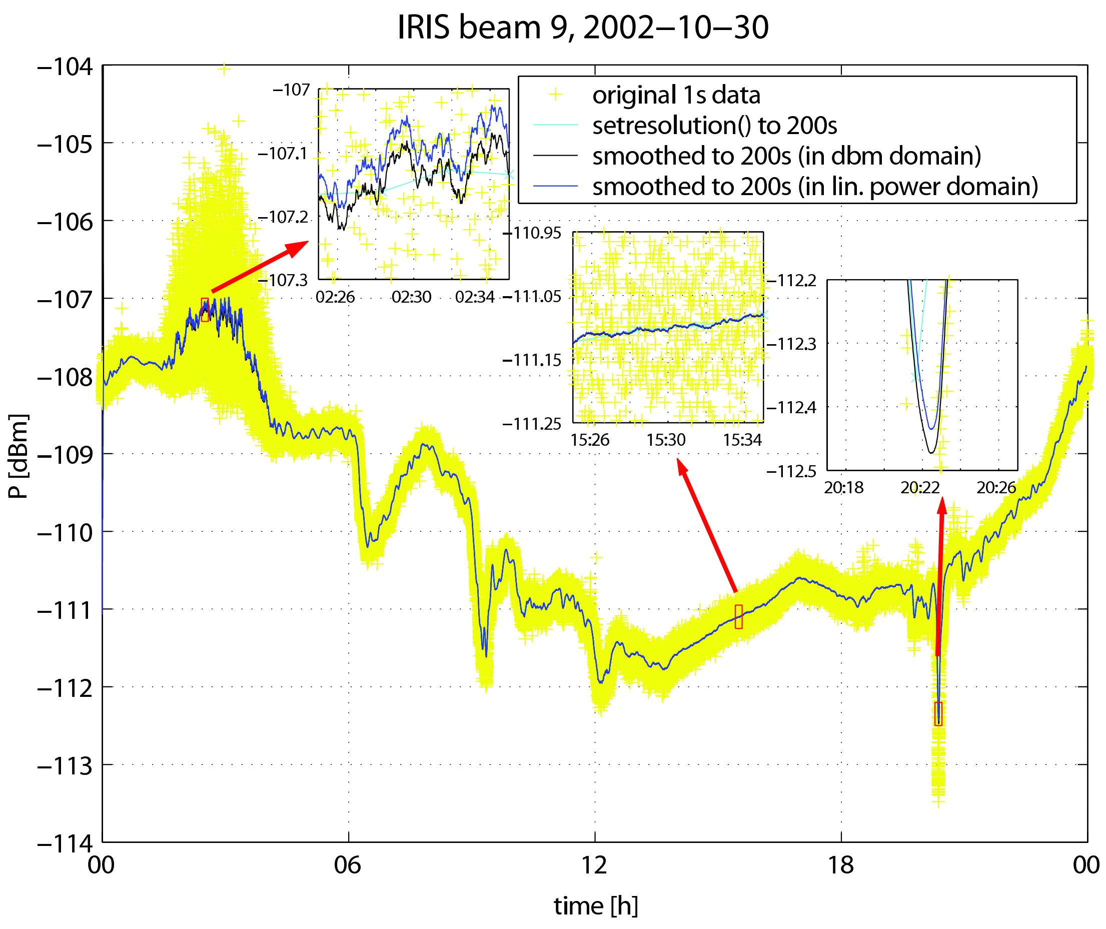


Figure 9.2: Different averaging methods, demonstrated for IRIS pencil beam data

One way to maintain the high time resolution while still averaging the data is to use a sliding window to calculate the mean around every original sample. This maintains the original time resolution. Apart from this, the results obtained by this method are identical to the previous ones (black line).

Now to the difference between post-integrating (averaging) the dBm-values instead of the original linear power values. The black line shows the yellow dBm-data post-integrated to 200s integration time (i.e. averaged over 200 samples). This gives essentially the same result as obtained by the MIA **setresolution()** function (the cyan line), only with higher time resolution: all samples on the cyan (low time resolution) line coincide with their counterparts on the black line.

On the other hand, the blue line shows the result of integrating the original yellow data in the linear power domain, i.e. before converting the values to dBm. We can observe the following facts:

- During quiet times (inset 2 around 15:30 in figure 9.2), no significant difference between the two methods can be noticed. In fact, if we magnify the diagram further, we find that the difference between the two curves is approximately 0.001dBm.
- During periods of strong scintillation (inset 1 around 2:30 in figure 9.2), post-integration in the dBm domain gives a result around 0.04dBm below the linearly post-integrated result. We cannot easily convert this absolute value into a percentage since during times of scintillation it is unclear which value should be used as a reference.
- Approximately the same effect can be observed for periods of absorption (inset 3 around 20:22 in figure 9.2). The dBm post-integrated values are around 0.03dBm below the linearly post-integrated values. This translates directly into an error of about 1.5% relative to the absorption at the time (around 2dB).

9.2.1 Conclusion

From the observations above we can conclude:

- Post-integrating data in the dBm domain does introduce a slight offset from the ‘correctly’ (i.e. linearly) integrated values.

- This offset is negligible during quiet times, and about the same as the specified resolution of IRIS (0.05dBm) during times of scintillation or absorption.
- This will affect IRIS absorption data, because the quiet-day curves used to calculate absorption are produced from quit-day recordings, which we have found to have only a very slight offset, whereas the current power data, especially during absorption events, will have a higher offset.
- It is not deemed necessary to change the algorithm that is currently used in MIA for IRIS data, as this would involve reprocessing all the existing data and no significant increase in accuracy can be expected because of the small values involved.
- However, for the discussions in the following sections, we will use the linear post-integration method for both IRIS and ARIES data, mainly because ARIES data is not (yet) fully integrated into MIA anyway.

9.3 Note on the Terms ‘Resolution’ and ‘Integration Time’

In MIA, the terms ‘resolution’ and ‘integration time’ are sometimes used slightly ambiguously. ‘Resolution’ primarily refers to the time resolution of the dataset in question. For example, if we have one sampled value for each second, the data is said to have a ‘resolution’ of 1s.

However, in MIA, the term ‘resolution’ often implies a certain integration time, i.e. the time during which the input data was integrated. Now, if we change the *resolution* of a dataset from, say, 1s to 2s using the MIA **setresolution()** function [Marb], not only will this result in only half the number of samples, it will also post-integrate the samples, so that each resulting sample is the mean of two original samples.

In other words, as well as having changed the (time-)resolution of the data, we have also changed the effective integration time. Most of the time, this is what we intended to do anyway.

Sometimes, however, we want to maintain the high time resolution even though we are post-integrating the data. This can be achieved with a sliding window algorithm (filter) as mentioned in section 9.2. It is also worth noting that data from an IRIS type system that is commonly referred to as ‘1s’ data (meaning a time resolution of 1s as well as an integration time of 1s) has in fact only been integrated for approximately 47ms due to the working principle of the IRIS receiver hardware. See section 9.4.2.1 for a more detailed discussion of this topic.

9.4 Relative Noise Intensity ARIES–IRIS

From previous mathematical and simulated results (see especially chapter 5) we expect the cross-correlated ARIES fan beam data to be noisier compared to IRIS data for any particular integration time. However, a first look at the ARIES results showed that "the width of the 1s-traces in ARIES and IRIS are compatible" [Nie02a] i.e. they are approximately the same width. This is unexpected and we will try to shed some light on this in the remainder of this section.

9.4.1 Expected Result

We expect the noisiness of the signal to become worse for shorter integration times and/or smaller bandwidths. Generally, we expect the signal from an ARIES pencil beam to be much noisier than a signal from an IRIS pencil beam. Reasons for this are:

- An ARIES pencil beam is looking at a much smaller area of the sky than an IRIS pencil beam. Therefore, we expect to receive less power, so the signal-to-noise ratio will decrease.
- ARIES pencil beams are formed through cross-correlation of two fan beams. The fan beams themselves have strong sidelobes and will pick up strong signals from all over the sky. It will therefore take longer to isolate the pencil beam by cross-correlation compared to how long it takes to form a pencil beam with a filled array (see the simulations done in chapter 5).

9.4.2 Analysis

The relative noise of the data was determined by calculating the standard deviation s of several different detrended datasets. The standard deviation s gives a measurement for the 'width' of the trace; 99.7% of all data points will fall into a $6s$ interval. Therefore we now have a means of plotting the trace width of recorded data for different parameters, first of all for different integration times.[2]

Of course, this can also be done for existing IRIS data, therefore enabling us to compare the two techniques.

[2]The width could also be determined for different receiver bandwidths, however, during the October 2002 experiment, data was only recorded at one fixed bandwidth due to time and equipment constraints.

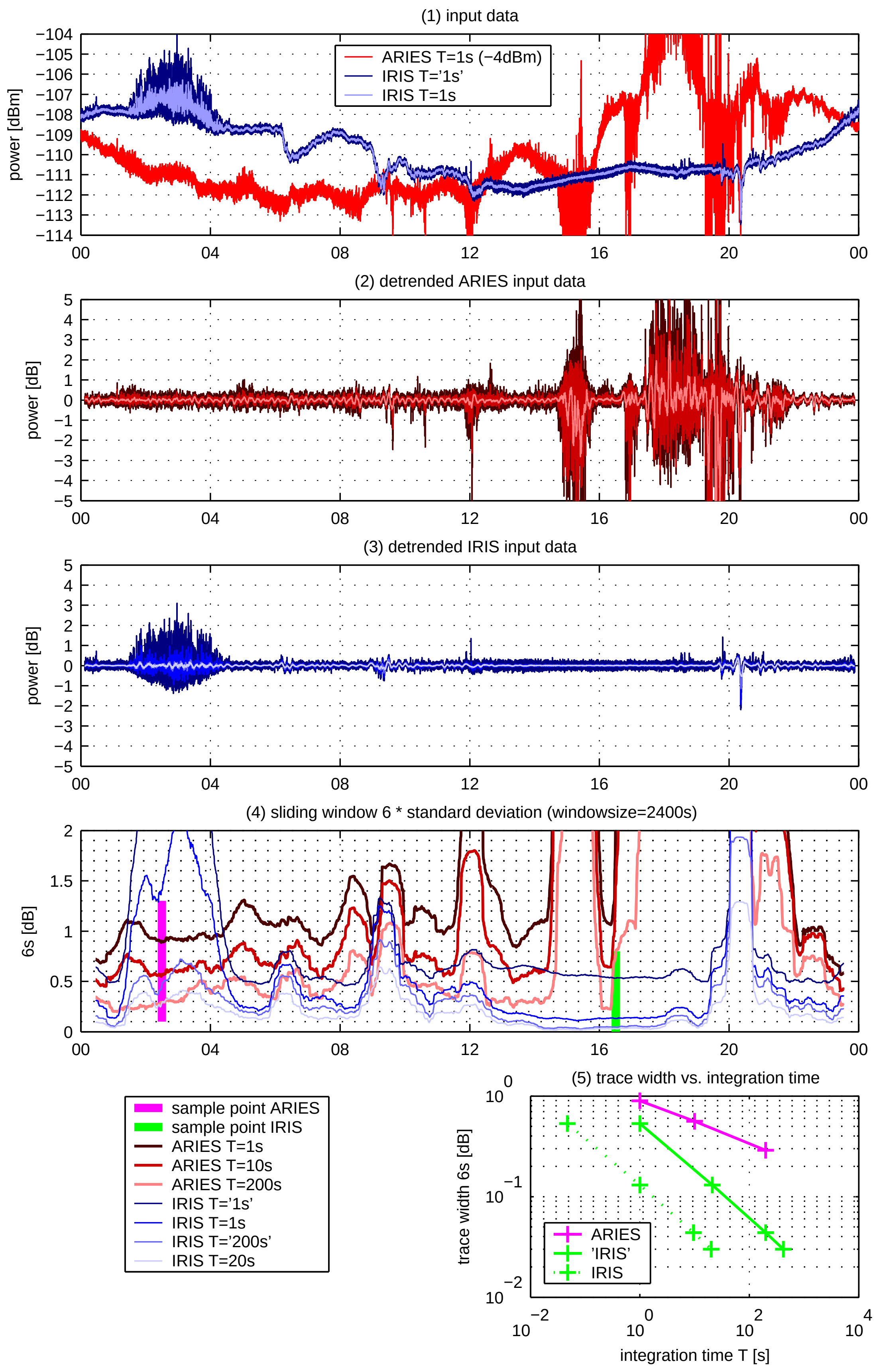


Figure 9.3: Beam width versus integration time. ARIES worst case beam versus IRIS beam 9 for 2002-10-30.

Figure 9.3 shows some results for 2002-10-30. The top panel shows the two basic input datasets: ARIES data from the 'worst-case' beam 595 (see chapter 7), integrated to 1s integration time and IRIS data from IRIS beam 9, with the highest possible 'resolution' of '1s' (which has in effect an integration time of only 47ms, see section 9.4.2.1 below). The panel also shows the same IRIS data post-integrated to a physically correct integration time of 1s by averaging over $\frac{1000\text{ms}}{47.5\text{ms}} \approx 21$ source samples.

Panels 2 and 3 show the same data post-integrated to different integration times, for ARIES and IRIS data, respectively. Also, the data has been detrended by subtracting a 900s average of the respective dataset from the raw data. Detrending makes it easier to visually compare the trace widths.

Panel 2 shows ARIES data post-integrated to integration times of 1s, 10s and 200s respectively. Panel 3 shows IRIS data at the original physical integration time of 47ms (designated "T='1s'") and post-integrated IRIS data for physical integration times of 1s (designated "T=1s", generated by averaging 21 samples), 9.5s (designated "T='200s'", generated by averaging 200 samples) and 20s (designated "T=20s", generated by averaging 420 samples).

Panel 4 summarises the information from panels 2 and 3. It shows the standard deviation ($\times 6$) for each of the curves in panels 2 and 3, calculated using a sliding window mechanism with a window size of 2400s. Therefore, we now have measurements for the trace widths at every moment in time.

Note the two vertical lines ('sample point ARIES' and 'sample point IRIS'). The values at the intersection of these lines with the ARIES respectively IRIS trace width curves are used to produce the diagram in panel 5. These sample points were positioned manually at what seems to be a reasonably quiet time (no scintillation, no absorption) for the respective dataset.

Panel 5 shows the trace width of the data over the respective integration time on a double logarithmic scale. The magenta line shows the relationship between integration time and trace width for ARIES data. The green lines show the relationship between integration time and trace width for IRIS data, where the dotted green line uses the 'physically correct' integration times as described in section 9.4.2.1, whereas the solid green line uses the integration times as commonly referred to in MIA.

9.4.2.1 Note on How to Interpret the Term ‘1s Data’ for an IRIS Type Riometer

There are different possibilities of interpreting IRIS data: What we commonly refer to as 1s data is really data that has only been integrated for far less than 1s due to the working principle of the IRIS system: The 49 outputs from the phasing network are fed into only 7 receivers. This is achieved by time-division switching, each receiver being fed by one of 7 columns of 7 beams. The power level for each beam is recorded once per second, and switching is arranged so that each second is divided into 8 time slots, the 7 beams being connected for 125ms in turn [BHH95].

The IRIS manual [DR94, p. 10] describes that the sampling process also involves a 25ms pause before the output of the A/D converter is integrated for 95ms, followed by another 5ms pause. The effective integration time for the analogue part of the system (the La Jolla Riometer, [La 74]) was adjusted to 6ms [DR94, p. 10], which is much smaller than the 95ms during which the signal is integrated in the A/D conversion process. Therefore this larger time constant of 95ms dominates, which would result in an effective integration time in the order of 95ms.

The IRIS riometer employs a noise-balancing technique, where an internal noise source is constantly adjusted to match the power of the received signal, thereby making the measurement independent of receiver gain changes. This is achieved by continuously switching between receiving the external signal and receiving the signal from the internal noise source with a frequency as high as 583Hz and a duty-cycle of 50% [La 74]. Therefore the effective integration time of IRIS (the time that we are actually looking at the signal) is normally considered to be somewhere in the region of $95/2 = 47.5$ms, and this is what we will use for the following discussions.

One implication of these findings is that, in order to obtain IRIS data with an effective integration time of 1s, we need to post-integrate $\frac{1000\text{ms}}{47.5\text{ms}} \approx 21$ so-called IRIS 1s data samples. This technique will not give very accurate results when used with rapidly (i.e. with time constants smaller than 21s) fluctuating input signals, as the 21 post-integrated samples are spaced approximately 1s apart in time and will therefore not cover 1s of input data but 21s. However, for relatively constant input signals where the sole purpose of integrating is to reduce the noise, it will not make any difference whether the data to be integrated was contiguous or not.

9.4.2.2 Note on Post-integration Techniques for Complex Samples

Raw ARIES data as recorded by the test system used during the October 2002 experiment consists of complex samples. Obviously, post-integrating (averaging) these raw samples is not the same as post-integrating (averaging) the absolute values of these samples. See figure 9.4 (the figure is for a post-integration time of $400 \times 0.5\text{s} = 200\text{s}$). In fact, the only correct way is to post-integrate the original complex samples. This is also how the raw data samples from the A/D converter are integrated in the first place. This means that we cannot easily convert, say, ARIES data that has been (post-)integrated for 10s to data with an integration time of, say, 60s. Instead, we always have to start from the original, complex samples, which in this case (the October 2002 experiment) are available with an integration time of 0.5s for most of the experiment's duration.

9.4.3 Conclusion

In conclusion, regarding the trace width as a measurement for the precision of the data, table 9.2 shows the accuracy that we can achieve with IRIS/ARIES for different integration times. This is the textual representation of the bottom panel in figure 9.3.

9.5 Relative Noise Intensity for Different ARIES Beams

Of course, the same considerations as in section 9.4 can also be used to compare two different ARIES beams. In this section we will compare the zenithal 16+16 ARIES beam to the worst-case ARIES beam 595.

The results can be found in figure 9.5. The zenithal beam seems to be worse than beam 595. The table in section 9.4.3 incorporates these results. It turns out that beam 595 is not actually the 'worst-case' beam as far as noise is concerned. Comparisons between all pencil beams were not possible within the constraints of the 2002 experiment setup, and are therefore not covered in this thesis.

9.6 The Dynamic Range of the Receivers

One concern that was raised is that it may well be that because we were operating at the very low end of the dynamic range of the receivers, we cannot record any form of absorption because

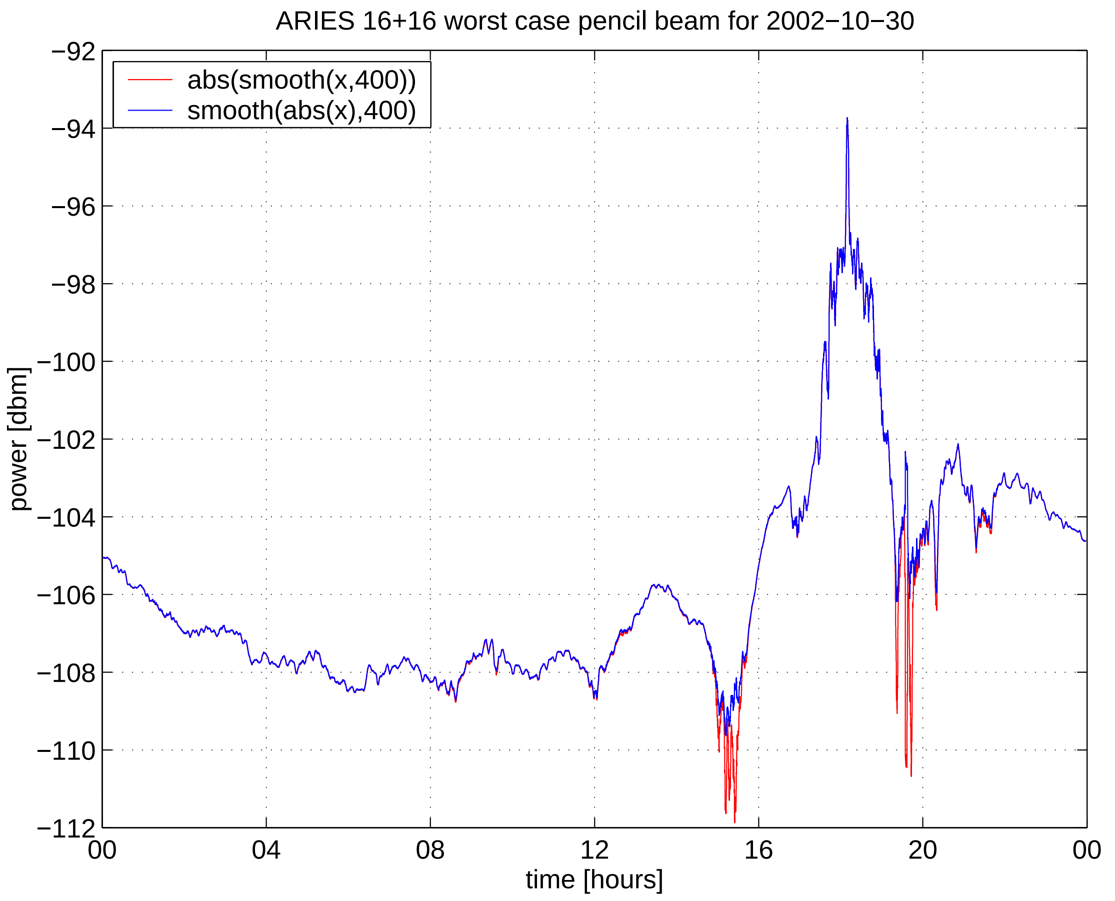


Figure 9.4: Difference between post-integrating (complex) raw data and post-integrating pre-processed data

	T=1s	T=10s	T=21s	T=200s	T=420s
IRIS	0.53dB	*0.20dB*	0.13dB	0.04dB	0.03dB
ARIES beam 595	0.90dB	0.56dB	*0.45dB*	0.29dB	*0.23dB*
ARIES zenithal beam	3.38dB	2.19dB		1.23dB	

Table 9.2: Achievable accuracy for different integration times. Figures in *italics* are interpolated respectively extrapolated from the graph in figure 9.3, panel 5.

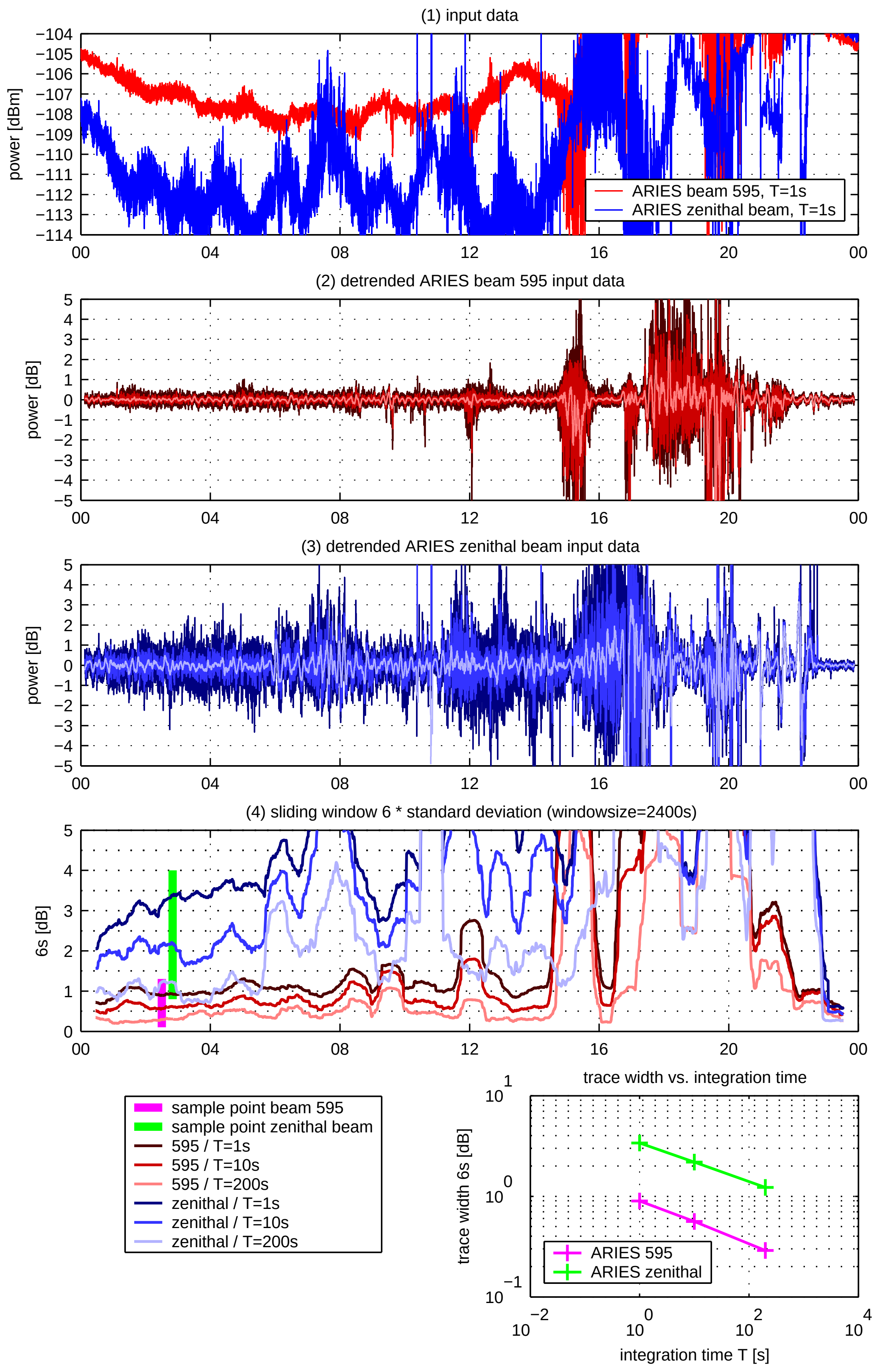


Figure 9.5: Beam width versus integration time (zenithal beam). ARIES beam 595 (2002-10-30) versus ARIES zenithal 16+16 beam (2002-10-27).

further attenuation of the incoming signal will get lost in the noise floor of the receivers.

Figure 9.6 shows calibration curves for the two receivers as recorded 2002-10-25, 15:44UT. Those curves were recorded by manually stepping through a 60dB input power range in -1dB steps, one step every 15 seconds. The input power was generated by a noise source followed by the manually operated attenuator, see the bottom path in figure 9.1. The x-axis is linear time, the y-axis in the top panel is in arbitrary linear power units as output by the integrator (T=1s) of the respective receiver. The y-axis in the bottom panel is in arbitrary dB units.

To the right of the calibration curves, we show two sets of recorded data for each receiver, the diagrams share the same y-axis scaling. We can see that, when looking at the logarithmically scaled plots, we are always operating in the lower half of the total dynamic range of the receivers.

9.6.1 Conclusion (Dynamic Range)

Even though we are operating at the bottom end of the dynamic range of the receivers, it seems we are not too close to or even below the noise floor. However, as can be seen clearly, especially in the top panel in figure 9.6, we certainly do not make use of the full available resolution of the A/D converter (12bit). For the final system, care will be taken to exploit the full A/D range.

9.7 Influence of the Radio Stars Alone

The readings from the experimental cross-correlated ARIES beams contain features that, at the time, were not easily explainable. This section will compare these readings to the simulated influence of only the strongest two radio stars, Cassiopeia and Cygnus. This is to show how much the recordings are influenced by these stars alone.

9.7.1 The Radio Stars

Figure 9.7 shows the ARIES 16+16 worst-case beam (beam 3501) recordings for 2002-10-30, together with the simulated influence of Cassiopeia alone, Cygnus alone and Cassiopeia and Cygnus together. Also, the simulated QDC is shown. These simulations were performed using the RIOSIM package as developed by the author and described in chapters 6 and 7. Please note that the absolute dBm-values in figure 9.7 are not too meaningful, as calibration is arbitrary at this stage.

We can observe the following facts:

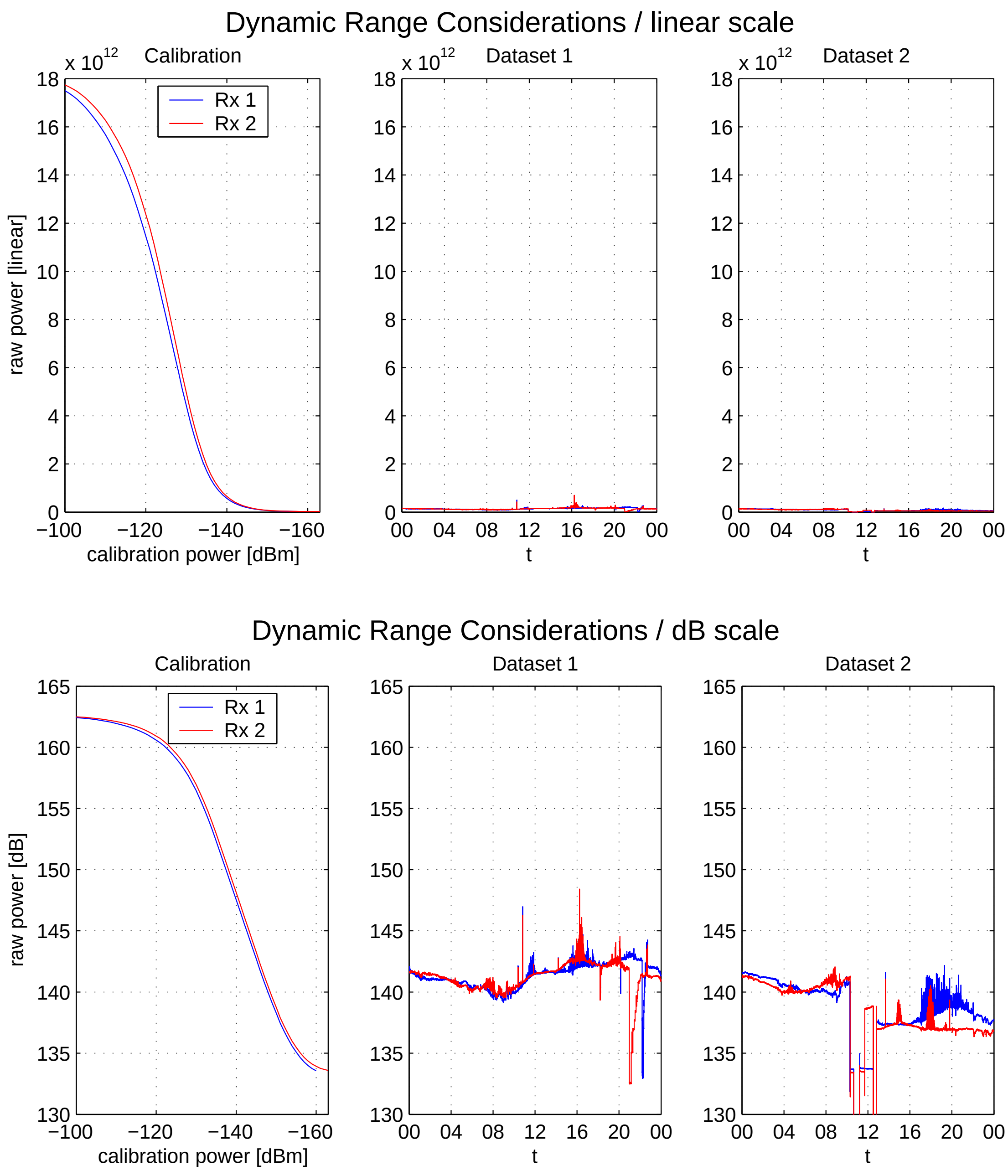


Figure 9.6: Receiver working range and signal dynamic range

- The simulated QDC shows a steady increase as we get nearer to the peak (18:00UT), then the simulated QDC values decrease
- The recorded data follows the same general trend.
- The peak due to Cassiopeia in the main lobe is clearly visible in the recorded data.
- Where the radio star simulations predict a strong influence of Cassiopeia and/or Cygnus on the beam, we often observe a significant *decrease* in received power.
- This suggests a phase difference between the signals from the two fan beams, due to reception in different sidelobes or in one main lobe and one sidelobe, which *reduces* the result of the cross-correlation.
- The most prominent example is around 15:00UT, where Cygnus is in the main lobe of the NS fan beam and in the second sidelobe of the EW fan beam. See the following section 9.7.2 for a more detailed discussion of this phenomenon.

9.7.2 Phase Considerations

With the untapered ARIES array as used for the October 2002 experiment, we get very strong sidelobes in the two linear arrays, and therefore even more so in the pencil beam, for an explanation see chapter 2, section 2.4.3.1. The resulting signal is a kind of 'weighted sum' of the incoming signals from all lobes. In other words, it is generally impossible to tell what exactly was going on in the main lobe, because even though the signals received from the main lobe are, of course, part of the resulting signal, all the signals coming from the sidelobes are by no means negligible and, if strong enough as a whole, they can even obscure (i.e. influence in positive and negative direction) the signal from the main lobe completely.

However, when we have a very strong point source moving along a known path, the influence of that point source is likely to dominate the signal from the beam in question. With ARIES, we do have two such prominent point sources, Cassiopeia and Cygnus, the two radio stars. Figures 9.8 and 9.9 show the traces of the two radio stars (Cygnus is the outer circle, Cassiopeia the inner circle) projected onto the ionosphere at 90km height. Also shown in these figures are the -3dB, -6dB, -9dB,... outlines of the two ARIES 16+16 fan beams (figure 9.8) that together form the 16+16 worst-case pencil beam. Figure 9.9 shows this pencil beam. The labels on the star traces are in UT for 2002-10-30.

We find that for the time in question, around 15:00UT to 16:00UT, Cygnus passes through the main lobe of the NS fan beam (2503), see figure 9.8 panel (b). At the same time, as we can see in panel (a), Cygnus passes through the second sidelobe of the EW fan beam (2502). Figure 9.10 shows the amplitude and phase response of a linear phased array of 16 elements, as is used to create the ARIES fan beams in this experiment. One finds that the average phase for a signal coming from the sidelobes is offset by $\pm 90^\circ$ compared to the average phase in the main lobe. For point sources, this phase difference can therefore amount to 180° and more, depending on the exact position of the point source relative to the sidelobes.

In the case of these particular two fan beams, we receive the same strong signal originating from Cygnus, but with a phase difference of somewhere between 90 and 180 degrees. Because this strong signal dominates the received signal in spite of all the other sidelobes, the complex correlation between the signals from the two fan beams will represent this phase shift. We can clearly observe this fact in figure 9.7 (15:30UT). As the star passes through this particular position, the complex result of the cross-correlation picks up the phase difference resulting in a very much reduced final result.

Similar observations can be made for other times, the other most noticeable event in the dataset presented is probably around 8:30UT, where the real component of the cross-correlated signal again gets very much reduced, whereas the absolute value remains approximately constant. This indicates a phase shift of around 90°.

9.7.3 Conclusion (Influence of Radio Stars)

- Strong signals from the sidelobes modify the cross-correlated signal from the pencil beam.
- Because the signals from the sidelobes have a relative phase difference, the cross-correlated signal may get significantly *reduced*. This *reduction* in signal is an effect specific to the Mills Cross due to the cross-correlation stage. It does not happen in IRIS type filled array systems.
- So far, the only way of avoiding this seems to reduce the sidelobes significantly *before* the cross-correlation takes place, i.e. by tapering the two linear arrays. This is in accordance with Nielsen's findings in the original report [Nie01]. We will discuss a more permissible post-processing (interpolation) approach in chapter 10.
- Therefore, the reduction in signal, which may at first glance be conceived as inexplicable,

actually proves that the cross-correlation approach is working as expected.

9.8 A Comparison of IRIS Pencil Beams to ARIES Pencil Beams for Several Days

Another issue that was raised is that the ARIES recordings seem to vary quite severely from day to day. In order to quantise this observation, figure 9.11 compares ARIES recordings for the period from 2002-10-28 till 2002-11-03. During this time, ARIES was configured for beam 3501, the 16+16 pencil beam pointing into the 'worst-case' direction. The dataset for 2002-10-29 contains some faulty data because of ongoing experiments at the ARIES site during that day (see experiment log for details). That is why the green lines in figure 9.11 should not be taken too seriously.

The top panel in figure 9.11 shows the recordings from ARIES, post-integrated to an integration time T=400s, taking the absolute value of the resulting complex samples.

The middle panel shows again the recordings from ARIES, post-integrated to an integration time T=400s, but this time only the real part of the complex result is plotted.

The bottom panel in figure 9.11 shows IRIS beam 9 data, again post-integrated to T=400s for the same period.

9.8.1 Conclusion

What at first glance seems a very high variation from day to day seems to come from the generally very disturbed conditions during the period in question. This can be derived from the fact that we also see significant variation in IRIS pencil beam data for this period, although not as high as in ARIES data, which was again expected since the ARIES pencil beam in question is twice as narrow as the IRIS one, therefore picking up finer structures and showing larger variations.

Long after the experiment had finished, it was also realised that the prototype receivers are very sensitive to (ambient) temperature changes, resulting in both gain and offset drifts. This will also affect the October 2002 recordings to some extent. No independent temperature data was collected.

9.9 Effect of Height Variation on Beam Intersection with IRIS (80–100km)

Most of the beam contour maps throughout this thesis were produced by intersecting the beam pattern in question with a spherical ionosphere at a height of 90km above the Earth's surface (see chapters 3, 6 and 7). When determining, which IRIS beam(s) are looking at the same piece of ionosphere as a given ARIES beam, this height of 90km was used for the calculations. In this section we examine how the projection height influences the locations of the projected beam outlines.

Figure 9.12 shows ARIES and IRIS beam projections for three different projection heights. Panels (a), (c) and (e) show the whole field of view for the two systems at 80km, 90km and 100km projection heights, respectively. Panels (b), (d) and (f) on the right-hand side show a magnified view around ARIES beam 3501 recorded during the October 2002 experiment around 2002-10-30. Beam 3501 is the big yellow contour, the small yellow contour is beam 3001, a 32+32 pencil beam pointing in the same direction as the recorded beam 3501. This contour is included for comparison purposes. The magenta contours are IRIS beams 9 and 10, from left to right.

9.9.1 Conclusion

- Changing the projection height changes the relative positions of ARIES and IRIS beam contours.
- For reasonable projection heights between 80km and 100km, the changes are rather small.
- Figure 9.12 shows that ARIES beam 3501 is always closely co-located with IRIS beam 10, as far as only the main lobe is concerned.
- Therefore, it makes sense to compare, say, recorded ARIES beam 3501 data to IRIS beam 10 data for the same time, no matter what the exact height of the absorption event might have been.
- We must keep in mind that ARIES beam 3501 as recorded has very strong sidelobes that lead to effects as described especially in preceding sections 9.7 and 9.8.

9.10 Summary

The first ARIES experiment has proven that the Mills Cross (cross-correlation) technique allows increased spatial resolution — even for the same number of antennas used — compared to a filled array riometer. However, as predicted, at the same time it leads to an increased noise level in the measurements with adverse effect for the minimum integration time. For filled array riometers the integration time can be as low as $\frac{1}{8}$s; for a correlation system the integration time will need to be at least some seconds to achieve comparable uncertainties. This agrees with the simulations that were initially carried out by the author and others (chapter 5). The measurements also indicated that antenna sidelobes introduce phase delays that can result in both signal reduction and increase especially in the presence of a strong noise source (radio star).

The experiment showed the need to suppress the sidelobes of such a system to a level even below the one determined by simulations and previous theoretical calculations due to the sensitivity of the Mills Cross to phasing differences in the signals coming from the two arms of the cross. This is achievable with appropriate tapering functions (see chapter 2). In addition, adaptive beam steering will be able to mask the influence of the strongest sources of interference, the radio stars, see the suggestions for future developments in chapter 11. In any case, this first experiment has proven that, together with advanced high-level processing software, a riometer based on the Mills Cross technique will be able to image absorption with sufficient temporal and unprecedented spatial resolution.

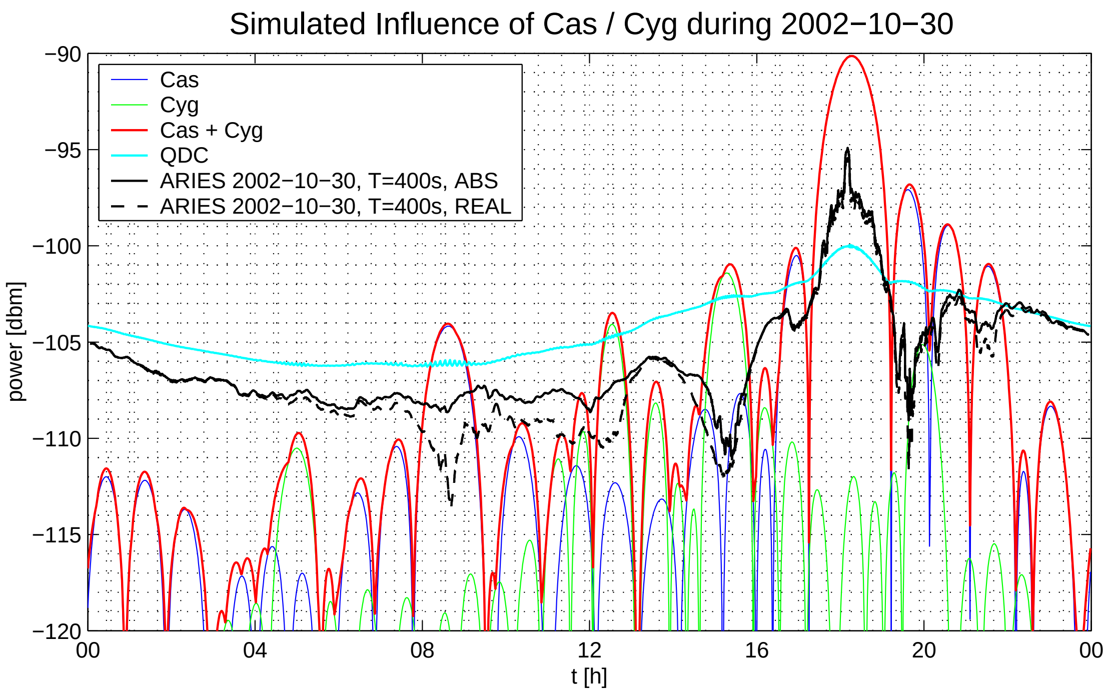
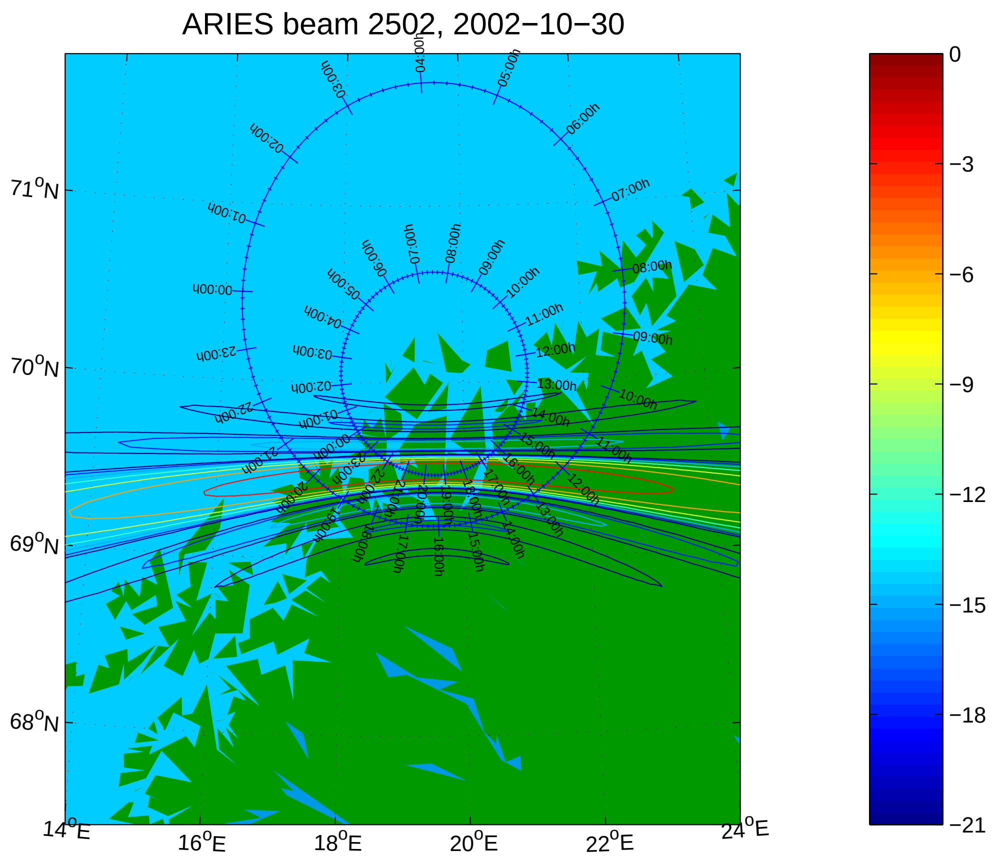
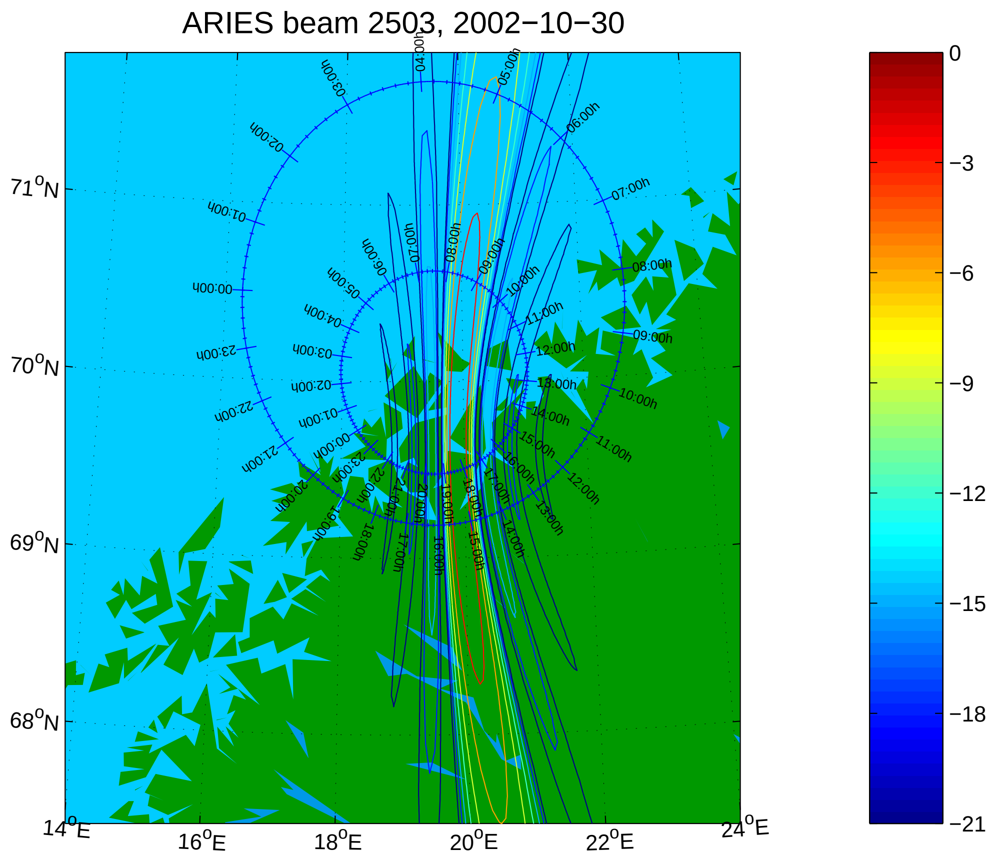


Figure 9.7: Influence of the strong radio stars



(a)



(b)

Figure 9.8: The radio stars' influence on beam 3501. Part A: fan beams

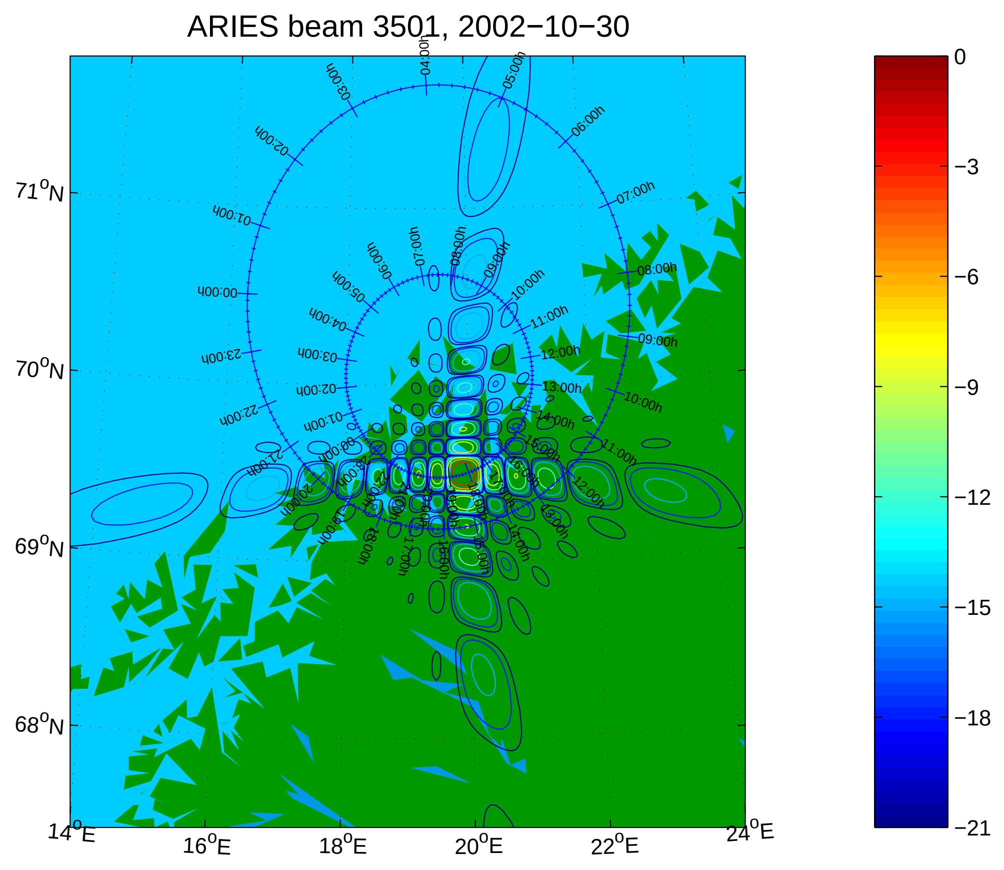


Figure 9.9: The radio stars' influence on beam 3501. Part B: pencil beam

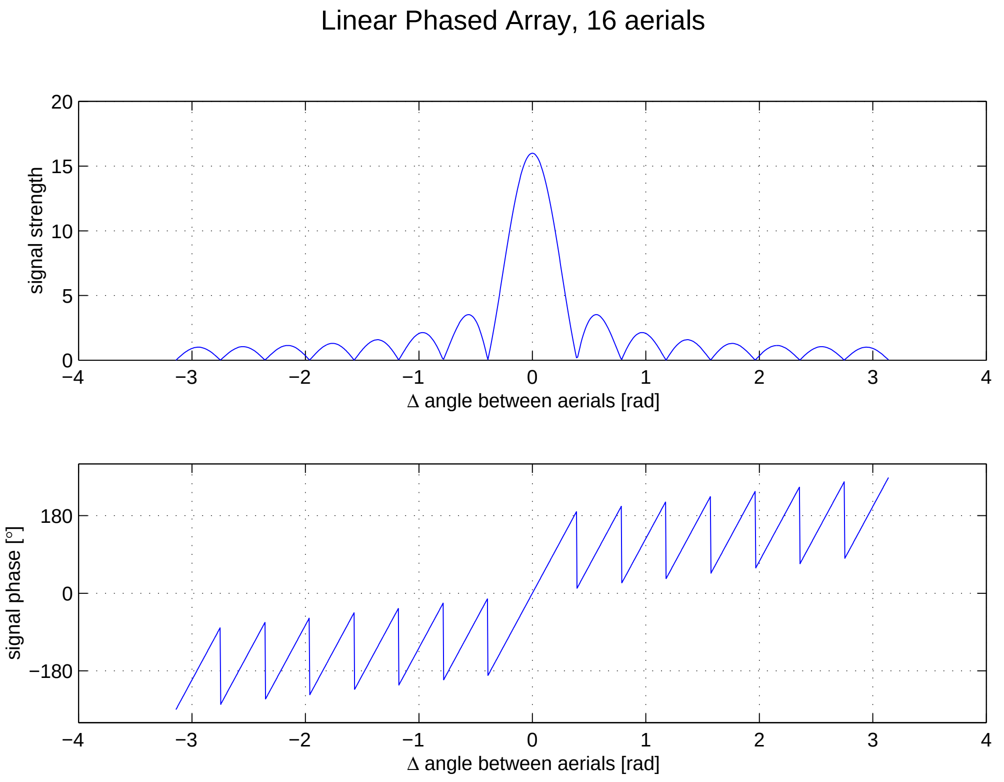


Figure 9.10: Amplitude and phase response of a 16-element linear phased array

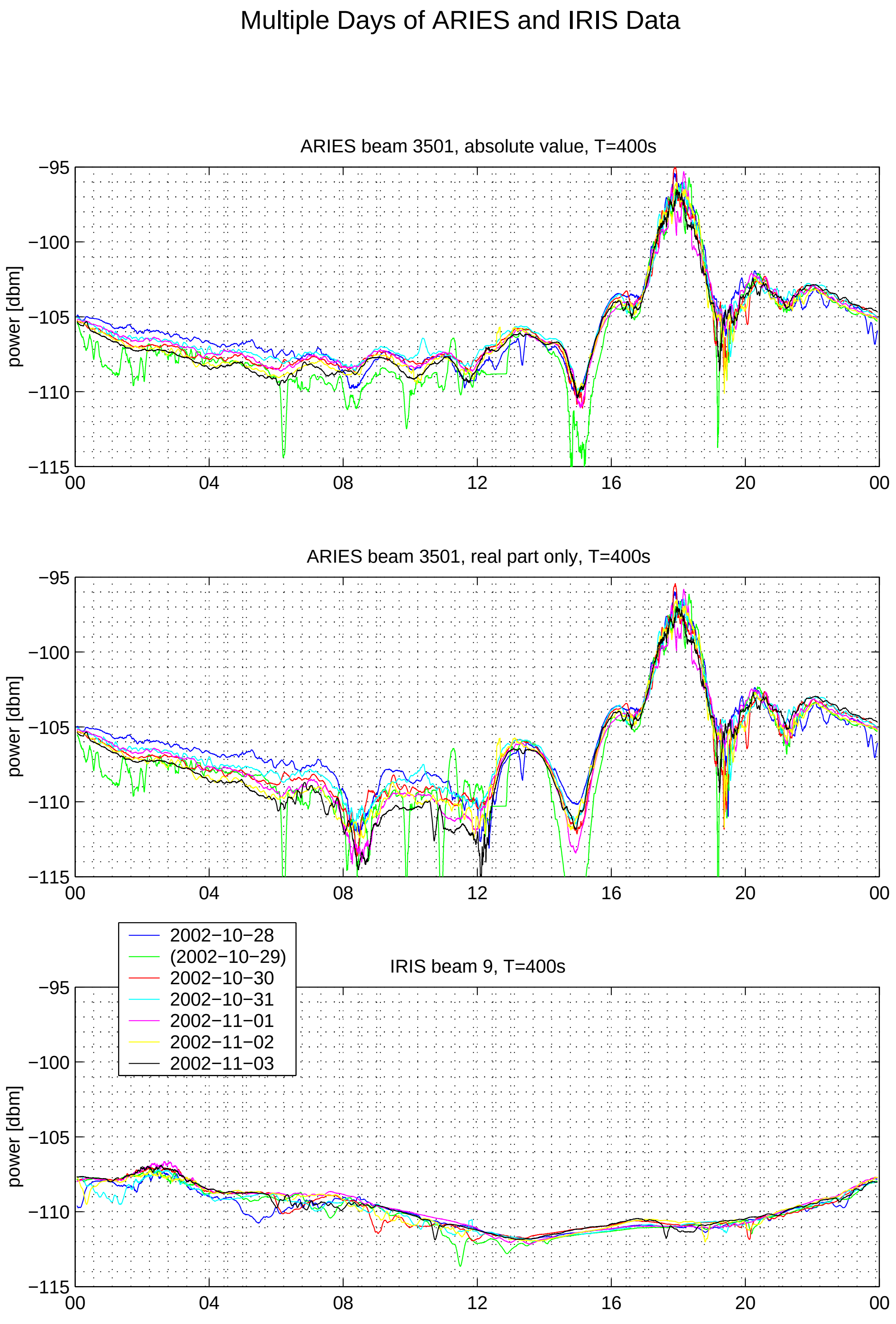
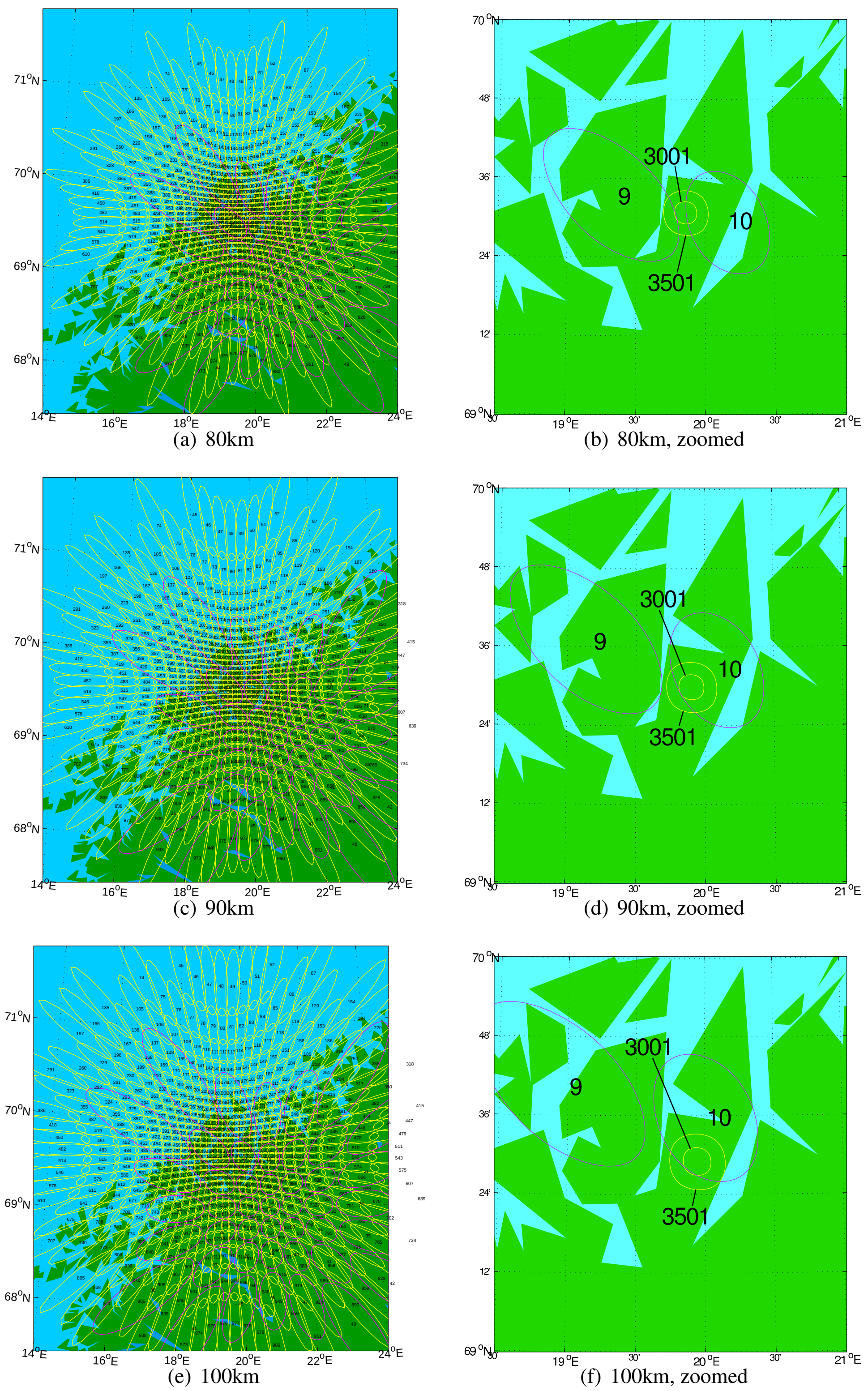


Figure 9.11: Pencil beam comparison for multiple days



(a) 80km (b) 80km, zoomed

(c) 90km (d) 90km, zoomed

(e) 100km (f) 100km, zoomed

Figure 9.12: ARIES / IRIS beam projections onto different heights

Chapter 10

A New Approach to Image Interpolation in Riometry

In this chapter, we present data recorded by ARIES, and how phasing issues affect the image output. We develop a metric for quantitatively comparing the quality of different image interpolation methods and apply this metric to various simulations. The traditional approach for riometer image interpolation (as used by IRIS) is presented, and its drawbacks are pointed out. We suggested two new approaches to riometer image interpolation in [GSH05], mainly using IRIS data due to the fact that no longer periods of complete (i.e. all-beam) ARIES data were yet available. This is now no longer the case, and in this chapter we will mainly present results using real ARIES data and simulations of the ARIES system. Before doing this, however, sections 10.1 and 10.2 below contain some more information on motivation, the currently used approach to image interpolation, the new algorithm and some additional background information.

10.1 Motivation

Sidelobes often generate additional complications in riometry. Especially in standard phased array-based imaging riometers, the first sidelobes of any given beam are never below a level of -13dB from the main lobe, see chapter 2. Especially during periods of high solar activity, strong bursts of radio noise will be received through sidelobes and falsely classified as coming from the pointing direction of the main lobe of the corresponding beam. Scintillation seen by a sidelobe will affect the main beam reading, even though the main beam might be pointing in a direction entirely free of scintillating strong radio sources.

This problem is not as severe in multiplicative array type riometers, since the sidelobes need to be reduced considerably even before the cross-correlation stage in order to prevent errors due to phase differences, see the initial investigations and analyses in chapter 9.

Furthermore, however, the so-called 'image data' from fixed beam instruments still consists of a finite number of data points per 'image,' 49 in the case of IRIS type riometers. This requires interpolation between the available data points to come up with a complete image, and leads to the question which interpolation method is most appropriate, given that the data points do not really represent (power) values at certain discrete directions but are instead integrated values over the whole sky, convolved with the respective beam pattern.

Since the general shape of each beam (beam pattern) can be calculated theoretically, and has been found to be consistent with actual observations[1], see for example figure 9.7 in chapter 9, we can try to use this additional knowledge to derive a more accurate representation of the spatial distribution of the received noise power, therefore no longer relying on the simpler interpolation methods.

We will describe a new approach, GLEAM, in section 10.3, including some notes on its actual implementation in MATLAB. This will be followed up by several case studies to evaluate its performance using both real and simulated data.

10.2 Prerequisites

In this section we introduce some general facts that are used for the observations in the following sections. We explain the role that obliquity factors (do not) play for the observations. Many plots in this chapter use the FLATM projection method as introduced in chapter 6, see especially figure 6.11.

10.2.1 The Need for Image Interpolation

As already repeatedly mentioned in earlier chapters, see for example figure 7.1 in chapter 7, imaging riometers are still based around a set of beams pointing in different directions. The more beams, the finer the level of detail that can be resolved. For data analysis, the raw beam data (a time series of power values for each beam) needs to be interpolated onto a (usually regularly

[1]Beam patterns can also be measured experimentally, though in case of riometers and to the author's knowledge this has not been done. One approach would be to use strong radio stars as known noise sources, though of course these stars cannot be moved around freely, so one would have to be content with the natural diurnal variation of their location(s). This is one of the suggestions for future work presented in chapter 11.

spaced) grid, an image. This image is a spatial representation of the observed area. Along with the imaging capabilities of riometers came the necessity to spatially interpolate between the measurements in order to form a real image of the observed area. Time series of these images can then be used for further studies like analysing the motion of absorption patches [MHMH04] and serve as a source for more advanced diagrams such as keograms, movies and virtual beams, which allow a closer look at the spatial distribution of structures and motion. These images can also be directly compared to images from other imaging instruments, for example optical cameras [dPKH02].

A major feature of interpolated images is that they enable, to a certain extent, the abstraction from beam data to data for arbitrary pointing directions of interest, sometimes called 'virtual beams.'

We will introduce the traditionally used interpolation method below, and then proceed to present and evaluate a new, alternative, approach. As an introduction, figure 10.1 compares real IRIS and real ARIES data for one specific moment in time. Each dataset is shown as a simple square matrix plot giving the instantaneous power values for each beam. Note how ARIES data forms more of an image simply due to the fact that there are more data values per unit area. See also the contour plot comparing ARIES and IRIS beam contours in chapter 7 (figure 7.1). Nevertheless, these plots still do not directly represent sky brightness, they are beam power values. We will come back to this issue, and how GLEAM improves on this, below. We will present both 'traditional style' interpolated images and GLEAM-based interpolated images further down.

10.2.2 Traditional IRIS Interpolation Algorithm

For current IRIS type systems, riometer absorption images are usually created by interpolating between absorption values for individual beams. The locations of the beam centres serve as grid points for subsequent linear interpolation. This technique generally produces good results. However, the fact that the actual shape of the imaging beams is not considered, potentially introduces errors and can lead to misinterpretations. In particular, any given imaging beam receives signals not from one direction but from a range of directions around the beam centre, depending on the beamwidth, which itself is inversely proportional to the extents of the receiving antenna (see chapter 2). Also, a not always negligible fraction of signal is received from sidelobes that point in a significantly different direction to that of the main beam.

The methods proposed in this chapter are inherently different in that they take the shape of the receiving beams into account. In doing so, they have the potential to compensate for the effects of sidelobes, to overcome the spatial constraints of linear interpolation (thus extending the field of view) and to uncover features that may not show up in traditionally interpolated images.

IRIS power/absorption images as created by the Multi-Instrument Analysis Toolkit (MIA) [MH04] use a projection similar to the FLATM projection to map the location of the 49 beam *centres* onto a flat two-dimensional grid[2]. MIA assumes that the recorded power/absorption values originate from the respective beam centre (the direction of maximum gain, also referred to as beam axis or boresight) and then uses linear interpolation to fill the space between the beam centres. An example of this can be seen in figure 10.2 (left panel), together with the triangles that are used internally by the two-dimensional linear interpolation algorithm. The vertices of the triangles coincide with the 49 beam centres. As the four corner beams (1, 7, 43, 49) have significant sidelobes [AGW72] and the assumption of all power being concentrated at the beam centre does not even approximately hold, these beams are usually ignored, leading to an interpolated image as depicted in figure 10.2 (right panel). Therefore the useable working area for this algorithm is defined by the convex envelope of the beam centres projected onto a flat grid by the FLATM projection.

Apart from the general inaccuracy stemming from the ‘all power is coming from the main beam pointing direction’ assumption, one other major drawback of this algorithm can immediately be seen in figure 10.2. Consider the bright (red) horizontal band that is visible in both panels. This is our galaxy, and it extends far beyond the instrument’s field of view. However, in both the left (49 beam) and the right (45 beam) case, the bright band seems to terminate well within the field of view. This is due to the fact that the linear interpolation algorithm fills the space in question by interpolating along a line between the two outermost beams, and there are no suitable beams closer to the centre to set the picture right. Similarly, any observed absorption patch moving into the field of view will appear much weaker — or cut off — in the interpolated image until it makes its way into the central area. To avoid this, the nominal field of view has to be reduced further so as to only cover the densely beam-populated area in the central part of the figure.

[2]Recent versions of MIA now use a default grid based on geographic latitude and longitude for interpolation, as geographic coordinates are more universally useful and accepted. This does not change the basic interpolation algorithm, however. We will stick with the FLATM projection in this paper, because it does not introduce any asymmetric distortion effects and is independent of instrument location.

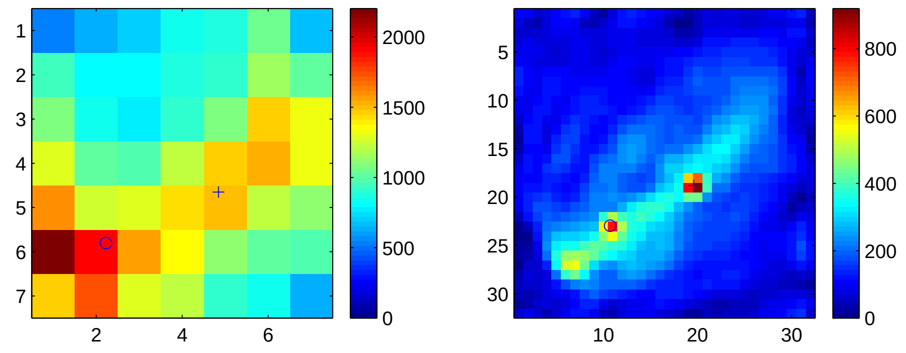


Figure 10.1: Non-interpolated IRIS (left) and ARIES (right) data for 2007-03-20 08:45. Also shown are the positions of the two strongest radio stars, Cassiopeia (+) and Cygnus (o), for the respective instrument locations as calculated by RIOSIM. Colour scales are in raw linear power units.

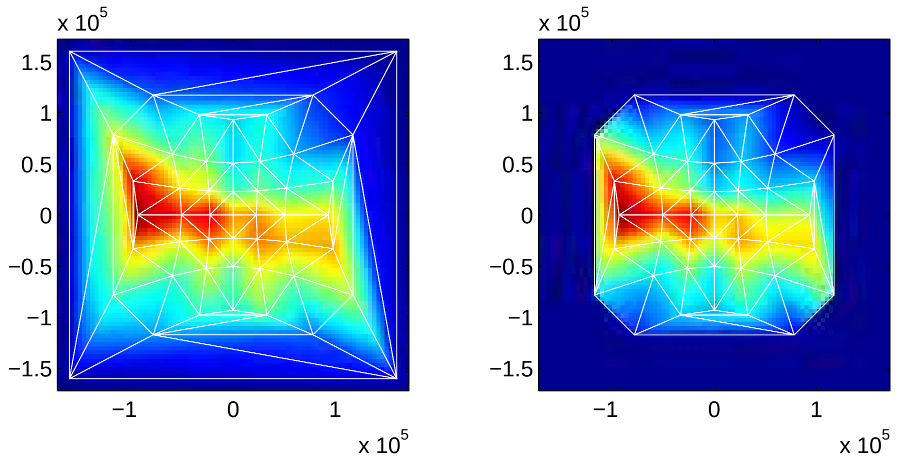


Figure 10.2: 'Traditional' IRIS image interpolation. Distances in metres. Colour represents arbitrary linear power units. Left panel shows Delaunay triangulation for all 49 beams, right panel shows Delaunay triangulation for only the 'good' beams, i.e. all IRIS beams except for the four corner beams 1, 7, 43 and 49.

10.2.3 Role of Obliquity Factors

The interpolation algorithms as discussed in this chapter all deal with interpolation of power data on the positive hemisphere seen by the receiving instrument. The data is not interpreted in any way prior to processing. In particular, no assumptions are implied as to the media that the incoming signals traversed prior to reception, i.e. no correction factors (in this case known as obliquity factors) for the varying observed thickness of the absorbing layer etc. are applied to the data. When comparing actual signals to theoretical signals based on convolution of beam patterns and sky map (as can be done with RIOSIM, see chapters 6 and 7), obliquity factors need to be taken into account as soon as there is an absorbing layer of electrons present. This layer appears thicker with decreasing elevation angles. See figure 10.3: The apparent thickness d' of the absorption layer decreases with increasing observation elevation angle ϕ. The law of sines for $\triangle ABC$ allows us to derive angle β:

$$sin\beta(\phi) = \frac{sin(\phi + 90^\circ)}{r_e + h} \cdot r_e \tag{10.1}$$

from which we can derive the obliquity factor $\delta(\phi)$ simply by looking at $\triangle B'C'A'$ (as long as $d \ll (r_e + h)$):

$$\delta(\phi) = \frac{d'(\phi)}{d} = \frac{1}{cos\beta(\phi)} \tag{10.2}$$

The left panel in figure 10.3 shows a plot of $\delta(\phi)$ for elevation angles $0^\circ \leq \phi \leq 90^\circ$ and $h = 90$km (blue line). For comparison reasons, $\delta(\phi)$ is also shown for an (unrealistic) height $h = 1000$km (red line). This is essentially a 'lower amplitude' version of $\delta(\phi)$ for $h = 90$km. The dashed line is $\delta(\phi) = 1/cos(90^\circ - \phi)$ as used in [HD02]. This is an approximation of equation 10.2 that can be seen to work well for elevation angles $\phi > 30^\circ$.

Obliquity factors will also have to be taken into account when it comes to deriving absorption from the input data, but in this chapter we are solely dealing with (spatial) interpolation of the underlying raw received power data. Existing algorithms can then be used to derive quiet-day curves (QDCs) and absorption data for arbitrary directions ('virtual beams') within the usable working area of the given interpolation algorithm. See for example [BHH95] (IRIS), [DS90] (Density method), [KDR85] (Inflection Point method) and [MH07] (Percentile method, manuscript in preparation) and references therein for discussions of various approaches to generating QDCs. Note, however, that equation 10.2 only works well if there is actually a well-defined absorption layer present. For quiet-day absorption, equation 10.2 generally exaggerates the expected results. This is because quiet-day absorption happens over a large range of heights, not within one narrow layer.

Note that this use of obliquity factors is different from the traditional use, where obliquity factors are applied to received power values from each beam, either ignoring the beam shape or including the effects of the beam pattern in the 'effective' obliquity factor [HD02]. The obliquity factor in equation 10.2 does not relate to *beams* but simply to *viewing directions*.

10.2.4 Metric

To compare the performance of the various algorithms and parameter sets, the following metric is used. This rates reconstructed images according to how similar they are to the original (in case of results based on simulated reception), or how they compare to a pre-selected reference image (in case of images created from real data, in which case there is no 'original' image that can be used as a reference).

The metric is based around the square of the difference in brightness summed up for all directions and weighted by area to compensate for the distortions introduced by the spherical coordinate system.

$$m = \frac{1}{4\pi} \int_{\theta,\phi} (B(\theta,\phi) - B_{cur}(\theta,\phi))^2 \cos\phi \, d\theta d\phi \tag{10.3}$$

For comparing results of interpolation methods whose data products do not actually cover the whole hemisphere, we will simply zero out the non-applicable area (typically elevation angles below some threshold ϕ_0) in all datasets prior to calculating the similarity metric. We will refer to such values as $m_{el \geq \phi_0}$.

10.3 The Parametrised Model Interpolation Method (GLEAM)

We will first describe the general approach, this is not specific to using spherical harmonics, but is indeed valid for any kind of model that can be parametrised with a set number of parameters less than or equalling the number of concurrently available data points. The initial ideas behind GLEAM were conceived in discussions with Senior [Sen]. We will then proceed to evaluate the influence of various particular base models and evaluate their 'quality' with respect to the metric introduced in section 10.2.4.

Rather than starting with an unknown power distribution (which is of course what we will be doing later on), let us assume for a moment that we know the spatial power distribution P across the visible hemisphere, i.e.

$$P(\theta,\phi) = \text{known} \tag{10.4}$$

We also know the radiation pattern for each of the given instrument's N beams for all possible directions (θ,ϕ):

$$B_{1...N}(\theta,\phi) = \text{known} \tag{10.5}$$

Given the power distribution and the radiation patterns, we can now calculate the power response (the power received) $p_{n,simul}$ for each beam n. This is the same method that can also be used to derive theoretical quiet-day curves (as in chapter 7 or in [RDVvB91]):

$$p_{n,simul} = k \cdot \int_{\theta,\phi} P(\theta,\phi) B_n(\theta,\phi) cos\phi d\theta d\phi \tag{10.6}$$

k is a constant that can be used for calibration purposes. The N values $p_{n,simul}$ directly correspond to the received power as measured by the receivers for beams $1...n$.

If we now find a way of representing the power distribution P in equation 10.4 by means of $M \leq N$ parameters instead of an infinite number of discrete values, we can work our way backwards from the actual received power values and derive (an approximation of) the original power distribution P, denoted P_{model}.

Let us assume that P_{model} is a linear combination of M functions of (θ,ϕ), weighted by γ_m, i.e. a function of the direction as specified by (θ,ϕ) and of M parameters $\gamma_1...\gamma_M$ as follows:

$$P_{model}(\theta,\phi,\gamma_1...\gamma_M) = \gamma_1 \cdot f_1(\theta,\phi) + \gamma_2 \cdot f_2(\theta,\phi) + ... + \gamma_M \cdot f_M(\theta,\phi) \tag{10.7}$$

In order to determine the parameters $\gamma_1...\gamma_M$, we make use of equation 10.6, replacing the simulated results $p_{n,simul}$ with the actual measurement results p_n $(n = 1...N)$ from the N beams and the 'known' sky brightness distribution P (equation 10.4) with the modelled brightness distribution P_{model} (equation 10.7):

$$p_n = k \cdot \int_{\theta,\phi} B_n(\theta,\phi) P_{model}(\theta,\phi,\gamma_1...\gamma_M) cos\phi d\theta d\phi \tag{10.8}$$

We expand equation 10.8 using the definition of our model in equation 10.7:

$$p_n = k \cdot \int_{\theta,\phi} B_n(\theta,\phi) \cdot [\gamma_1 \cdot f_1(\theta,\phi) + \gamma_2 \cdot f_2(\theta,\phi) + ... + \gamma_M \cdot f_M(\theta,\phi)] cos\phi d\theta d\phi \tag{10.9}$$

Rearranging:

$$\begin{aligned} p_n &= \gamma_1 \cdot k \cdot \int_{\theta,\phi} B_n(\theta,\phi) f_1(\theta,\phi) cos\phi d\theta d\phi \\ &+ \gamma_2 \cdot k \cdot \int_{\theta,\phi} B_n(\theta,\phi) f_2(\theta,\phi) cos\phi d\theta d\phi \\ &+ ... \\ &+ \gamma_M \cdot k \cdot \int_{\theta,\phi} B_n(\theta,\phi) f_M(\theta,\phi) cos\phi d\theta d\phi \end{aligned} \tag{10.10}$$

Combining all constant terms into constants c:

$$p_n = c_{n,1}\gamma_1 + c_{n,2}\gamma_2 + ... + c_{n,M}\gamma_M \tag{10.11}$$

which can be written as one matrix equation for all N beams:

$$P = C \times \Gamma. \tag{10.12}$$

In other words, we get one linear equation with M unknowns $\{\gamma_1...\gamma_M\}$ for each of the N beam patterns. This set of equations can be solved (possibly in a least-squares sense for $N \neq M$) and therefore the unknown model parameters $\{\gamma_1...\gamma_M\}$ can be determined. Once these parameters are known, the (model) sky brightness can be calculated in any arbitrary direction (θ,ϕ) by using equation 10.7.

Figure 10.4 is a diagram of how real and simulated data, real, mapped and reconstructed brightness distributions, sky brightness distribution model and model coefficients are interrelated. In the following sections we will traverse this graph on various paths to demonstrate applicability and evaluate real-world behaviour of this method.

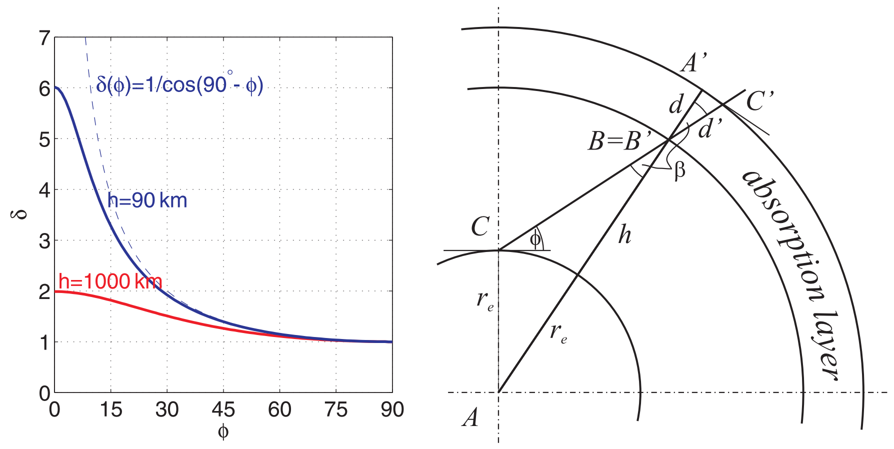


Figure 10.3: Obliquity factor δ correcting for apparent thickness of absorption layer (drawing not to scale)

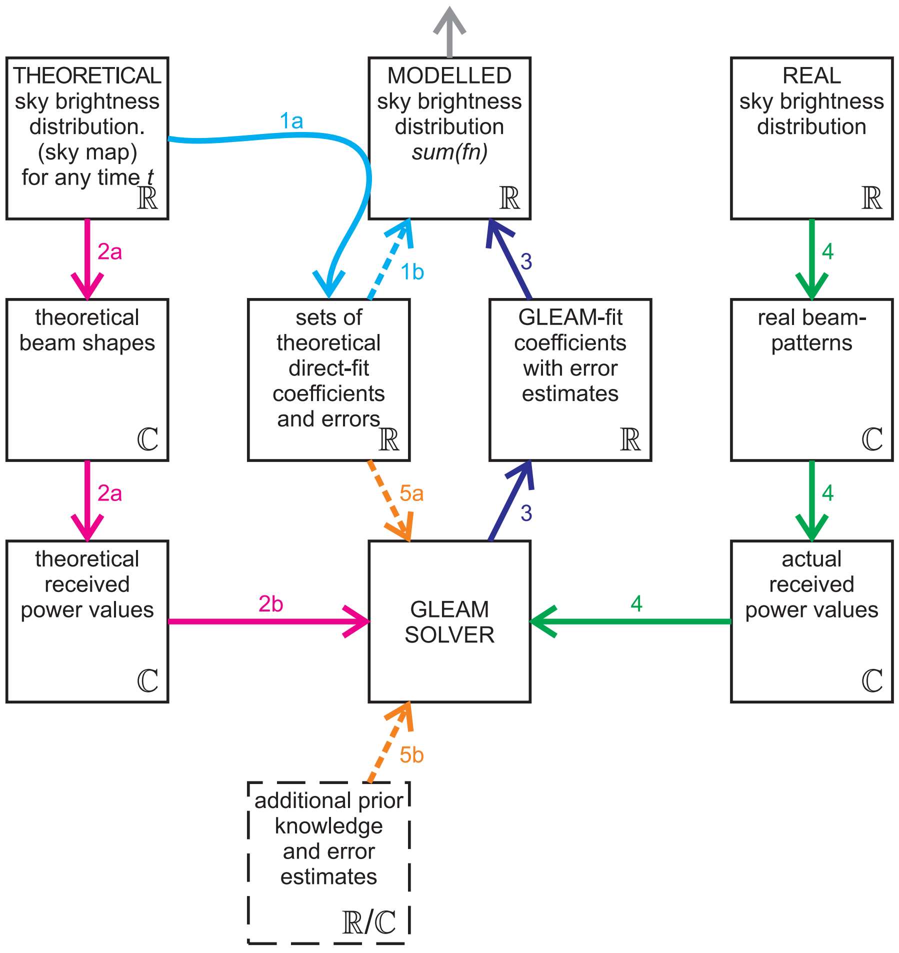

THEORETICAL sky brightness distribution. (sky map) for any time t ℝ
MODELLED sky brightness distribution sum(fn) ℝ
REAL sky brightness distribution ℝ
1a
1b
2a
3
4
theoretical beam shapes ℂ
sets of theoretical direct-fit coefficients and errors ℝ
GLEAM-fit coefficients with error estimates ℝ
real beam-patterns ℂ
2a
5a
3
4
theoretical received power values ℂ
2b
GLEAM SOLVER
4
actual received power values ℂ
5b
additional prior knowledge and error estimates ℝ/ℂ

Figure 10.4: Inter-relations of the parametrised interpolation model method (GLEAM)

10.3.1 Implementation Notes

Note that we do not need to know about the required directions $\overrightarrow{d_i}$ from the outset, we only need to define a grid for numerically integrating equation 10.10, this grid does not directly relate to the possible output directions, it is only used for the integration (summation) process, therefore limiting the accuracy of the results if chosen too wide-meshed. Once we have defined this ‘internal’ grid, the M-by-N matrix C (made up of the values $c_{1\dots N,1\dots M}$ in equation 10.11) can be computed. For any given set of input values b_n we then only need to solve the system of linear equations (equation 10.12) and then use equation 10.7 to calculate the interpolated power values for arbitrary directions (θ, ϕ). In this sense GLEAM has ‘self-interpolating’ properties, meaning that once we have calculated the coefficients c, we are not limited to certain predefined directions. Instead, equation 10.7 will directly give results for any desired direction (θ, ϕ).

10.4 Suitable Orthogonal Basis Functions

To demonstrate how a linear sum of orthogonal basis functions can be used to approximate any given brightness distribution, we approximate three different synthetically generated test distributions using different sets of basis functions and calculate the similarity metric as defined in equation 10.3 (section 10.2.4) for each one. Figure 10.5 shows the results. The top three images are the original distributions, with the reconstructed images being displayed beneath their respective original for various basis functions. The similarity metric m is also given for each case.

As expected, it can be seen that, for the same number of coefficients, use of spherical harmonics results in a much higher similarity than, say, the crude ‘polar block’ approach. Note that images that resemble a typical sky brightness distribution (left column) appear to be represented especially well.

10.4.1 ‘Polar Blocks’

For initial testing purposes, a basic set of orthogonal functions has been implemented in the **gleam_orth_polarblock** functor. These functions divide the upper hemisphere up into little chunks or ‘blocks,’ hence the name. Each function only has a value of 1 within its own little block, and is 0 everywhere else. Thus, these functions are inherently orthogonal. As can be seen in figure 10.5, interpolation results are crude. Nevertheless, the polarblock functor serves

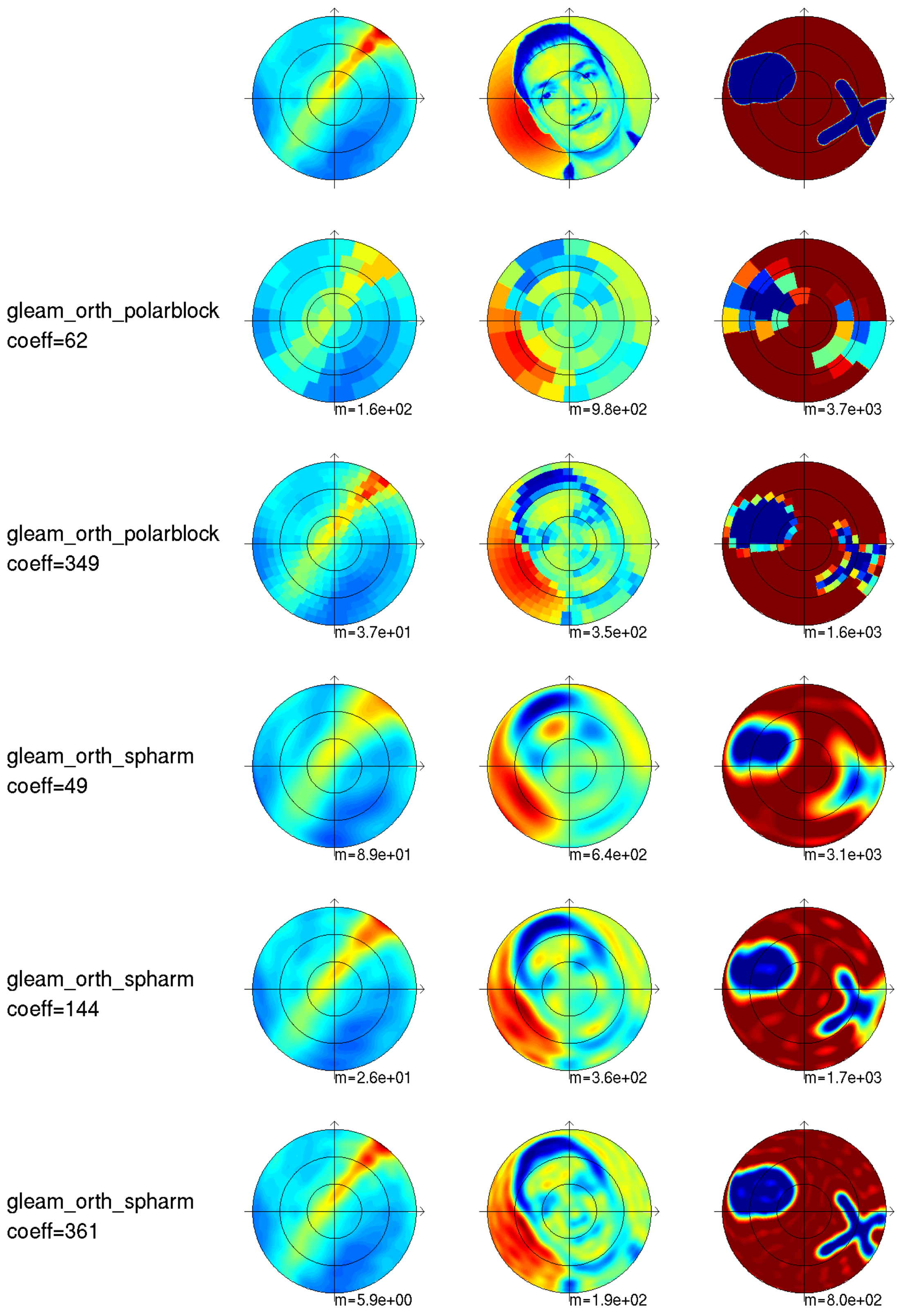


Figure 10.5: Examples of direct orthogonal basis functor fits for three different source images. Left column: theoretical sky map as produced by **CGrillTaohSkyMap**, middle column: highly detailed test image, right column: high-contrast simple shapes. Top row shows original (source) image, remaining rows show image reconstructed from fitting coefficients.

its purpose of demonstrating the working principles behind GLEAM for a straightforward case.

10.4.2 Spherical Harmonics

Spherical harmonics [Hob55] can be used to describe intensity distributions around a sphere. See, for example, [San92] for a geophysical application. Spherical harmonics are useful because they form an orthogonal basis for functions on the sphere in a manner analogous to sines and cosines on the interval $[0, 2\pi]$. We are, of course, only using a finite number of spherical harmonics to approximate these functions, as is often done for Fourier sine/cosine series.

Spherical harmonics are solutions to Laplace's equation in spherical coordinates [Wei02]. Without going into any further mathematical details, we stick to the practical description of the properties of spherical harmonics as found in [PFTV88]:

"The spherical harmonic $Y_{lm(\theta,\phi)}$, $-l \leq m \leq l$, is a function of the two coordinates θ, ϕ on the surface of a sphere. The spherical harmonics are orthogonal for different l and m, and they are normalised so that their integrated square over the sphere is unity."

The degree of the spherical harmonic in question is depicted by l, the order by m. The spherical harmonics are related to *associated Legendre polynomials* P_l^m by the following equation ([PFTV88]):

$$Y_{lm}(\theta, \phi) = K_l^m P_l^m(cos\theta)e^{im\phi} \tag{10.13}$$

Since we are only dealing with real numbers, we will use real spherical harmonics defined as follows [Gre03]:

$$Y_{lm}(\theta, \phi) = \begin{cases} \sqrt{2}K_l^m cos(m\phi)P_l^m(cos\theta), & m > 0 \\ \sqrt{2}K_l^m sin(-m\phi)P_l^{-m}(cos\theta), & m < 0 \\ K_l^0 P_l^0(cos\theta), & m = 0 \end{cases} \tag{10.14}$$

In both equations 10.13 and 10.14, the scaling factor K is defined as:

$$K_l^m = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}} \tag{10.15}$$

Note that this scaling factor (equation 10.15) is constant for any given l, m. For our purposes we can therefore ignore this part, the weighting coefficients will automatically adjust themselves to accommodate for this missing factor.

For modelling the sky brightness distribution according to equation 10.7, we need a linear combination of a number of spherical harmonics less than, or at the most equal to, the number of available beam power readings N (49 in case of the IRIS observations, or anything up to about 500 for ARIES when using the central 556 ARIES beams). For ease of implementation, we establish a unique order between the spherical harmonics starting with $f_1 = Y_{l=0;m=0}$ and then moving on to higher degrees, cycling through all possible orders $m = -l, ..., 0, ..., +l$ for each degree, until we arrive at, for example, $f_{49} = Y_{l=6;m=6}$. Figure 10.6 is a graphical representation of the first 49 real-valued spherical harmonics including order and degree values for each graph.

10.4.3 Adjusted Spherical Harmonics

Ordinary spherical harmonics as described in 10.4.2 have the disadvantage of modelling the brightness distribution over the surface of a complete sphere, resulting in relatively low resolution at any given region of interest. Especially, in our case, a riometer will only ever 'see' at most a hemispherical subsection of the whole sky. It would be sensible to choose a set of basis functions, which represent the power distribution over the hemisphere alone. One possibility is capped spherical harmonics [Hai85]. However, capped spherical harmonics can be complex to implement and can result in a range of computational problems [ST97]. Therefore, in the investigations presented in this chapter, we use a simple approximation to spherical capped harmonics called 'adjusted spherical harmonics' as proposed by DeSantis [San92]. It is based on a coordinate transformation that 'adjusts' (compresses) the elevation angles ϕ in the original definition of the spherical harmonic to those of interest (in our case the visible hemisphere). By employing these adjusted spherical harmonics, we effectively double the resolution in the hemisphere of interest. We still use the same logical order of spherical harmonics as described in section 10.4.2 above.

Simulations to evaluate how much of an advantage these 'adjusted' spherical harmonics show over ordinary spherical harmonics produced the result shown in figure 10.7. It turns out that, at least for our type of images, no clear advantage can be achieved. If anything, only very moderate improvements appear to be achievable by using compression factors of about 1.5 in some cases. Hence, no fitted images using 'adjusted' spherical harmonics are included in figure 10.5 and the subsequent discussions.

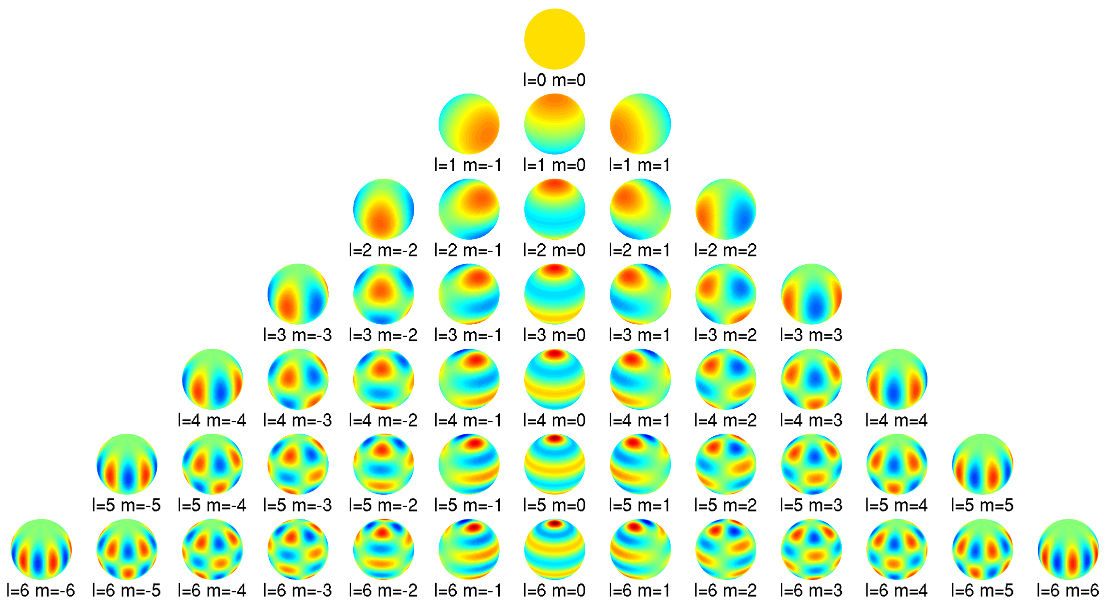


Figure 10.6: The first 49 real spherical harmonics. l=degree, m=order. Colour scale ranges from -1 (blue) to $+1$ (red).

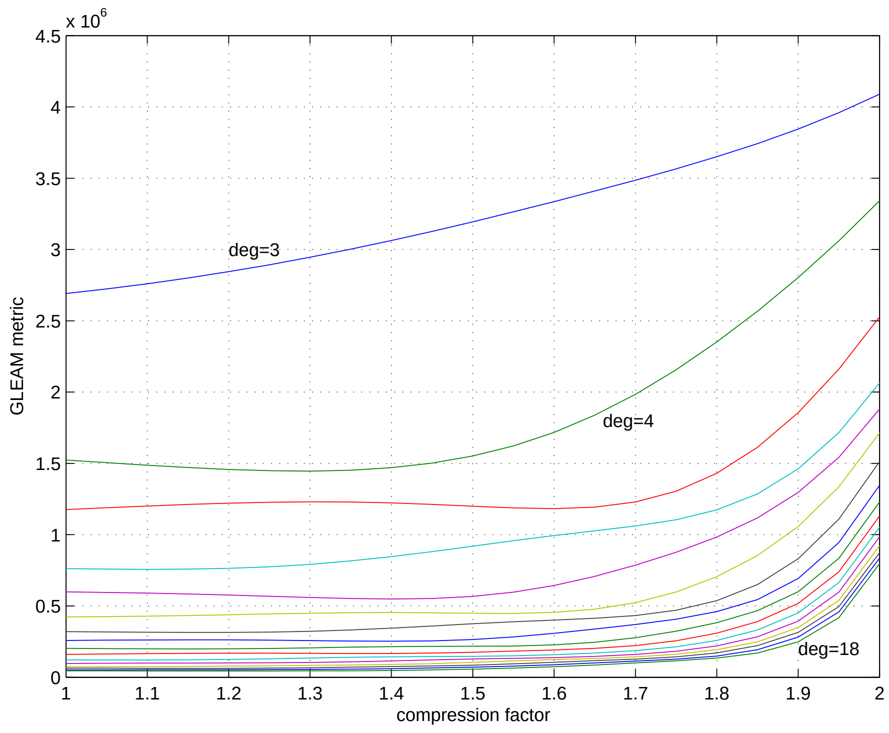


Figure 10.7: The influence of various compression factors on a direct fit of a sky map to 'adjusted' spherical harmonic basis functions of various degrees. 'GLEAM metric' is the quality metric calculated according to equation 10.3 for the respective fit. Lower values are better.

10.5 MATLAB Implementation

For flexibility and ease of use, the GLEAM algorithm was implemented in two independent parts. **gleam_orth_basis_functor** and derivatives (figure 10.8, right-hand side) are functors that provide sets of orthogonal basis functions, along with their respective **calculate()** methods to evaluate their value for any given (set of) direction(s) (θ, ϕ). There is also a function **sum()** for calculating the weighted sum of a given set of basis functions according to the specified weighting coefficients. The weighting coefficients might have been calculated by the GLEAM algorithm (see below), or by a direct fit such as calculated by **gleam_fit()**.

The GLEAM algorithm itself (as implemented by **gleam_genericalgorithm**, see figure 10.8, left-hand side), needs to know about various other parameters, namely

- The instrument for which to run the algorithm. This is because we require knowledge of the individual beams' radiation patterns (equation 10.5).
- If prior knowledge in the form of 'mean' coefficients is to be included, we also need to know about the instrument's location in order to simulate power reception. For this case, a suitable sky map can also be specified.
- The beam numbers of all beams that are to be included in the fitting process. Note that as opposed to simple interpolation as described in section 10.2.2, even beams with significant degradation might be able to contribute something to the overall accuracy of the modelled brightness distribution.
- Potentially, error bars for any input values can also be specified, for example for putting less emphasis on more unreliable beam readings.
- Finally, the GLEAM algorithm needs to know which (set of) basis functions to use when solving equation 10.12. This is indicated by the 'aggregation' arrow in figure 10.8.

In addition to calculating a set of weighting coefficients (which can then be passed to **gleam_-orth_basis_functor::sum()** for retrieving the modelled brightness distribution) for any given input data, **gleam_genericalgorithm** can also display various debugging-type information, including quality indicators for the result based on the input error bars and rank of the internal matrix C.

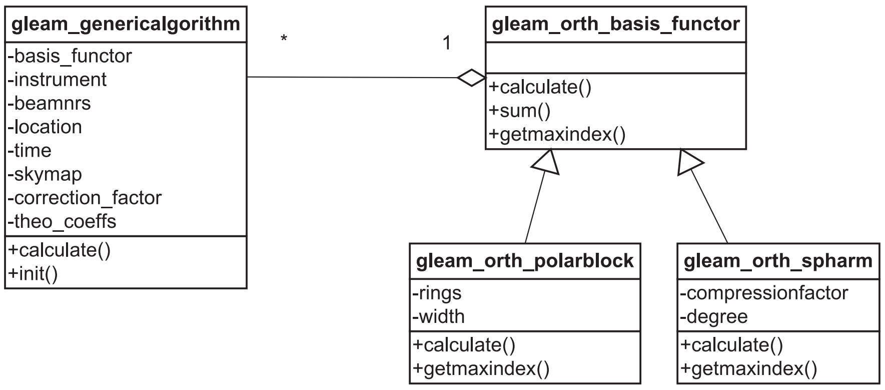


Figure 10.8: GLEAM class diagram

10.6 Performance with Simulated ARIES Data

10.6.1 Comparison with Sky Map

Using the radiation patterns and sky maps as discussed in chapters 6 and 7 to generate simulated received power values for the ARIES riometer, we can then apply the GLEAM algorithm to these simulated received values in order to reconstruct a model of the original sky brightness distribution, resulting in a set of coefficients $\{\gamma_1 ... \gamma_M\}$. With these coefficients, we can reconstruct the sky brightness distribution, and since all values stem from a simulated sky (the sky map) in the first place, we can then directly compare reconstruction and original using the metric defined in section 10.2.4.

As an additional measurement of the accuracy of the reconstruction, we also yet again simulate power reception, this time using the reconstructed sky as brightness distribution for the simulation. We expect these simulated beam power values to closely match the ‘original’ ones. All these results are shown in figures 10.9 to 10.12, for four different example constellations. The four figures are laid out identically, the following description is valid for each one of figures 10.9 to 10.12.

The top left-hand panel shows the sky map for the particular moment in time that the simulation was run for. The top central panel shows a direct fit of the model that was used in the respective simulation to this sky map, and its similarity metric. This process is depicted by paths ‘1a’ and ‘1b’ in figure 10.4: knowing about the model, a set of coefficients is derived directly from the sky map using a least-squares fit (path ‘1a’). These coefficients are then fed straight back into the model, resulting in a direct-fit modelled brightness distribution (path ‘1b’).

The sky map in combination with the theoretical radiation patterns for (a subset of) all riometer beams can also be used to simulate reception (as discussed more extensively in chapters 2 and 7). See path ‘2a’ in figure 10.4. This leads to the simulated beam power values depicted in the left-hand bottom panel (shown together with further simulated beam power values as discussed below). The central bottom panel shows the same values on a 32×32 matrix plot, with colour representing linear power units.

These simulated beam power values can then feed into the GLEAM solver (path ‘2b’ in figure 10.4), which does not require additional prior knowledge in this purely simulated case. GLEAM will output a set of best-fit coefficients based on the input complex beam power values, and these parametrise the model to produce a model brightness distribution (path ‘3’). The top

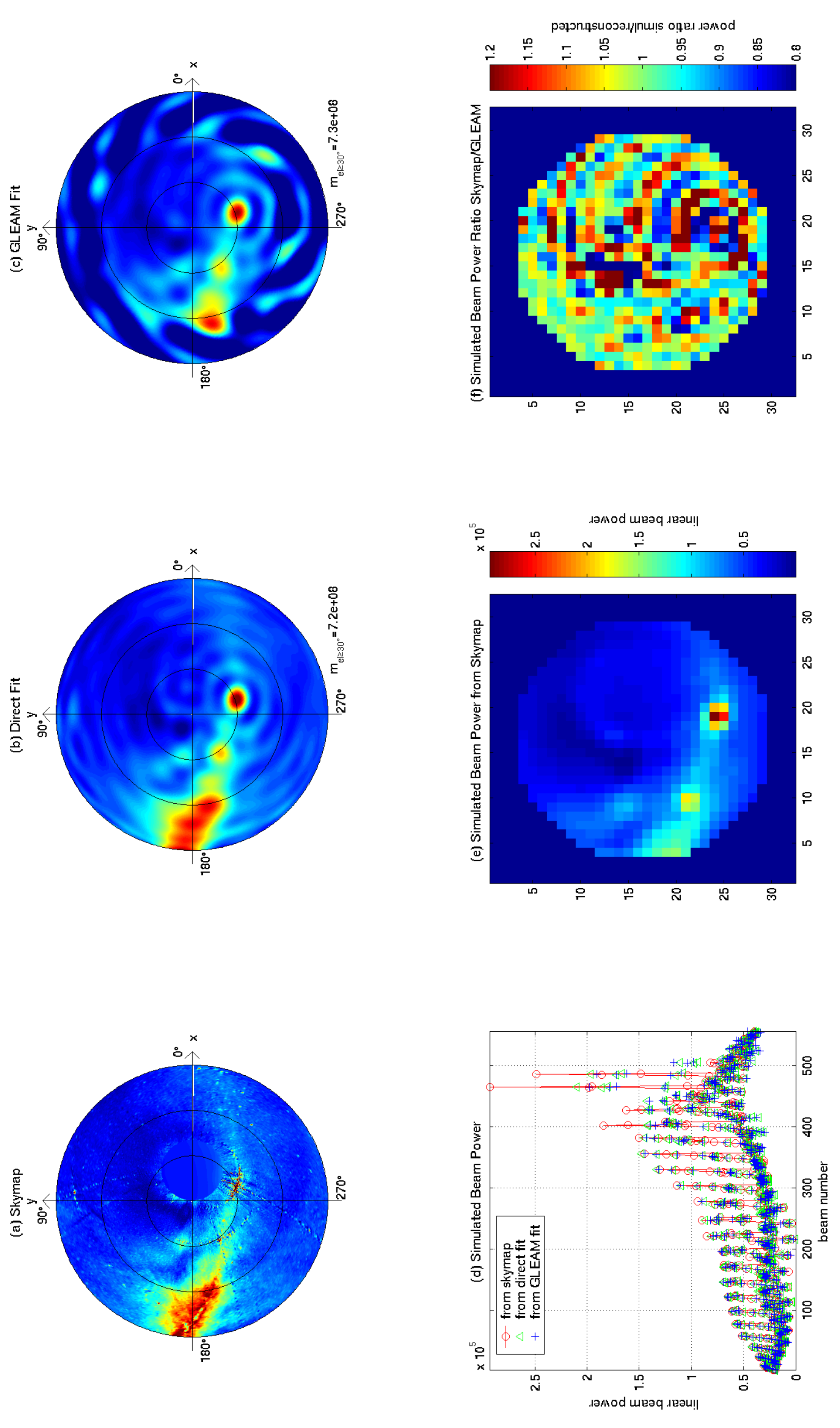


Figure 10.9: GLEAM with simulated ARIES data; Constellation 1. (a) Simulated sky map, (b) direct fit of model to sky map, (c) GLEAM fit using simulated received beam power values shown in (d), (e) simulated received beam powers visualised as 32 $\times$ 32 matrix, (f) comparison between simulated received beam powers from sky map versus from GLEAM reconstruction. Further explanations see text.

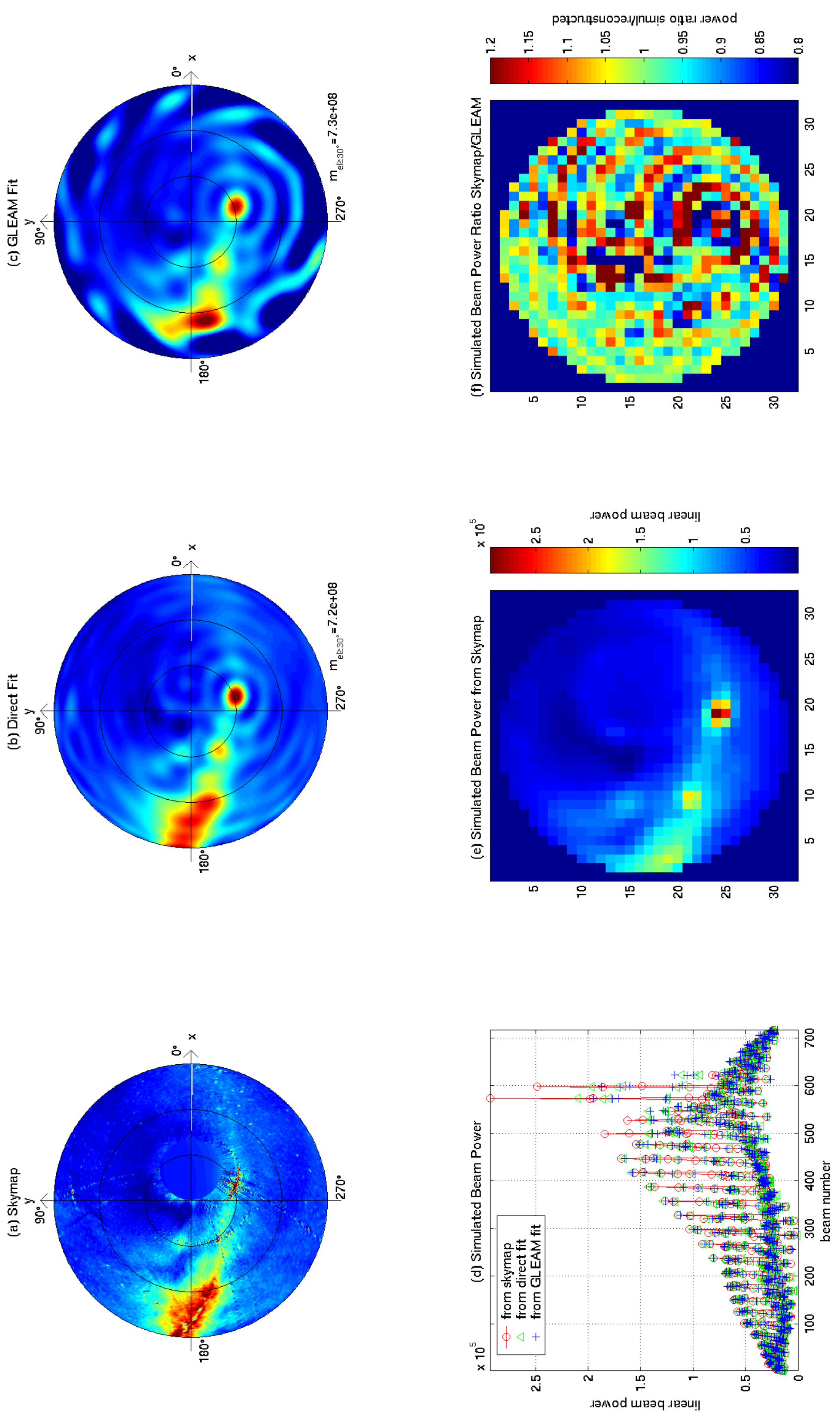


Figure 10.10: GLEAM with simulated ARIES data; Constellation 2. Explanation see figure 10.9.

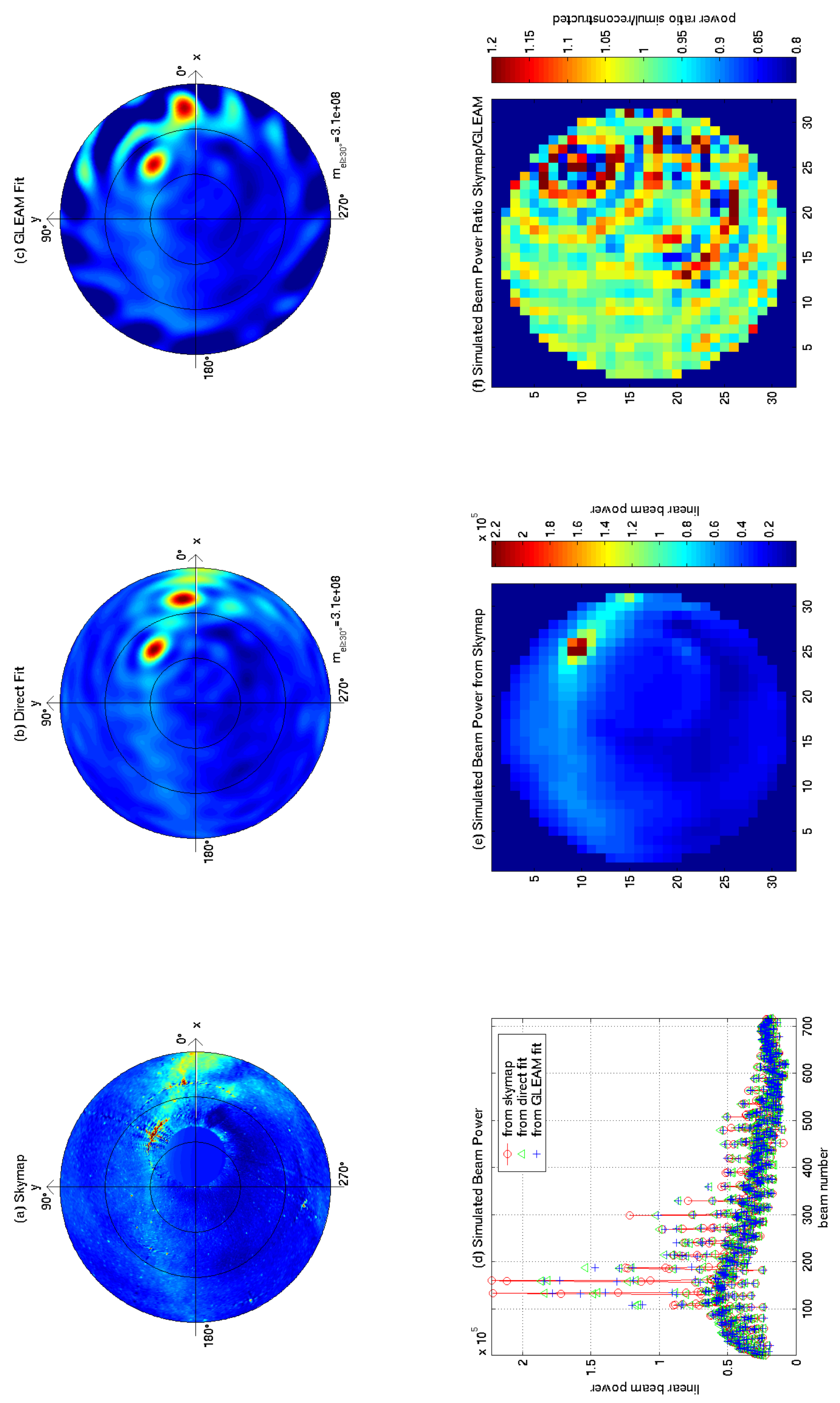


Figure 10.11: GLEAM with simulated ARIES data; Constellation 3. Explanation see figure 10.9.

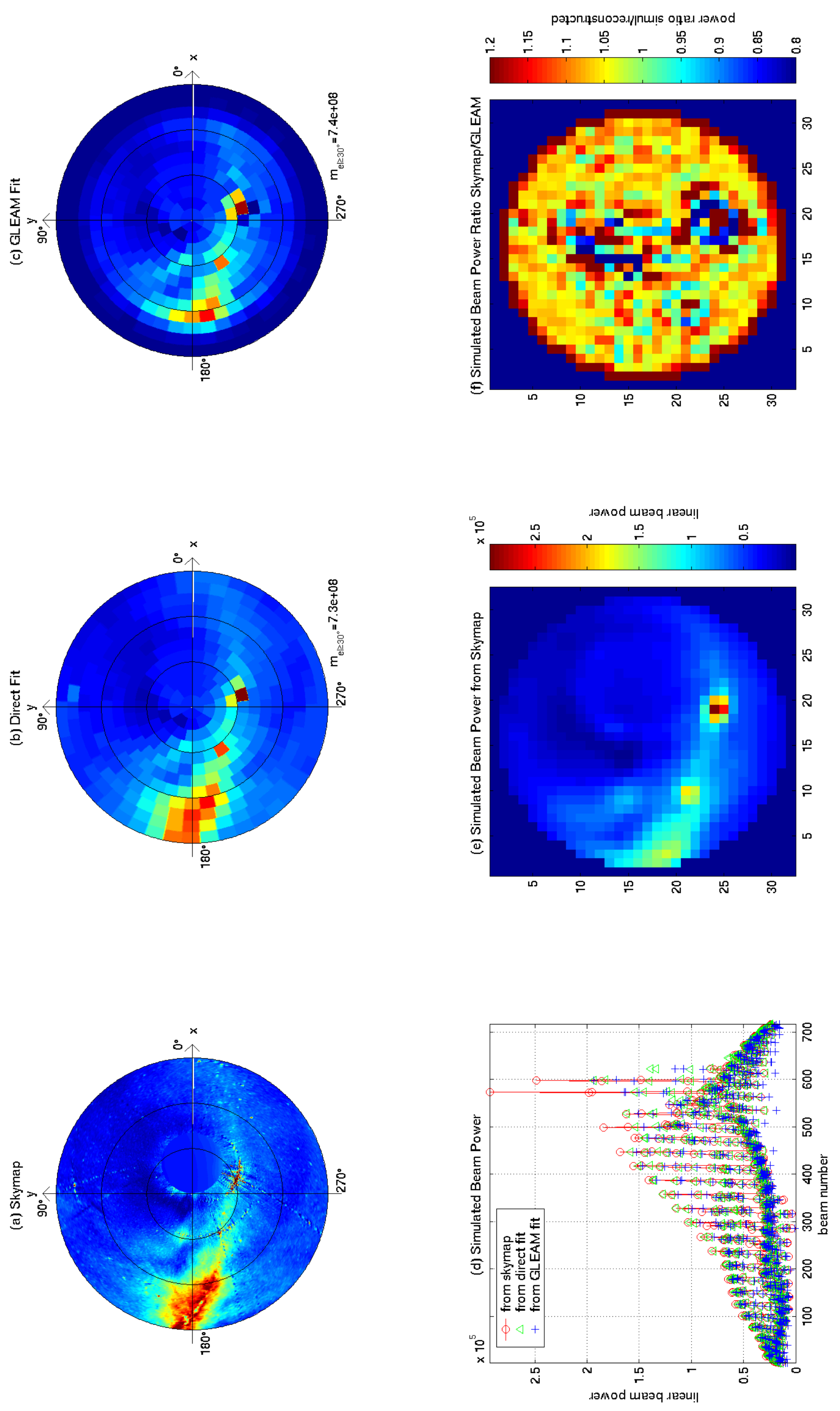


Figure 10.12: GLEAM with simulated ARIES data; Constellation 4. Explanation see figure 10.9.

right-hand panel shows this resulting distribution together with the similarity metric comparing it to the original sky map.

To go round full circle, this reconstructed brightness distribution can once again be used as the source for a simulated power reception, and the simulated received power values should match the ones 'received' from the original sky map closely. All three sets of power values are shown in the bottom left-hand panel as mentioned earlier, and the bottom right-hand panel shows the power ratio between GLEAM-fit reconstructed beam power and directly sky map-derived power for each beam on a 32×32 matrix plot, with the colour scale ranging from 80% to 120%. In addition to the metrics calculated in the upper panels, the (in)homogeneity of the matrix in the lower right-hand panel is yet another measure of how well the reconstruction worked, and which beams are particularly out. The colour scale is a direct measure of the relative difference between absolute-power-from-original-sky-map and absolute-power-from-reconstituted-sky-map for each beam. The ringing effect produced by the spherical harmonics functors for strong point sources is especially evident in this kind of plot.

Table 10.1 summarises the configurations used, and results achieved, for the four figures 10.9 to 10.12. Simulations were run for both a set of 556 central ARIES beams and a larger set of 716 central beams. Two different moments in time, i.e. sky map orientations, were simulated. An example using the generally inferior polarblock set of basis functions is also provided for comparison (figure 10.12).

The metric as calculated for the figures discussed above only takes values above an elevation angle of 30° into account, for two reasons: (1) As will also be seen in later figures using real data (see section 10.7.2), GLEAM performs badly around the horizon, primarily due to the fact that antenna sensitivity tends towards zero as one approaches the horizon (effective antenna aperture goes to zero), so there is very little information present in the data for GLEAM to extract. (2) For later comparison with the 'traditional' interpolation algorithm, we have to limit ourselves to the largest common field of view of the two algorithms, which, for the traditional algorithm is limited by the locations of the lowest beam centres together with the distortion introduced by interpolating in the FLATM coordinate system (and not on the native spherical coordinate system of the receiving antenna). This effectively limits the field of view, for which reasonable data in all azimuth directions is available, to $el \geq 30^\circ$ for the central 556 ARIES 'good beams.'

case	fig.	time	model	beams	direct-fit metric	gleam-fit metric	traditional interp. metric
1	10.9	2006-02-22 07:00h	spharm (degree 19)	central **556** beams (‘good beams’)	$m_{el\geq 30^\circ} = 7.2e8$	$m_{el\geq 30^\circ} = 7.3e8$	n/a (see text)
2	10.10	2006-02-22 07:00h	spharm (degree 19)	central 716 beams (‘existing beams’)	$m_{el\geq 30^\circ} = 7.2e8$	$m_{el\geq 30^\circ} = 7.3e8$	$m_{el\geq 30^\circ} = 7.1e8$
3	10.11	2006-02-22 **20:00h**	spharm (degree 19)	central 716 beams (‘existing beams’)	$m_{el\geq 30^\circ} = 3.1e8$	$m_{el\geq 30^\circ} = 3.1e8$	$m_{el\geq 30^\circ} = 3.0e8$
4	10.12	2006-02-22 07:00h	**polarblock** (12 rings)	central 716 beams (‘existing beams’)	$m_{el\geq 30^\circ} = 7.3e8$	$m_{el\geq 30^\circ} = 7.4e8$	n/a (see text)

Table 10.1: Summary of GLEAM performance with simulated ARIES data. Refers to figures 10.9 to 10.12. Important differences in the simulation configurations from figure to figure are emphasised in **bold**. A degree 19 spherical harmonics model uses 361 coefficients, a 12-ring polarblock uses 349 coefficients. Further explanations see text.

10.6.2 Comparison with Traditional Image Interpolation

In figures 10.13 and 10.14 we compare the respective cases shown in figures 10.9 to 10.12 to the traditional IRIS interpolation approach. Figure 10.13 shows the image at various stages in the interpolation process, for visualisation and reference purposes. This is a good demonstration of the distortions involved in the various projections. Figure 10.14 shows the interpolated images for each of the 3 different cases 1–3 together with the value of the similarity metric (again calculated for $el \geq 30°$ due to the field of view being limited by the elevation angles of the actually used beams in the traditional algorithm. (No metric has therefore been calculated for case 1, as the beams don't fully cover the area $el \geq 30°$.) Case 4 in figure 10.12 only differs from the other cases in that polar blocks are used as underlying basis functions for the GLEAM algorithm. This is of no influence on the traditional algorithm, hence the traditional image would look identical to that for the case in figure 10.10. As said above, no metric was calculated for case 1 due to the even smaller field of view. In the traditional interpolation method, case 1 behaves identical to case 2, so long as only the central area covered by both configurations is considered. Note that scaling for the traditional algorithm is somewhat arbitrary, as this algorithm interpolates beam power values which are related to, but not the same thing as, the sky brightness distribution. These values have been scaled so that their mean value matches the mean of the 'considered' part of the sky brightness distribution, before calculating the metric.

10.6.3 Behaviour in the Presence of Absorption

Deviating from the virgin sky map, we now simulate an absorption patch by attenuating the signal from a certain part of the sky as illustrated in figure 10.15. The path that the absorption patch takes from beginning until end of the simulation is also show by means of 'snapshots' at 5 time unit intervals. In order to have fewer parameters, (sidereal) time was 'frozen' for these simulations, i.e. the position of the sky map relative to the observing instrument remains constant while the absorption patch moves through the field of view. This is why the time scale is in arbitrary time units.

Figure 10.16 records the simulated beam readings for the beams highlighted in figure 10.15 over time, with the simulated absorption patch moving through the sky as depicted in figure 10.15. Panel (a) shows raw beam received power values, panel (b) is the GLEAM sky brightness reconstruction in the respective beams' main pointing directions.

All power/brightness values are shown on a dB scale, relative to their quiet-day levels, i.e. the

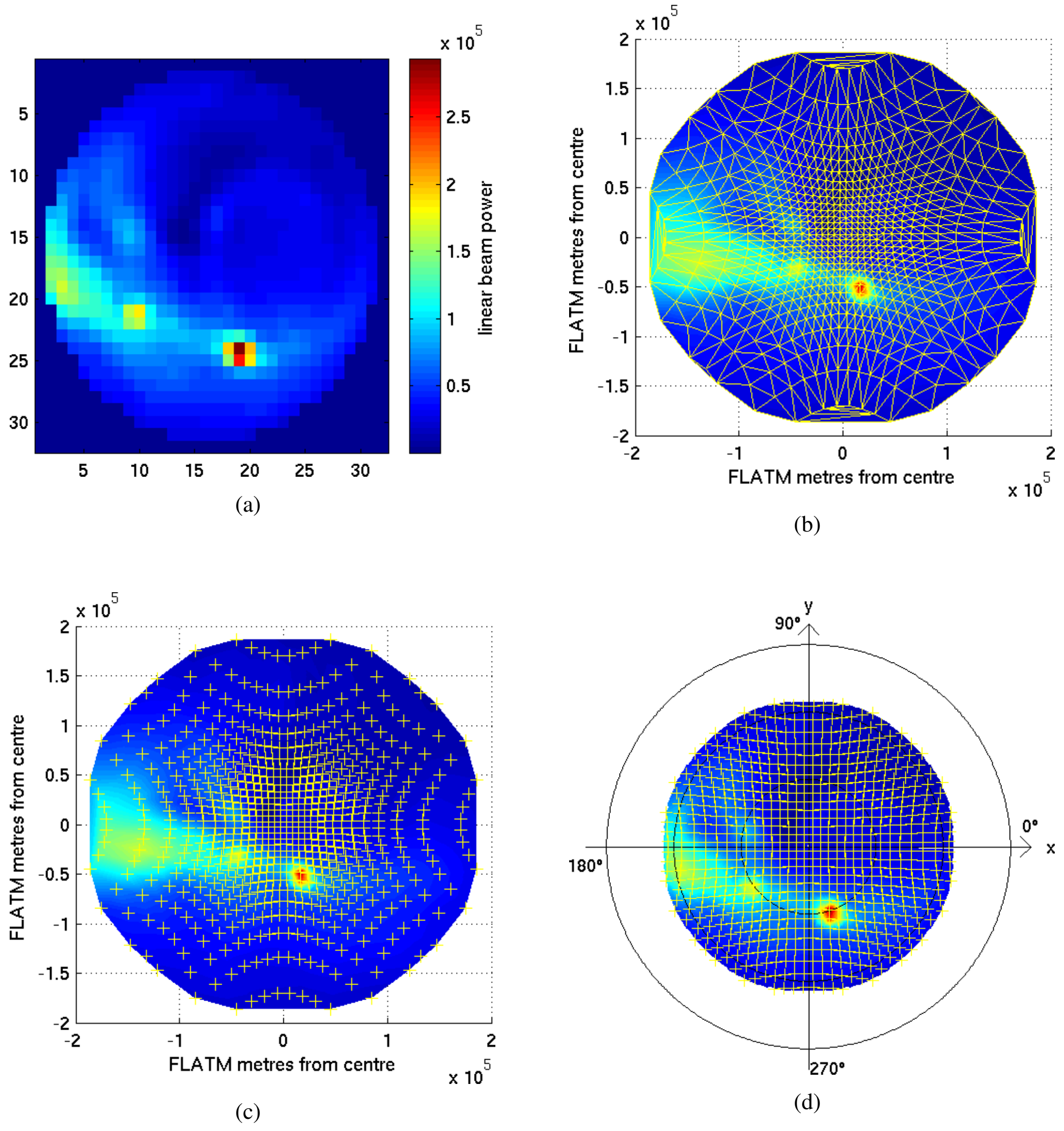
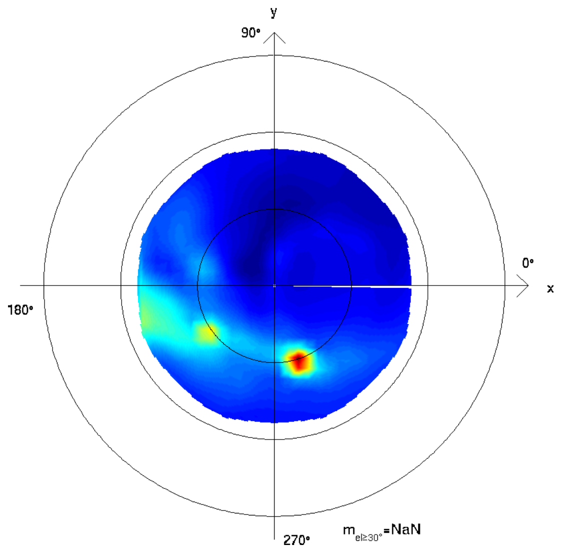
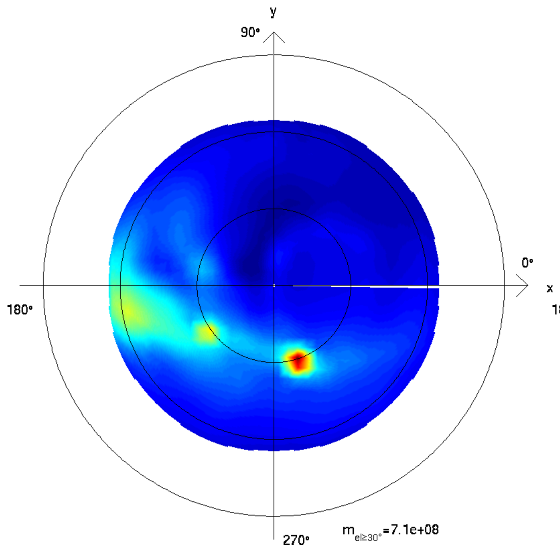


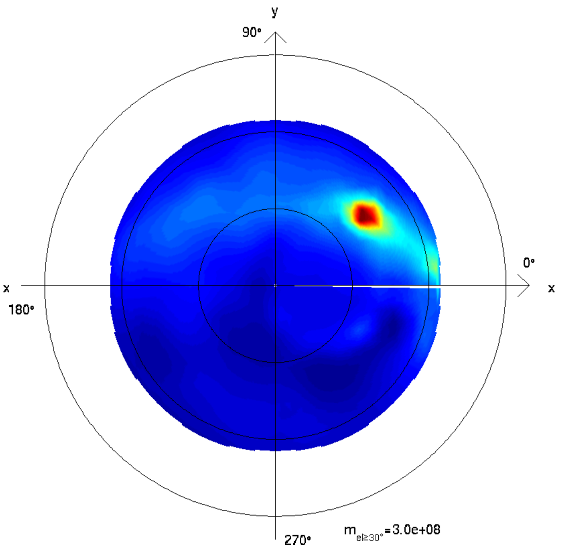
Figure 10.13: Visualisation of the traditional image interpolation algorithm for ARIES. (a) 32×32 matrix plot of simulated beam power values; (b) interpolated image in FLATM coordinate system, together with vertices of triangles used for interpolation; (c) same as (b), but only beam centres highlighted; (d) polar plot of (c), again with highlighted beam centres.



(a) Case 1



(b) Case 2



(c) Case 3

Figure 10.14: Performance of the traditional interpolation method for the three constellations 1–3. No metric is calculated for case 1 because the area covered does not encompass the whole $el \geq 30°$ area. The two m values for cases 2 and 3 compare directly to those in figures 10.10 and 10.11.

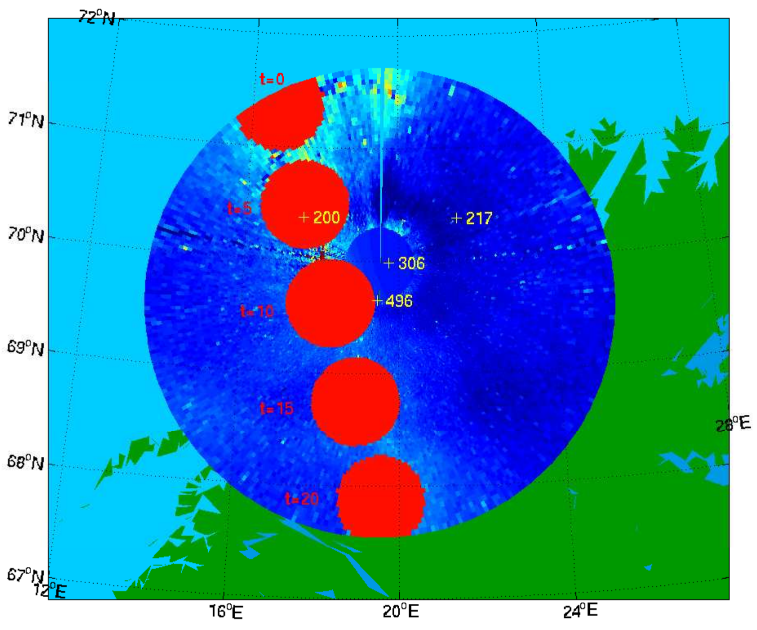
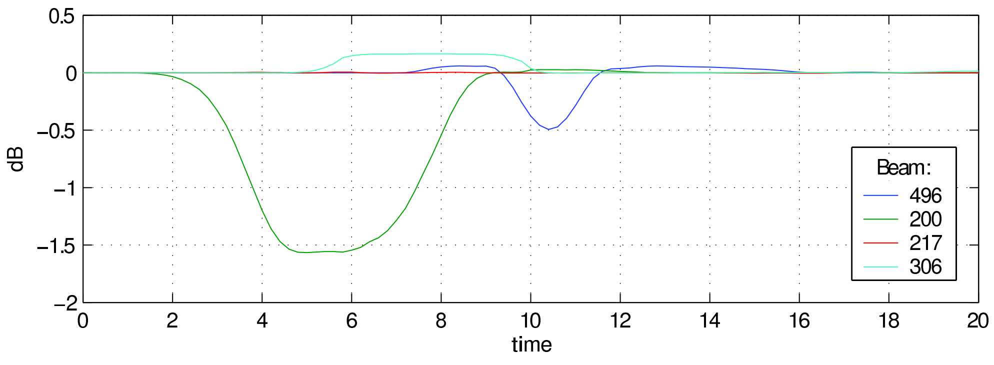
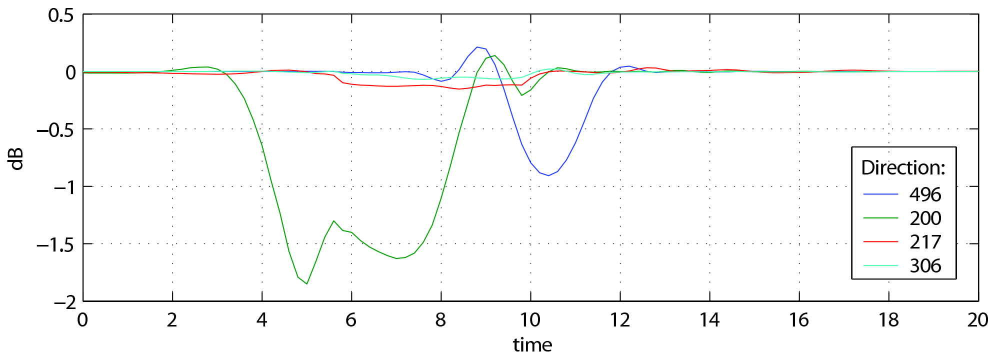


Figure 10.15: Path of a simulated absorption patch across a 'frozen' sky map. FLATM-projection down to $el = 20°$. Also shown are the beam centres of some ARIES beams.



(a)



(b)

Figure 10.16: Simulated power readings during absorption event for selected beams. (a) beam power, (b) GLEAM reconstructed brightness in main pointing direction. X-axis in arbitrary time units corresponding to the time scale shown in figure 10.15.

levels without the presence of the simulated absorption patch. Due to the 'frozen' time axis, the quiet-day levels are perfectly constant. The solid lines show the variation of beam (sky) power as the simulated absorption patch moves through the sky along the path indicated in figure 10.15.

As expected, beam 200, which is located directly in the path of the absorption patch, shows a clear reduction in signal for both cases during this time ($3 \leq t \leq 9$). More importantly, however, it can be seen that, for example, beam 306, which is clearly not in the path of the patch, nevertheless suffers from its presence, see the period $6 \leq t \leq 10$. Worse still, whereas in a filled array type riometer we would simply expect this beam to show a certain amount of absorption (as the effect of the beam's sidelobes: some of the sidelobes are covered by the absorption patch, therefore reducing overall received power for that beam), in case of a cross-correlating riometer, this beam actually observes an *increase* in power due to a *decrease* in negative correlation (some area of the sky that is seen by the fan beams with a $\sim 180^\circ$ phase shift are obscured by the 'absorption' patch). This is the 'negative correlation' issue alluded to before (see especially chapters 4 and 9). To a lesser extent, this effect is also visible in the readings for beam 496, shortly before ($8 \leq t \leq 9$) and after ($12 \leq t \leq 16$) the patch touches that beam.

Due to the fact that GLEAM knows about the complex radiation patterns, we expect the GLEAM reconstruction to lessen, if not eliminate, this effect. Panel (b) in figure 10.16 shows the reconstructed brightness values in the main pointing directions of the same set of beams (now referred to as 'virtual beams'). Again, beam 306 is clearly outside the path of the absorption patch. It can be seen that, while the reconstruction does improve matters, some remnant of the effect still remains. A certain amount of ringing is also evident, caused by the sharp boundaries of the simulated absorption patch.

10.6.4 Additional Theoretical Knowledge

In preparation for the GLEAM runs with real data, we now introduce additional information derived from simulations. This information can be used as 'additional knowledge' in the real data scenario discussed in section 10.7 below.

Our primary source of additional knowledge is the expected general shape of the sky brightness distribution. By directly fitting our model to a theoretical sky map for several specific times during one (sidereal) day, we can get an idea of how the weighting coefficients for that particular model are expected to vary over time. This was visualised in figure 10.17 for the coefficients of a degree 19 **gleam_orth_spharm**-based direct fit at 10 minute intervals during the course of one day. Coefficients 7, 13, 21, 31 and 57 are labelled. We can see that each coefficient tends to oscillate within a certain range about some mean value. This can be fed into the GLEAM solver in the form of expected value and associated error bars (path '5a' in figure 10.4).

In addition, a simple 'damping' prior knowledge can also be used to reduce the overshooting evident in GLEAM interpolated images. This knowledge simply consists of a target value of 0 for every coefficient, along with relatively large error bars.

The generic GLEAM implementation in **gleam_genericalgorithm** supports easy addition of new prior knowledge (path '5b' in figure 10.4) by adding the appropriate equation-generating lines to the **calculate()** function. It is likely that more advanced forms of prior knowledge can also be devised, which may well depend on specific underlying basis functors. The currently implemented functionality (simulated sky map knowledge and damping knowledge) can be applied for all functors.

10.7 Performance with Real ARIES Data

In this section, we will apply the GLEAM algorithm to real ARIES data. As a prerequisite, we will investigate how closely the simulations match the real recordings (section 10.7.1). A good match is important, as GLEAM relies on adequate knowledge about the receiving instrument's beam patterns to correctly fit the modelled sky brightness distribution to the data. Section 10.7.2 contains examples of real data GLEAM interpolations.

The data flow adhered to in this section is indicated by paths '3', '4' and '5' in figure 10.4: Real recorded power values (resulting from the convolution of the real beam radiation patterns with the real sky brightness distribution) form the input to the GLEAM algorithm (path '4'). For any moment in time, this results in a set of coefficients, which parametrise the sky brightness model (path '3'). Additional knowledge (see section 10.6.4 above) from direct sky map fits (path '5a') and other sources (e.g. general damping — path '5b') may also be included.

10.7.1 Comparison of Real ARIES Data to Simulation

For the interpolation algorithm to work adequately, it is necessary to describe the beam patterns of the receiving instrument correctly in both phase and amplitude. During initial experiments, the actual hardware was changed several times, from initial phasing leads to analogue Butler Matrix electronics to fully digital beamforming in various untapered and tapered flavours and configurations. This makes comparisons difficult, and for accurate simulations each type of beamformer needs to be accurately modelled in software. We will first provide an impression of how closely real recorded data and simulation match for the latest system incarnation, and having confirmed that they do indeed match, will then proceed to apply the GLEAM algorithm to real ARIES power data. The resulting images will be compared to images generated by interpolating between beam centres in the 'IRIS' way (section 10.2.2).

For the investigations in this chapter, we will mainly be concerned about reconstructing the sky brightness distribution on medium (minute) timescales. We will therefore mainly be using 60s post-integrated data.

The 'earlobe' plots presented below are based on multi-day (sidereal) average values. This is essentially an 'unbiased' quiet-day curve calculator in that it does not make assumptions as to whether the data consists of mainly spikes or absorption and tries to calculate a suitable envelope, but simply derives a mean quiet day. Provided a large enough dataset (1 month works adequately) is available, smooth quiet-day curves can be obtained. In figure 10.18, we plot some of these as 'earlobes' (i.e. their variation over time) on the complex plane. The corresponding plots for all 1024 ARIES beams can be found in figures G.1 (overview) and G.2 to G.7 (zoomed-in) in appendix G. Figure 10.18 is a an even further zoomed-in view of figure G.1, used here to explain the plot format.

Each panel contains data specific to one ARIES beam. The MIA (high-level standardised) and raw (as output by the FPGA firmware) beam numbers are given in the top left corner of each panel.

The yellow curve is actual ARIES data for one day, specifically 2007-03-23, a reasonably undisturbed (quiet) day. The data has been averaged (post-integrated) to 30 seconds. The curve shows the diurnal variation of the (complex) beam power value on the complex plane. The origin is the centre of the plot, the positive real axis extends to the right and the positive imaginary axis to the top. The scaling is identical for all panels. For cleaner plots, no labels have been plotted, as the absolute values at this stage are arbitrary (linear) raw ADC units.

The black curve is a one-month average centred on 2007-03-23. This is essentially a quiet-day curve as discussed in chapter 3, calculated by the following unbiased averaging algorithm:

1. Collect all raw (1 second) sample values for current time $\pm(0...60)\text{s} \pm (0...15) \times 86163\text{s}$ (86163s is the duration of one sidereal day).

2. Sort all these samples in ascending order.
3. Throw away the bottom and top thirds.
4. Calculate the mean of the remaining samples. This is the (unbiased, i.e. neither favouring high nor low values) quiet-day value for this moment in time.
5. Increment current time by 30s, repeat until end of day.

Note that this algorithm favours neither high power (e.g. through solar interference etc.) nor low power (absorption), but provides a well-balanced mean value. This is opposed to the bias often used in riometry quiet-day curve generators that try to establish the upper envelope of a set of sidereal power variation recordings. Here, for comparison with simulations, we are interested in as unbiased a view as possible.

Together, the yellow and black curves give an idea of how the mean recording varies from any actual day.

Finally, the red curve is the simulated power variation during one day, calculated purely theoretically from sky map and beam radiation patterns as described in chapter 7.

It can be noted that real data and simulation are generally in good agreement in both phase and amplitude (see also the plots for all beams in appendix G). This is an important prerequisite for running the GLEAM algorithm with real data.

10.7.2 Real Data Image Plots / Movies

This final section presents sequences of power image data generated from real ARIES data for both a quiet and a disturbed dataset. For each case, images generated using the traditional interpolation method are also shown for comparison. Figure 10.19 is a sequence of images for the whole day 2007-03-23 at 1h intervals for both the traditional (top panel) and the GLEAM (bottom panel) interpolation methods.

Figure 10.20 is a temporal zoom into the 24min block at 8:00–8:23 on 2007-03-23 aiming to show the minute-to-minute variation for both algorithms. The colour axis in all panels is in arbitrary linear power units. Note that, for GLEAM, these power units directly relate to sky brightness (temperature), as GLEAM internally uses a model of sky temperature. To obtain an impression of the 'raw' performance of GLEAM biased as little as possible by simulations, all figures for this section were generated without making use of additional prior knowledge in the form of sky map-derived mean coefficient values and error bars. A coefficient damping factor of ‘1’ as described in section 10.6.4 is used.

The final sequence of images (figure 10.21) is the graphical representation of an absorption event as recorded on 2007-03-24 between 4:00h and 8:00h. Pictures are at 10 minute intervals.

10.8 Summary and Conclusions

The motivation behind interpolating riometer images was presented, along with various prerequisites. The traditional image interpolation algorithm as used within the SPEARS group was explained. The new GLEAM approach was then developed and applied to several simulated and real ARIES datasets in different configurations.

Deriving GLEAM interpolated images from simulated beam data (section 10.6) shows that GLEAM interpolated images have a tendency to show ‘ringing’ effects when using spherical harmonics as the underlying basis functions. This is especially evident in the vicinity of strong point-like power sources, and can be seen well in panels (b) and (f) of figures 10.9 to 10.11 and, to a lesser extent, also in figure 10.12.

Compensating for the effects of negative correlation is one of the main reasons behind investigating this alternative image interpolation method, and this was shown to work as expected (section 10.6.3), although the ringing effect makes results less clean than was hoped for, see especially figure 10.16.

The availability of final ARIES data from the complete and fully working system from 2007-03-05 onwards allowed applying GLEAM to real data, this was done in section 10.7. An initial comparison between real and simulated data was used to confirm the close match between real and simulated recordings, confirming that the beam pattern descriptions as used internally by GLEAM are accurate. Sequences of GLEAM images were then plotted, along with traditionally interpolated images, for a whole day, a short period of 24 minutes and a period of absorption. GLEAM accurately represents the sky brightness in all three cases. Whereas the traditional interpolation method produces artefacts due to the interpolation grid and sidelobes, spherical harmonics-based GLEAM images tend to contain the ringing effects alluded to earlier. Also, variations affecting the whole image from instant to instant, as opposed to the more localised issues seen in the traditional approach, are evident in the GLEAM interpolated images. Some methods to further improve the performance of GLEAM will be suggested in chapter 11.

In general, GLEAM images produce good maps, down to elevation angles below the pointing directions of the beams used in the traditional approach, making best use of what little information the data contains about these regions. GLEAM interpolated images are a major step from beam-centred thinking towards the more generic approach of considering continuous descriptions (in both time and space) of sky brightness the primary output of a new generation of riometers.

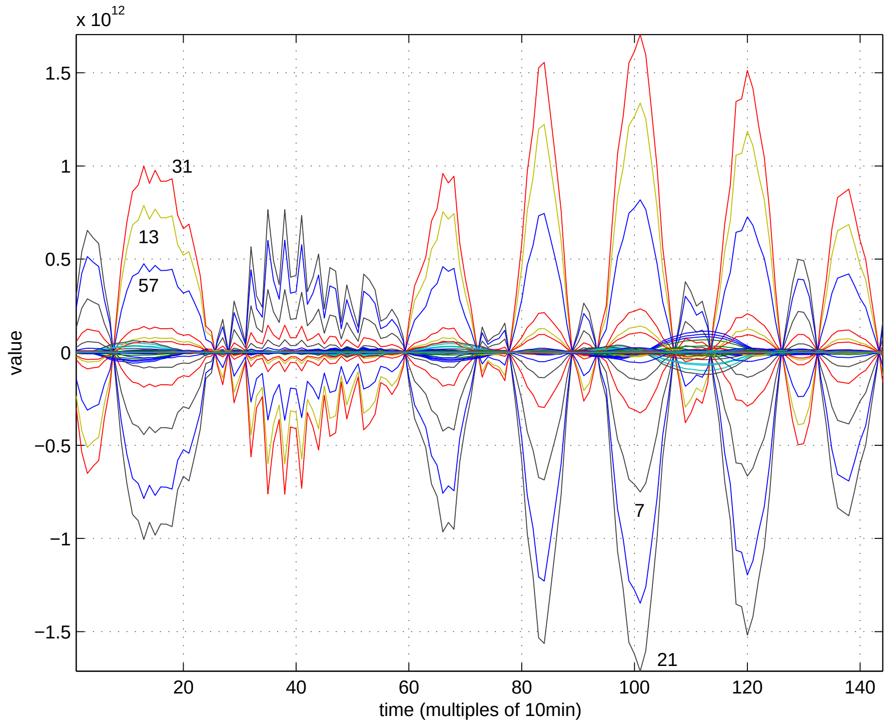


Figure 10.17: Diurnal variation of model weighting coefficients for direct sky map fit. Shown are the coefficients for a **gleam_orth_spharm** functor of degree 19. Some of the higher-amplitude curves have been labelled with their respective coefficient number.

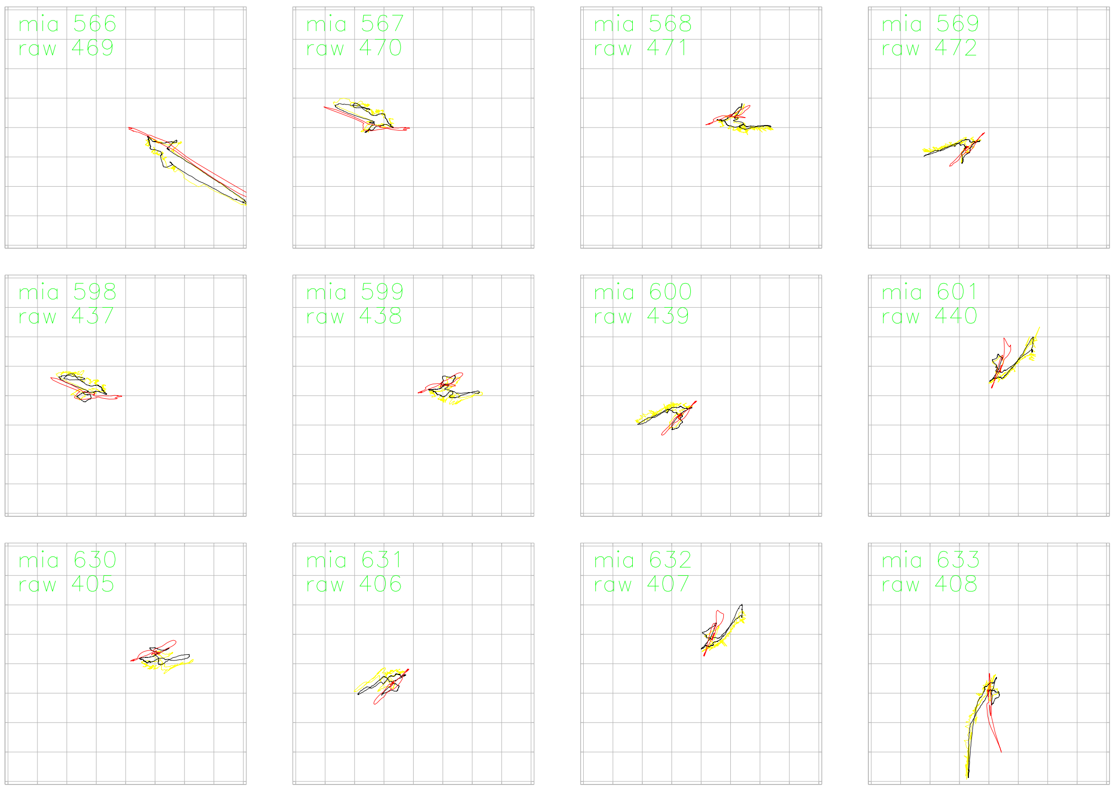
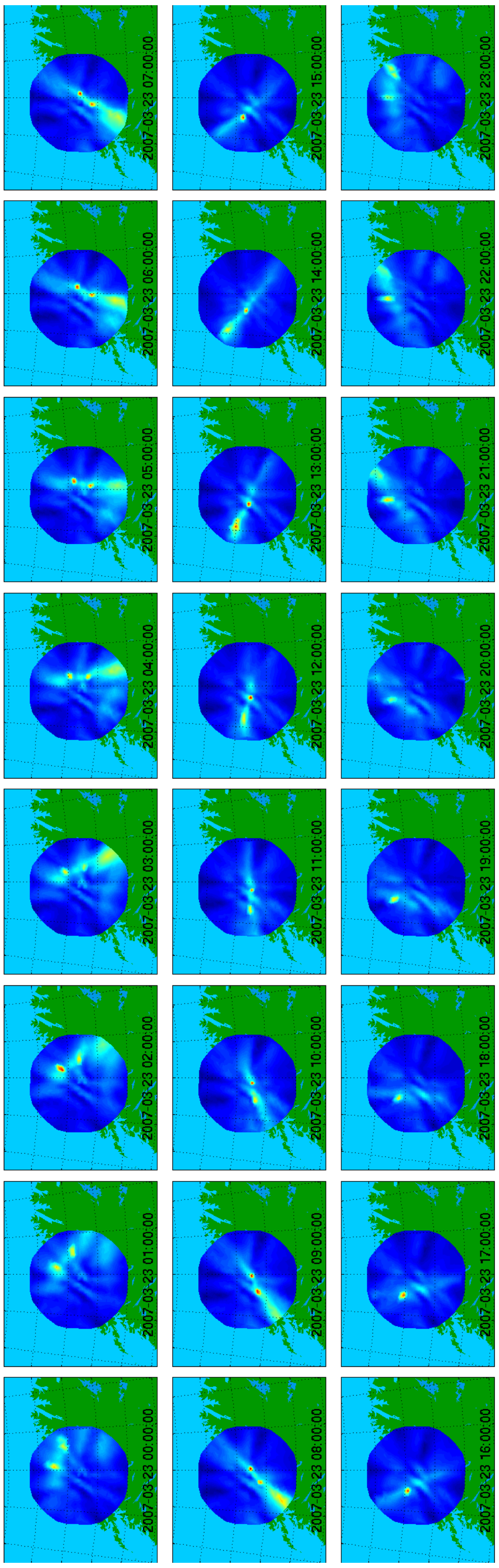
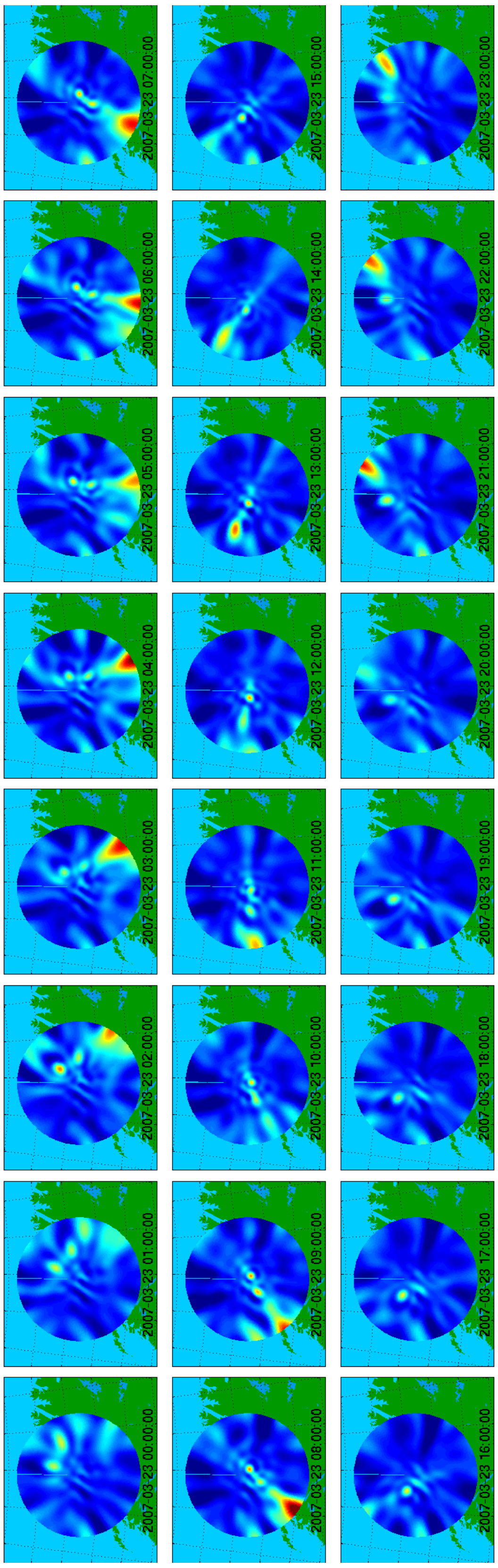


Figure 10.18: ARIES real and simulated complex beam data for 12 exemplary beams. Each panel is a plot of power data over one sidereal day on the complex plane. X-axis is the real axis from left (negative) to right (positive), y-axis is the imaginary axis from bottom (negative) to top (positive), origin is at the centre. Each panel is for one ARIES beam and includes 1-month average (black) ±15 days of 2007-03-23, single-day data (yellow) for 2007-03-23 and simulated data (red). Axis scaling in arbitrary linear power units.

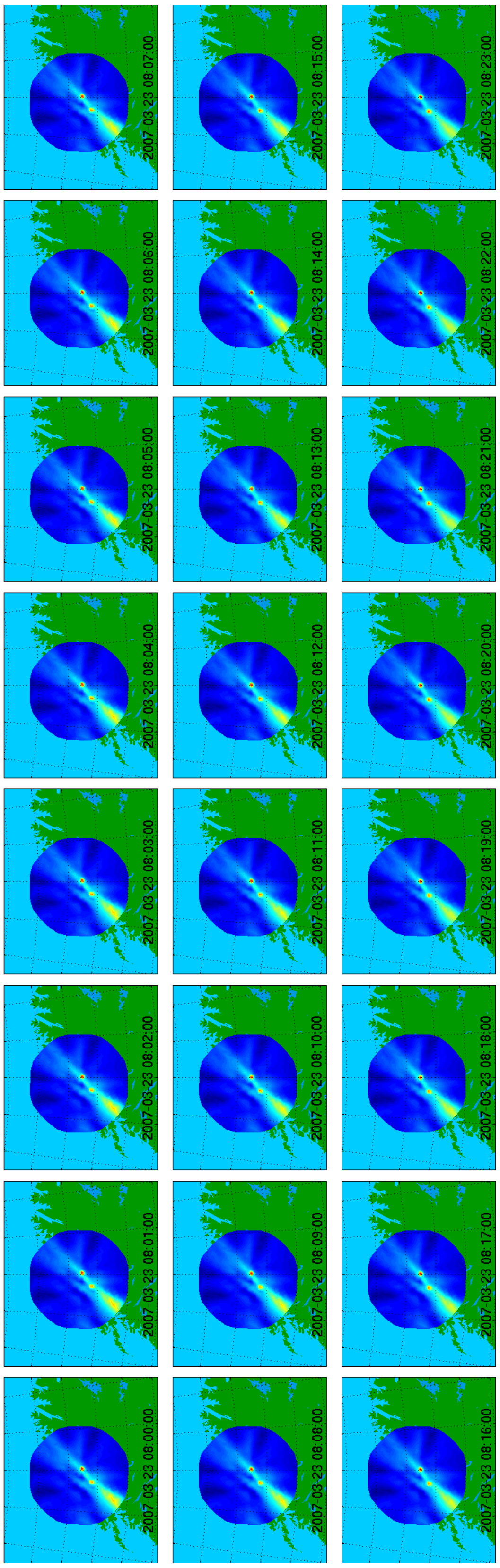


(a)

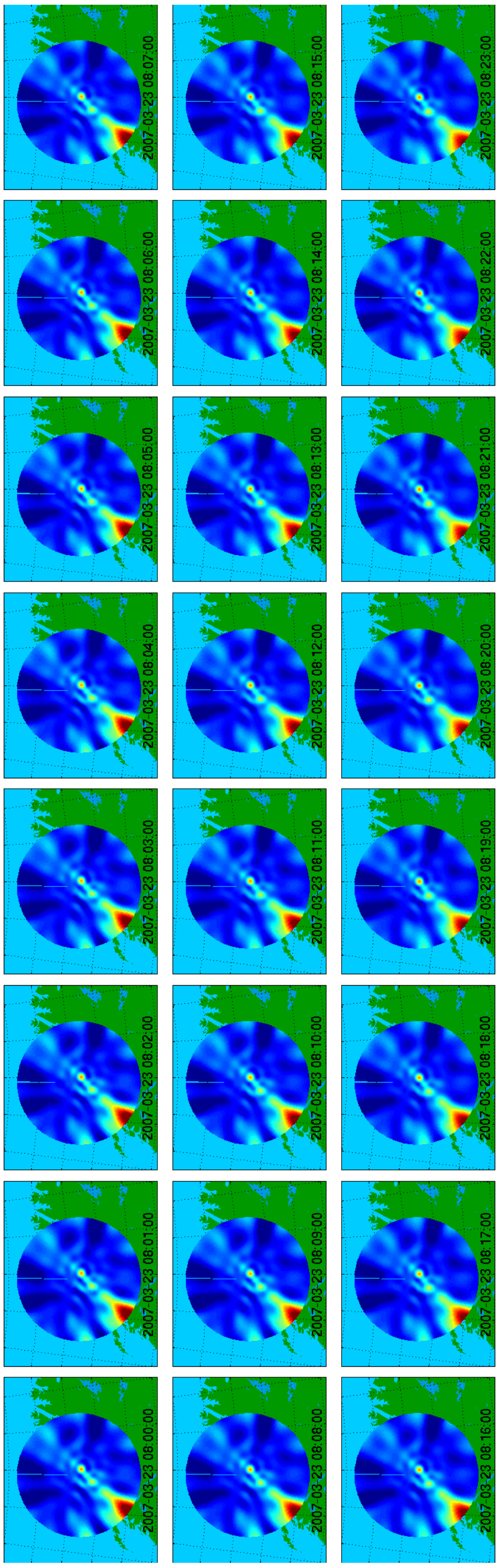


(b)

Figure 10.19: Sequence of ARIES power images for 2007-03-23. (a) Traditional interpolation algorithm, (b) GLEAM interpolation algorithm.

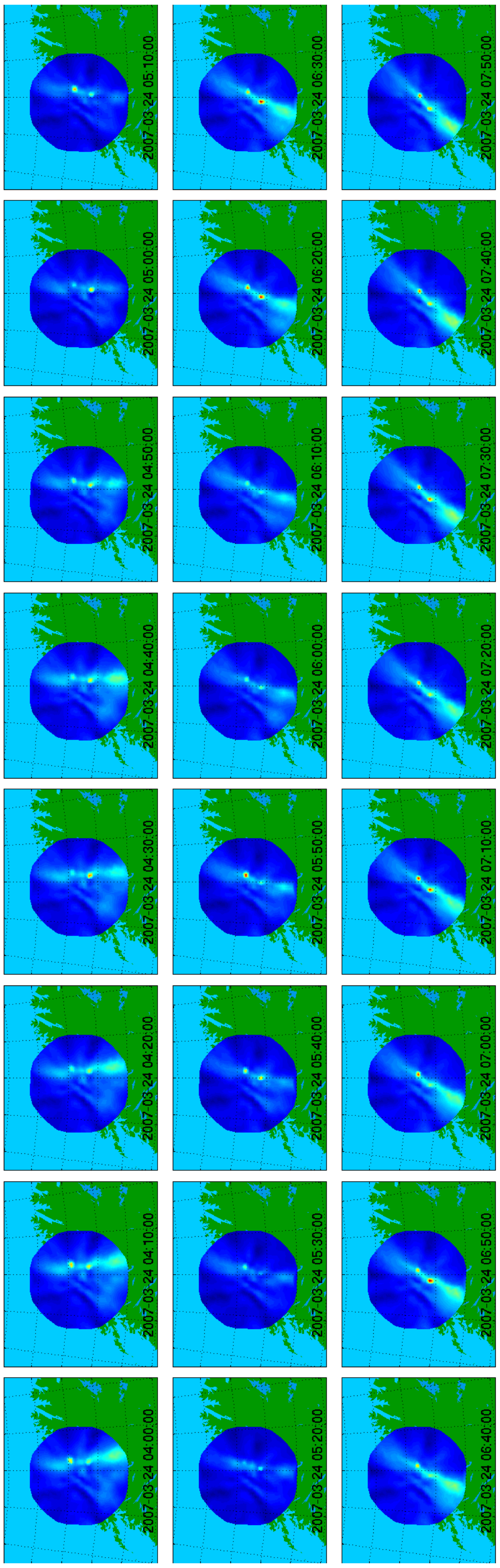


(a)

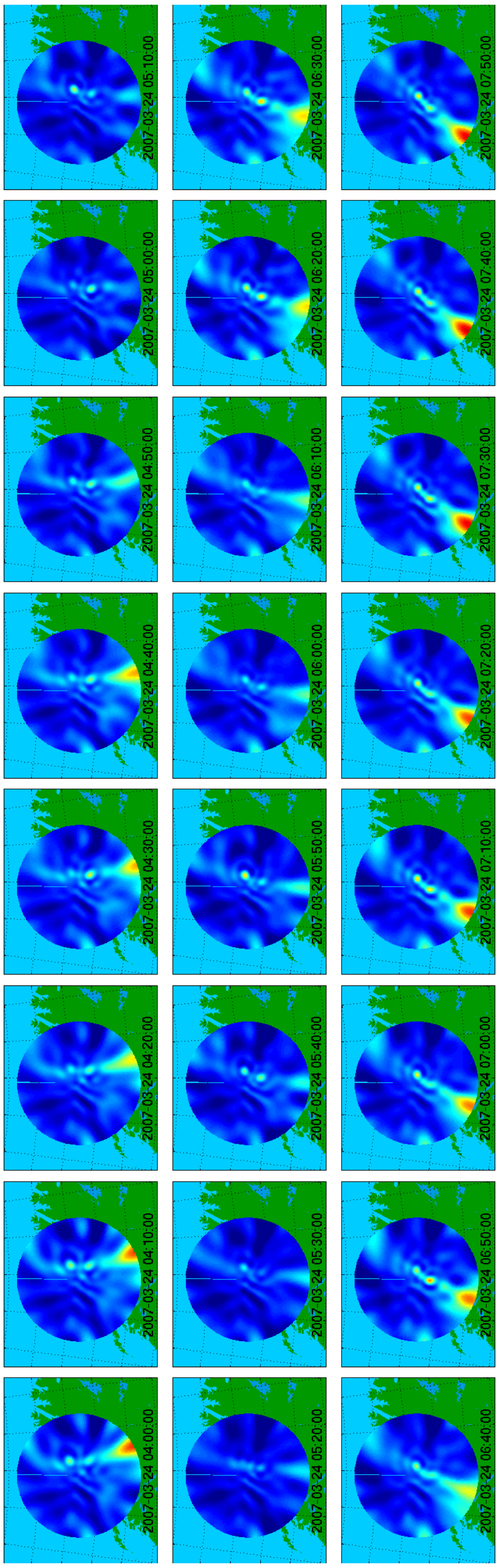


(b)

Figure 10.20: Sequence of ARIES power images for 2007-03-23, temporal zoom for 8:00h–8:23h. (a) Traditional interpolation algorithm, (b) GLEAM interpolation algorithm.



(a)



(b)

Figure 10.21: Sequence of ARIES power images for absorption event 2007-03-24 4:00h–8:00h. (a) Traditional interpolation algorithm, (b) GLEAM interpolation algorithm.

Chapter 11

Summary, Conclusions and Outlook

This chapter summarises the thesis. Many different research areas, engineering disciplines and ideas have been touched upon during the course of this thesis. In the effort of making the work as self-contained as possible, many of these ideas have been found to protrude over the set boundaries in both time and space. Therefore, this chapter also contains various ideas, suggestions and directions for future activities, improvements and research, demonstrating how the work presented in this thesis can serve as a foundation for exciting future developments in riometry and beyond.

11.1 Riometer Simulation Toolkit

The preceding chapters followed the development cycle of the Advanced Rio-Imaging Experiment in Scandinavia (ARIES), a new type of imaging riometer based on a Mills Cross antenna array. Such an array antenna achieves an angular (spatial) resolution identical to that of a filled phased array, but with significantly fewer antenna elements. In the case of ARIES, only 63 antenna elements are needed due to the Mills Cross design as opposed to 1024 for a filled phased array with the same resolution. Simulations both at signal level and at a more abstract level concerned with the radiation (reception) properties of the antenna system and beamforming network were executed, showing the feasibility and expected behaviour of such a riometer system. Low-level simulations served to explain and verify every step of the reception process, tracing signals all the way from the noise sources in the sky through to the cross-correlator (chapter 4). A separate set of simulations (chapter 5) was specifically aimed at determining achievable integration times and it was found that useful integration times are indeed achievable, an outcome that was later confirmed by experimental results (chapter 9).

The simulations led to the development of a universally applicable riometer simulation toolkit, RIOSIM. This toolkit was subsequently used throughout the thesis for simulating and visualising high-level concepts such as beam outlines (footprints), radio star footprints and three-dimensional representations of complex-valued radiation patterns. It forms the basis for more specific tools such as a theoretical quiet-day curve generator (section 7.4), a radio star tracker (section 7.2) and the generic scintillation calculator described in sections 7.7 and 7.8. RIOSIM was used to predict (and confirm) ARIES beam alignment during the various prototyping stages. Especially the ability to predict not only the influence of the continuous sky background but also of the predominant radio stars (point sources) allows for very accurate confirmation of beam pointing directions. RIOSIM is also essential for simulating ARIES reception for the investigations of the new riometer image interpolation algorithm (GLEAM) in chapter 10, as it forms the basis for the algorithm's knowledge about the instrument's beam patterns as used in the GLEAM solver.

It should be noted that RIOSIM is in no way specific to one particular riometer, such as ARIES, or even to riometers in general, but is fully modularised around the basic concepts of radiation patterns, radio stars and sky maps (section 6.2). It is therefore capable of simulating the behaviour of arbitrary antenna arrays.

For example, RIOSIM's support for *FEM simulated beam patterns* (section 6.3.10) opens up new opportunities of replacing theoretically calculated radiation patterns with individually FEM-simulated beam patterns for each individual antenna beam. This has the potential to result in a more accurate digital representation of real riometer system radiation patterns, therefore potentially leading to improved results for all algorithms and analyses that rely on knowledge of beam shapes, in this thesis particularly the GLEAM fitting process (chapter 10), beam footprint plotting (section 7.1) and derived algorithms such as quiet-day curve (QDC) generation (section 7.4) and scintillation prediction (section 7.7). Realistic FEM-modelling of riometer beams, especially for multi-beam instruments, requires significant processing power and modelling effort.

With RIOSIM, future hardware improvements will be able to feed back directly into the simulations. For example, improved hardware capable of *measuring receiver phase offsets* can directly improve the accuracy of received power simulations and therefore also the more advanced applications such as GLEAM: In chapter 10 we noted that the 'earlobe' plots of complex ARIES received beam power generally show good agreement between simulated and real recorded values. Nevertheless, discrepancies of up to about 30° in phase between real recordings and simulated readings are evident for some beams, for example MIA beam 947 in figure G.7. It is likely that this can be explained with non-ideal behaviour of the receivers. Modifications to the FPGA part of the system to directly feed the calibration signal through to the cross-correlator without going through the beamforming process will allow measurements of these phase offsets and their behaviour with time and temperature, ultimately allowing us to compensate for these effects by taking them into account at the beam pattern simulation stage.

An implementation of an asynchronous processing framework providing remote access to RIOSIM applications was presented in section 7.8 for the example of the RIOSIM scintillation calculator and this was found to be quite cumbersome. The existing web-based Space Plasma Environment and Radio Science (SPEARS) group data request facilities are described in appendix H, along with some suggestions as to how these could be re-implemented for a *new generation data request system* based on recent developments and requirements and the conclusions drawn from section 7.8.

In addition to making use of FEM-simulated radiation patterns, *real beam pattern analyses (beam shape measurements)* appear possible and such results can then be fed back into RIOSIM's beam pattern descriptions: Riometer observations rely on knowledge about the riometer's beam shapes (see chapters 2 and 3). GLEAM relies heavily on accurate beam descriptions. So far, theoretical and modelled beam radiation patterns have been used. For the proposed investigations, the primary goal is to verify the theoretical beam pattern of the riometer in question using actual measurements. The secondary goal is to then use the information from these observations to derive a more accurate beam pattern by adjusting the parameters determining the theoretical radiation pattern shape to fit the observations as closely as possible. This improved, i.e. more accurate, beam pattern can then be used as a basis for cleaning the raw image data with algorithms such as GLEAM more effectively. One idea to achieve this cost-effectively is to use radio stars passing through the beams as calibration point sources. For example, the amount of scintillation observed for a given radio star at any given time relates to beam sensitivity in that direction. To cover all possible (although admittedly still rather limited) locations of the radio stars, a dataset of one day's worth of data is needed. At first thought, this does not necessarily have to come from a contiguous period of time. However, since we are interested in determining the beam shapes as accurately as possible, it seems adequate to eliminate as many sources of error as possible. Also, long-term IRIS recordings suggest that the beam shapes change with certain environmental parameters, especially snow depth. A constant snow depth cannot be guaranteed for long periods of time. Later on, if several datasets from different times of the year are found to be suitable, it may even be possible to compare the derived beam patterns and get a first quantitative measurement of how the beam patterns change with the seasons.

11.2 Advanced Riometer Operating Software

Operating ARIES both during its various prototype stages and in its final system configuration necessitated the development of a flexible operating software, ARCOM, see chapter 8. The structure (architecture) of this software was designed to address the 'Wicked Problem' of an ever-evolving system. The software is based around a set of interchangeable components, connected into a multi-branched pipeline through block-based multi-client shared memory interfaces and individually controlled through CORBA command interfaces. Three basic types of components exist. Recorders retrieve data from a hardware device such as the ARIES FPGA or a network of temperature sensors and feed it into a processing pipeline. Processors manipulate (process) data, for example by post-integrating or archiving. Adaptors provide a native ARCOM CORBA command interface for existing third-party hard- and software like uninterruptible power supplies (UPS) or watchdog timers. This structure is supported by a streaming data format not unlike the one used in digital television (DVB). Together with several low- and high-level tools for day-to-day operation and data maintenance, ARCOM can readily be tailored to support a wide range of data acquisition and processing tasks for a variety of instruments. ARCOM is therefore not limited to ARIES, or even riometers. Since its inception, it has in fact also been deployed for the Advanced Imaging Riometer for Ionospheric Studies (AIRIS). Other instruments have also benefited from ARCOM concepts, for instance the new high-speed photometer for optical emission measurements (SPARKLE) developed by the author, which employs a packet-based streaming data format very similar to the one used by ARCOM.

In the field of riometry, the fully modular design of ARCOM achieves unprecedented flexibility and expandability with exciting new possibilities emerging as communication infrastructure improves. Originally only linked to the wider internet by a low-speed dial-up internet connection, with more and more instruments having access to 'always-on' internet connectivity, *live data feeds* are becoming a possibility, even for permanent use, not just for temporary campaign use as used to be done over dial-up connections.

Such live data feeds can feed into forecasting, or at least ‘nowcasting,’ systems, for example early-warning systems for space weather effects. The ARCOM architecture readily supports this kind of application (see the ARCOM component diagrams in chapter 8, for example figure 8.7), and some possible usage scenarios have been outlined as examples in the deployment diagrams in figure 11.1. Panel (a) shows a basic configuration with the main PC taking over the additional role of live broadcaster through an extra ARCOM component linked into the running system. This component simply filters out specific (beam) data of interest and broadcasts these directly to recipients of the live data feed in the form of ARCOM packets encapsulated into User Datagram Protocol (UDP) packets. A post-integrator component inserted between filter and transmitter can potentially post-integrate the raw 1s values to longer integration times.

In this scenario, the task of creating higher-level data products (for example absorption time series or images) is left to the recipient, which may or may not be an ARCOM-based system.

For many recipients and/or further data processing, a scenario like in panel (b) is more suitable. The main PC only ever connects directly to a secondary ‘server’ node located on the UK internet backbone. This server, running a separate ARCOM installation, takes over the tasks of post-processing and broadcasting the data, ensuring that the control PC does not get overloaded and reducing the security risks associated with the control PC directly connecting to outside nodes beyond the control of the system operators.

The scenario in panel (c) is a variation of (b) geared towards maximised (both on-site and off-site) availability of useful data products in real-time. The main control PC transmits live power data to a secondary processing node on-site. ARCOM components on the processing node maintain a local data archive, calculate quiet-day curves and images at appropriate resolutions, display this information for local users and transmit summary information across a network link to further remote processing and/or live viewer nodes, depending on availability and speed of the connection. As in scenario (b), the off-site processing node is responsible for data dissemination and definitive archival. Additional services such as video streaming to the WWW and web data requests can also be run on this or additional co-located nodes. Again, processing overhead on the main control node is kept to a minimum.

All scenarios are readily supported by the ARCOM architecture. Much of what is suggested here had actually originally been devised as suitable configuration for deployment, but later got abandoned due to limited funding and hardware availability at the time.

Following on from the discussion above, fat-client support for *ARCOM streaming on- and off-site live visualisation* is an important area of future work. An initial demonstrator in form of a Java application ‘`mon`’ has already been developed, allowing users one-click access to a ‘rich media’ experience of live ARCOM data streams through their web browser in combination with Sun’s Java Web Start [Sun05] technology. Rolling out such a solution to the community or even the general public might result in significant publicity and contribute to making science (and particularly riometry) more approachable. For load-balancing reasons, i.e. to guarantee that instrument operation remains unaffected no matter how many users connect in through the Web Start interface, such a configuration will require a multi-node setup such as the ones suggested above, see figure 11.1 panels (b) and (c). The current demonstrator only supports basic beam projection plotting for incoming ARCOM data streams, but could be expanded to display any information contained in the data stream, including instrument health, environmental temperature monitoring, live surveillance camera images, etc.

In future, with the availability of even more powerful signal processing hardware, ARCOM will also readily support *next level digital beamforming*. For ARIES in its current incarnation, the analogue beamforming matrix (Butler Matrix) to form the fan beams from each arm of the Mills Cross has already been replaced by a purely digital implementation on an FPGA. Nevertheless, this configuration means that the positions of the fan beams are fixed. It may be beneficial to steer the fan beams so as to avoid interference from the strongest radio stars. Beam steering can in principle be achieved with digital beamforming techniques through appropriate parametrisation. Real-time closed-loop control of beam pointing directions with null tracking will require additional implementation work on the FPGA side and a new component on the ARCOM high-level software side.

11.3 A Novel Approach to Riometer Image Interpolation

Initial raw ARIES recordings were presented in chapter 9. Raw recorded data is not necessarily the most useful data for subsequent scientific analysis, and as a first step towards higher-level data products, the need for image interpolation and the traditional interpolation method as used for, e.g., the IRIS riometer, were introduced. This led to the development of a new type of riometer image interpolation algorithm (GLEAM) that, by inherently knowing about instrument specifics such as complex beam directivity patterns, fits the real recorded data to a model of the actually observed sky brightness distribution. GLEAM results were presented in the form of image sequences, and these were also compared to images generated by the traditional method. Even without very specific prior knowledge, such as simulated received power based on synthetic sky maps, the GLEAM approach delivers performance that alleviates some of the shortcomings of the traditional approach. In particular, GLEAM interpolated images are no longer limited to the convex envelope of the beam centres used during the interpolation, effects of negative correlation and sidelobes due to the Mills Cross beamforming principle are reduced and there are no interpolation artefacts at the borders of the image. Interpolated images such as the ones produced by GLEAM lead the way to lessening the emphasis on individual beams, with images (i.e. power values in arbitrary 'virtual beam' directions) becoming the primary data product of next generation riometers.

Availability of these image sequences can then lead to *quiet-day curve (QDC) and absorption calculations based on images (virtual beams)*: With a next generation riometer outputting sky brightness in arbitrary 'virtual beam' directions, existing algorithms can be used to derive quiet-day curves (QDCs) and therefore absorption for these directions, see for example the discussion in section 3.1. In addition, the notion of a quiet-day sequence of images (a quiet-day movie) is now meaningful, and new methods can be developed to produce these three-dimensional (two spatial dimensions plus time) descriptions of quiet days, interpolating, weighting and averaging the raw brightness data in all three dimensions as appropriate. These three-dimensional methods promise higher accuracy QDCs that adjust better to environmental changes and eliminate the need for operator-verified, fixed quiet-day curve periods of, say, 14 days, such as currently used with IRIS. They will help to further reduce the need for manual intervention, thereby providing more objective and repeatable absorption measurements. A basic version of a three-dimensional method was implemented for producing the mean data plots in figures 10.18 and G.1–G.7, albeit still at (real) beam level. See the description in section 10.7.1.

Further *improvements to GLEAM* might eliminate some of the issues encountered in chapter 10 and are readily supported by GLEAM's object-oriented architecture. As pointed out in chapter 10, a certain amount of 'ringing' seems to be inherent to any image generated by GLEAM. Nevertheless, to further improve the performance of the GLEAM algorithm and minimise these effects, several approaches suggest themselves:

Instead of feeding post-integrated data into the GLEAM solver (as done in chapter 10), it might be beneficial to *feed the raw 1s data into GLEAM* and then post-integrate the modelled result generated by GLEAM. This should potentially make the GLEAM interpolated images smoother and minimise the differences from frame to frame. As an additional experiment, the calculated coefficients could also be averaged before calculating the corresponding brightness distributions, though this is likely to be inferior to post-integrating the resulting image, as each coefficient affects all parts of the interpolated image.

Additional prior knowledge might be devised. As pointed out in section 10.6.4, this is likely to depend on the type of underlying basis functions used.

A suitable implementation of a '*capped harmonics*' functor might help to maximise the achievable level of detail for any given number of coefficients (see figure 10.5). [Hai88], [Hai85] and [ST97] might be useful starting points.

Using received power *information from all beams* (even the degraded ones) might contribute to a further slight increase in interpolated image quality, albeit no major improvement should be expected from this as these beams contain little information, see the comparison between the 556-beam and 716-beam cases in table 10.1.

As suggested by [Sen], altogether different methods of modelling the sky brightness distribution may also be investigated. This includes *splines* and the *maximum entropy method*. Useful starting points are [PFTV88], [Rei85] and [SK05]. Note that these methods will require an iterative approach, which is inherently different from the matrix inversion approach as used by GLEAM.

Although not detailed further in this thesis, due to the underlying RIOSIM toolkit, it is straightforward to *apply GLEAM to data from other riometers*. This has already been done for IRIS data during initial development of GLEAM as published in [GSH05]. At the time, results were less than satisfactory. With the current capabilities of incorporating prior knowledge, considerable improvements can be expected. Applying GLEAM to 256 beam (16×16 aerial) filled array riometers will also be an interesting investigation.

The availability of good quality images from multiple riometers will enable *absorption height triangulation*: Using absorption data from two or three imaging riometers, the height profile of the absorbing regions can be derived by means of triangulation and/or tomography. This is one of the scientific goals of ARIES. With the availability of interpolated images covering wide fields of view, methods that have successfully been employed for optics in the past (see for example [AKK05]) can be applied to imaging riometer data from instruments with overlapping fields of view.

By bringing together knowledge from various areas of Engineering and Physics, it is hoped that the multi-disciplinary developments presented in this thesis will contribute to advancing instrumentation and analysis methods in riometry and beyond, thereby making valuable contributions to, and providing foundations for, a wide range of future research.

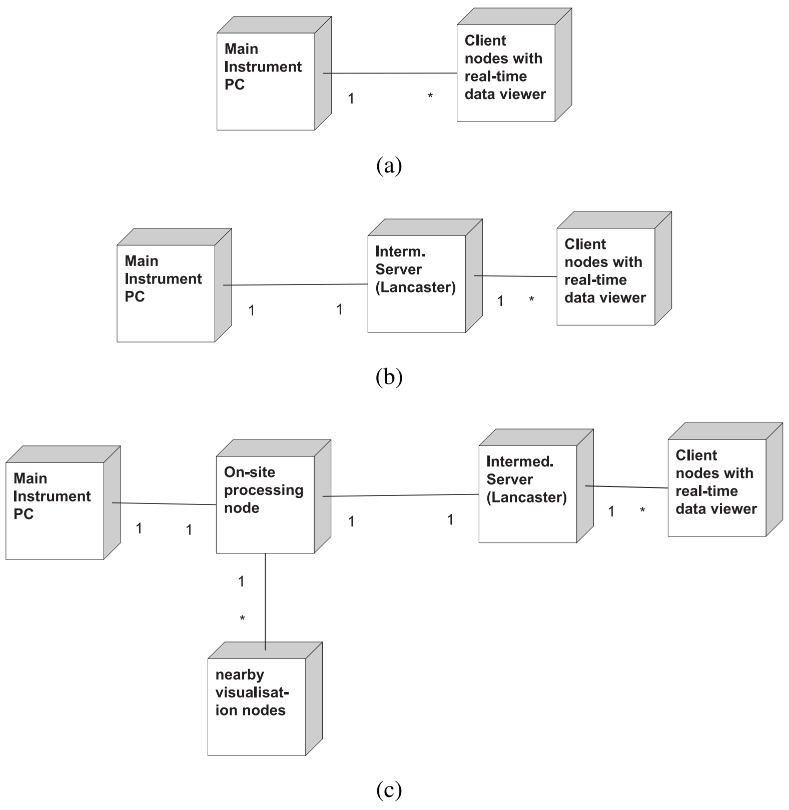

Main Instrument PC
1
*
Client nodes with real-time data viewer
(a)
Main Instrument PC
1
1
Interm. Server (Lancaster)
1
*
Client nodes with real-time data viewer
(b)
Main Instrument PC
1
1
On-site processing node
1
1
Intermed. Server (Lancaster)
1
*
Client nodes with real-time data viewer
1
*
nearby visualisat-ion nodes
(c)

Figure 11.1: Advanced ARCOM deployment examples supporting real-time data feeds

Appendix A

Glossary

absorption	Attenuation of an incoming signal due to the properties of the medium, e.g. the ionosphere. Absorption varies with frequency of the signal as $\frac{1}{f^2}$.
ABC	Abstract Base Class
abstract (base) class	A term used in object-oriented programming for classes that cannot be instantiated. Abstract classes usually form the base for more concrete implementation classes.
aka	also known as
ARCOM	Advanced Riometer COMponents. Operating software developed for advanced riometer systems, based on a componentry framework and high-speed shared memory interfaces for component interconnection. See chapter 8.
CBR	Cosmic Background Radiation
CGI	Common Gateway Interface. A standardised way of passing information from a web server process to third-party programs and back in response to an HTTP request from a client.
CMB	Cosmic Microwave Background
COM	Component Object Model. A framework for software componentry mainly used by Microsoft Windows operating systems.
componentry	In Software Engineering: A design principle for complex software systems. A software component is an independent software unit that can be composed with

	other components to create a software system [Som04]. Examples of popular componentry frameworks are Microsoft's COM and OMG's CORBA. Components usually try to encapsulate a highly coherent set of functionality, while themselves only exhibiting loose coupling to other components.
concrete class	As opposed to an abstract (base) class. A class that can be instantiated to create a useable object.
CORBA	Common Object Request Broker Architecture. A vendor-independent architecture and infrastructure for software components to work together over networks [HV99, PR00, Obj05].
crossed dipole	see 'turnstile antenna'
DUNES	Dial-up NEtworking for remote Stations. A piece of software developed by the author to access remote nodes via telephone modem, supporting cost-effective operating modes that dial up to local internet service providers (ISPs) instead of relying on long-distance telephone connections. See appendix I and [Gri06b].
FEM	Finite Element Method. A method of solving complex mathematical systems by dividing them up into a number of finite elements and iteratively calculating a numerical solution. Used, for example, in the NEC (Numerical Electromagnetic Code) program to analyse the electromagnetic response of an arbitrary structure consisting of wires and surfaces in free space or over a ground plane [BP77].
footprint	The area covered by a given antenna beam on the surface of interest. Satellite footprints describe the area on the surface of the Earth where reception is possible. In riometry, riometer beam footprints are often projected onto the ionosphere to show the spatial extent of any given beam.
functor	In object-oriented (OO) programming: An object that can (only) provide a well-specified piece of functionality, for example evaluating a function (as represented by the functor in question) at specific points. Often implemented using polymorphism.
GMST	Greenwich Mean Sidereal Time. See section C.4.
great circle	In spherical trigonometry: A circle (on a sphere) whose centre coincides with the centre of the sphere. Examples of a great circle are the equator or any line of equal

	longitude.
Grid	A collaborative network, which lets people share computing power, databases, instruments and other on-line tools securely across corporate, institutional, and geographic boundaries without sacrificing local autonomy. (Definition taken with slight modifications from [Uni07].)
Gridservice	As opposed to a simple Webservice, a Gridservice is usually a stateful service provided by a processing Grid, and initially instantiated by the client using a service factory. Each client will receive (a reference to) its own instance of the requested Gridservice.
HTTP	Hypertext Transfer Protocol. A set of rules for exchanging files, heavily used on the World Wide Web (WWW) and more recently for Web- and Gridservices.
IDL	(1) Interface Definition Language. A generic term for specifying messaging interfaces. More specifically, in CORBA, a C++-like language for specifying CORBA interfaces. (2) Interactive Data Language. A data visualisation and analysis software by RSI, Inc., similar in features to MATLAB.
ionosphere	The part of the Earth's atmosphere ranging from a height of about 70km up to 1000km. The properties of this part of the atmosphere are determined by the fact that the gas atoms and molecules are ionised. The ionosphere as a whole is still electrically neutral, but ionisation enables the flow of electric current. Ionisation in the ionosphere is mainly caused by solar radiation and the interaction of the solar wind with the Earth's magnetic field (magnetosphere). See section 3.4.1.
IOR	Interoperable Object Reference. The CORBA term for an identifier that uniquely identifies any given CORBA object (component) at run-time.
IRIS	Imaging Riometer for Ionospheric Studies. An imaging riometer with 8×8 antenna elements [BHH95]. In particular the riometer operated by the Ionosphere and Radio Propagation Group at Lancaster University [IRI].
isotropic radiator	An imaginary antenna that emits unpolarised radiation with equal intensity in all directions.

keogram	Time-space-diagram used to show how a slice of (image) data varies with time. Commonly used to identify movement of structures through the field of view of an imaging instrument.
LAST	Local Apparent Sidereal Time. See section C.5.
LUT	Look-up table. A table that associates a given set of input values with certain output values. LUTs are often used to speed up processing: all possible results are precomputed and then stored in a LUT instead of computing them on-the-fly. A `CDelayBuffer` object (see 4.2.1.2) is an implementation of a linearly-interpolating LUT.
MATLAB	A data visualisation and analysis software by The Mathworks, Inc [Matb]. Extensively used in the SPEARS group at Lancaster University, especially for MIA. See also IDL.
metadata	Data describing data. For example, a time series of data points on its own only contains the data itself. Metadata is used to describe what this data is about. For example, metadata could describe that this series of data points represents absorption data at a time resolution of 5 seconds starting at midnight on 5 May 2003. One of the reasons why XML has turned out to be so popular is that it tries to directly associate data and metadata inside the same file.
MIA	Multi Instrument Analysis Toolkit [Marc, MH04]
NaN	Not a Number. A numerical value to represent the fact that there is no numerical value for the value in question. Often used to indicate 'missing data.'
obliquity factor	When a ray (for example a radio wave) traverses a layer of matter in a non-perpendicular direction, that layer appears thicker than it really is. Therefore, absorption measured in some off-zenith direction needs to be corrected for this oblique viewing angle. Several methods exist for this correction. One method is to simply calculate the ratio of apparent thickness to real thickness for the main pointing direction (see chapter 10, section 10.3). Advanced methods try to take the shape of the receiving antenna beam pattern into account ('effective obliquity factor'), see for example [HD02].

OO	Object-Orientation. Also object-oriented design. An approach to software design that seeks to divide (software) systems up into interacting 'objects,' each with its own capabilities (responsibilities) and properties (attributes). Objects share common functionality by means of inheritance. Basic design rules of object-oriented programming are that each class of objects should encapsulate a well-defined slice of functionality and that interfaces between objects should be clearly defined and enable loose coupling (little interdependency) between objects.
pABC	purely Abstract Base Class. An abstract base class containing no implementation code whatsoever. The C++ equivalent of a (Java) 'interface.'
phase centre	(of an antenna or antenna array.) An (arbitrary) point in space used as a reference from which all relative phase offsets for signals received from the antenna(s) are calculated. To correctly combine the radiation patterns of two (arrays of) antennas, the phase centres need to coincide.
polymorphism	In object-oriented (OO) programming: The fact that objects of classes that are derived from a common interface appear identical to external entities, therefore allowing them to be swapped in and out at run-time.
Prime Meridian	The meridian that passes though Greenwich, UK. This meridian defines the origin of longitude.
QDC	Quiet-Day Curve. The QDC represents the power level that is received during one sidereal day on a perfect 'quiet day,' i.e. when no absorption occurs. There are several ways to derive a QDC, for example the inflection point method as described in [KDR85] or the empirical method used by IRIS. See also the discussion in section 10.2.3.
real-time	Often further divided into soft and hard (stiff) real-time. A hard real-time system places stringent timing requirements on the processing of data passing through the system, whereas soft real-time systems have more relaxed timing requirements where occasional delays are non-fatal. Examples of hard real-time systems are CNC (computerised numeric control) manufacturing machines, where processing delays would result in extensive damage to the machine. An example of a soft real-time system is a domestic central heating controller.

riometer	Relative Ionospheric Opacity Meter. Measures absorption of cosmic radiation in the ionosphere. See chapter 3.
riometry	Science to do with riometers.
RIOSIM	A toolkit for simulating riometers developed in this thesis, see chapter 6.
scintillation	A term for rapid variations in apparent brightness of a distant object when viewed through a medium such as the atmosphere or ionosphere. Caused by refraction due to small-scale variations in the medium density. See for example [Ric77].
sidereal time	Time with respect to the stars: sidereal time uses stars outside our solar system as a fixed point of reference. The Earth rotates around its axis in one sidereal day. A sidereal day is about 4min shorter than a 'regular' day due to the fact that a 'regular' day is measured with respect to the sun, and not only does the Earth rotate around its axis, it also rotates around the sun, thereby taking slightly longer for a 'complete' revolution with respect to the sun.
SVG	Scalable Vector Graphics. An XML-based file format for the description of vector-based drawings. Will be read natively by next generation web browsers. Currently supported through external applications and/or plugins.
TCP/IP	Internet Protocol: Transport Control Protocol. A connection-oriented protocol used to transfer data over potentially unreliable networks and guaranteeing that all data arrives unharmed and in its original order.
turnstile antenna	Two linear dipole antennas perpendicular to each other, connected with a 90° phase shift between them. Also often referred to as 'crossed dipole.'
UDP	User Datagram Protocol. A connectionless but fully routable internet protocol. Its low overhead makes it ideal for real-time data dissemination.
UML	Unified Modelling Language. A way of describing systems graphically in an object-oriented way. The UML defines several standard diagram types such as Class Diagrams and Sequence Diagrams. The UML was originally designed by James Rumbaugh, Ivar Jacobson and Grady Booch (aka 'The Three Amigos'), all three widely recognised for their contributions to the development of object-oriented technology [BRJ00, RJB99].

VPN	Virtual Private Network. Several Computers that are logically connected to form a secure, private network. There is no dedicated physical connection between the hosts. Instead, some public network (e.g. the internet) is used to virtually connect the hosts. Communication on such a network is usually encrypted to ensure privacy.
Webservice	A simple service accessed through the standard HTTP (web) protocol. A Webservice usually provides relatively simple information in response to one single query by the client. No state information is usually retained over multiple requests. See also Gridservice.
Wicked System	In Software Engineering, the term ‘Wicked System’ is used to describe complex systems that are conceptually difficult to design and implement. In this context, the term ‘wicked’ refers to the fact that the requirements for such a system are not well-defined, and any given implementation is likely to in turn change the requirements and therefore the system design [Som04].
Wiki	A WikiWikiWeb system as originally devised by Ward Cunningham [LC01]: “The simplest online database that could possibly work.” A collaborative WWW-based platform allowing any visitor to view and edit pages using a simple markup language.
XML	Extensible Markup Language. A flexible text format developed by the World Wide Web Consortium (W3C). XML is playing an increasingly important role in the exchange of a wide variety of data on the web and beyond (Webservices, Grid) [Qui03].
zenith	The point directly overhead.

Appendix B

Astronomical Coordinate Systems

B.1 Astronomical Coordinate Systems

During the course of this work, several different coordinate systems had to be used. This section discusses some details about the different coordinate systems that are used in Astronomy. Unfortunately, there is quite a variety of them, and many ambiguities can arise. Different coordinate systems are used in different fields of Astronomy. As the riometer model needs to know about locations on Earth as well as in the sky, it is essential to be familiar with the different coordinate systems in use, along with knowledge of how to convert between them.

This appendix briefly describes what coordinate systems there are, and what they are used for. The ones that are directly relevant for the work to follow are then explained in more detail, together with explanations on how to convert between them.

Note that most of the coordinate systems presented are spherical coordinate systems, describing a position on the surface of a sphere, as opposed to three-dimensional coordinate systems. This is because Astronomy often deals with objects that are so far away from the observer, that the actual distance is no longer of interest, but only the direction in which the object in question can be found. In that case, dealing with — two-dimensional — spherical coordinates helps reduce the complexity of the problem. However, there are benefits in using real three-dimensional coordinate systems, these include a more consistent behaviour that can be described with vector equations, and constant precision over the whole space [Vin98]. More details will be discussed in the descriptions of the different coordinate systems.

The information in this appendix has been collated from several different sources, the most important ones are [Vin98, Walb, Sma62].

Table B.1 gives an overview over the most commonly encountered coordinate systems in Astronomy. Not all of those systems will be described in detail later on, because not all of them are particularly relevant for this work. It seems reasonable, however, to at least introduce the basics of all these commonly used coordinate systems.

B.1.1 Common Basics

All of the coordinate systems in table B.1 need some means of specifying their principal position in space. The coordinates then describe the position relative to those principal components. In the case of Cartesian coordinates, this principal construction is the unit vector triple $\{e_x, e_y, e_z\}$. These vectors are perpendicular to each other. The triple $\{e_x, e_y, e_z\}$ is called a triad, and a vector to any position in three-dimensional space can be expressed as a linear combination of these three principal vectors.

Similarly, all the spherical coordinate systems in table B.1 are based on a principal circle, and a fixed point on that circle. The coordinates then describe the position of an arbitrary point on the sphere relative to those principal components. Thus the main difference between all the coordinate systems in table B.1 is — apart from different naming conventions — that their fundamental components are different.

The following sections will discuss the relevant spherical coordinate systems in more detail. In particular we will define the spatial relation between each of the spherical coordinate systems and the Cartesian coordinate system. This will enable us to convert coordinates from one system to another, using the Cartesian coordinate system as an intermediate. Direct conversions from one spherical coordinate system to another are, of course, also possible, and these will be applied and derived where applicable. The main benefit of converting from one spherical coordinate system to another directly is speed, as there is only one (generally less computationally intensive) step involved, as opposed to two steps for converting to Cartesian coordinates and then to the target spherical coordinate system.

However, using the Cartesian coordinate system as an intermediate allows us to perform the same set of operations on coordinates that were originally specified in different coordinate systems, without the need to re-implement the necessary algorithms. One example is rotation around the origin by an arbitrary amount. This can be implemented in Cartesian coordinates with a simple matrix multiplication.

It has to be noted that even with the two-step coordinate conversion using an intermediate

Name	basic units	used to describe the position of...	centre of coordinate system
Cartesian	x, y, z	points in space	arbitrary fixed origin
mathematical	azimuth θ, elevation ϕ	points on a sphere	centre of sphere
electromagnetic	azimuth angle ϕ, polar angle θ	points on a sphere	centre of sphere
geographic	longitude, latitude ϕ	terrestrial objects	centre of the Earth
geomagnetic	geomagn. longitude, geomagn. latitude	terrestrial objects	centre of the Earth
horizontal	altitude a, azimuth A	celestial objects	observer
first=local equatorial	HA h, declination δ	celestial objects	centre of the Earth
(second) equatorial	RA α, declination δ	celestial objects	centre of the Earth
ecliptic	ecl. long. λ, ecl. lat. β	celestial objects	(centre of the Earth)
galactic	gal. longitude l^{II}, gal. latitude b^{II}	celestial objects	(centre of the Earth)
supergalactic	SGL, SGB	celestial objects	(centre of the Earth)

Table B.1: Common coordinate systems

Cartesian coordinate system, conversion is in most cases not straightforward, because the relative position of the underlying Cartesian coordinate systems to each other are normally not constant, but need to be determined based on variables such as time of observation, position of observer, etc. [Walb]. Also, strictly speaking, some of the coordinate systems are not based on entirely orthogonal axes, but are slightly distorted [Walb]. For our discussion, this will not be considered, as we can achieve adequate accuracy without taking these properties into account.

The following sections will firstly describe the three general coordinate systems that are used for specifying the positions of points on an arbitrary sphere. We will then discuss coordinate systems that are used to describe positions on the Earth, moving on to coordinate systems describing the celestial sphere centred on the Earth. Finally, we will reach the coordinate systems that no longer rely on any of the properties of the Earth. The latter coordinate systems are mentioned in this place only for reasons of completeness, no further use will be made of them in this thesis.

The following explanations will make use of certain terms (great circle, zenith, etc.) that are explained in the glossary.

B.1.2 The 'Mathematical' Spherical Coordinate System

This coordinate system is widely used in mathematics. Mathematical software like MATLAB [Matb] usually includes functions for converting between the mathematical spherical coordinate system and the Cartesian system. See [Matb, functions cart2sph and sph2cart].

The mathematical spherical coordinate system is directly coupled to an underlying Cartesian system. These two systems share the same origin. Azimuth θ and elevation ϕ are then angular displacements in radians measured from the positive x-axis, and the x-y plane, respectively. A third variable, r, can be used to specify the distance from the origin to a specific point in direction (θ, ϕ).

This coordinate system is the basic spherical coordinate system for all software developed in this thesis.

B.1.3 The 'Electromagnetic' Spherical Coordinate System

This coordinate system is closely related to the mathematical one described in section B.1.2. The difference is, that in case of the 'electromagnetic' spherical coordinate system, the angular displacement from the positive x-axis is referred to as Azimuth angle ϕ, and θ is the angle measured from zenith downwards towards the x-y plane, referred to as Polar Angle [Kra88, p. 24]. This naming confusion can cause quite a lot of trouble. The 'electromagnetic' spherical coordinate system is mainly used in books on electromagnetics, with antennas often pointing in zenith direction, identified by small Polar Angles θ.

B.1.4 The Geographic Coordinate System

The geographical coordinate system is used to describe a position on the surface of the Earth or, in fact, on any celestial body. We will stick to the Earth for the following explanations.

The Earth spins around its axis. The North and South Poles are where this (imaginary) axis meets the Earth's surface, the equator is the great circle perpendicular to the axis, and therefore midway between the two poles. To describe a location on the surface of the Earth, we use latitude and longitude (two coordinates, because the surface is two-dimensional). A great circle through the poles and the location to specify, X, is called a meridian of longitude. The latitude of X is then the angular distance along this meridian from the equator to X, usually measured in degrees from -90° at the South Pole to $+90^\circ$ at the North Pole.

There is no obvious point of origin for measuring longitude; for historical reasons, the zero-point is the meridian which passes through Greenwich, England (also called the Prime Meridian). The longitude of X is the angular distance along the equator from the Prime Meridian to the meridian through X. Longitudes are usually measured from 0° to 180° East of the Prime Meridian and from 0° to 180° West, following the directions of the arrows in figure B.1. Sometimes, longitudes are also measured East or West 0° to 360°. Note that the geographic coordinate system only forms a right-handed coordinate system if longitude is measured Eastwards.

Note

Be aware that the geographic coordinate system discussed in this section uses the centre of the Earth as its origin. It does not necessarily assume that the Earth is an ideal sphere, but all measurements take place relative to the centre. This gives what is known as *geocentric* coordinates as opposed to *geodetic* coordinates which are based on the local vertical at the location of the observer, taking into account the flattening of the Earth.

Relationship to the Cartesian Coordinate System

For maximum similarity with the ‘mathematical’ spherical coordinate system, we define the relationship between the geographic coordinate system and the Cartesian coordinate system as shown in figure B.1: The x-axis points towards the intersection of equator and Prime Meridian, the z-axis points towards the North Pole.

B.1.5 The Horizontal Coordinate System

This is the simplest celestial coordinate system. It uses the horizon as seen by the observer as its fundamental circle. The poles are the zenith overhead and the nadir underfoot; these are defined by the local vertical, for example by using a plumb-line.

We can now draw a great circle through zenith, nadir and our target object X. The altitude of object X is then defined as the angular distance along this vertical great circle from the horizon to X, measured from $-90°$ at nadir to $+90°$ at zenith.

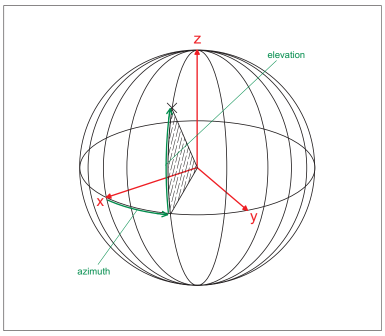
We then need a fixed point of origin on the horizon, before we can specify the azimuth of object X. This fixed point is attained by looking at where the spin axis of the Earth intersects the celestial sphere, the intersection points are the North and South Celestial Poles. The vertical (great) circle through these is called the principal vertical. Where this intersects the horizon, it gives the North and South cardinal points (the North point is the one nearest to the North Celestial Pole). Midway between these are the East and West cardinal points.

The azimuth of object X is now the angular distance around the horizon from the North cardinal point to the vertical (great) circle through X, measured from $0°$ to $360°$ westwards.

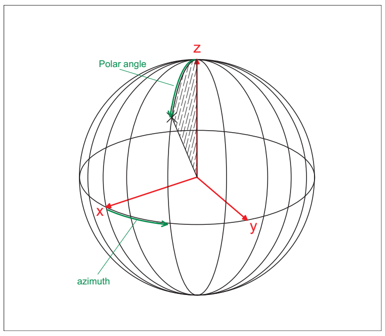
An interesting property of the horizontal coordinate system is the fact, that the altitude of the North Celestial Pole in the horizontal coordinate system is equal to the latitude of the observer specified in the geographic coordinate system.

Relationship to the Cartesian Coordinate System

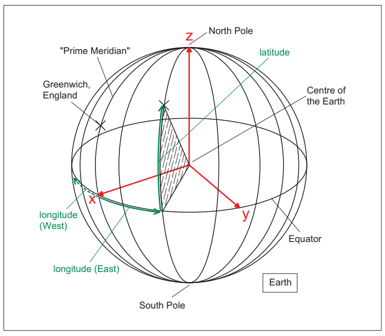
For maximum similarity with the ‘mathematical’ spherical coordinate system, we define the relationship between the horizontal coordinate system and the Cartesian coordinate system as shown in figure B.1: The x-axis points towards the intersection of equator and principal vertical, the z-axis points towards the zenith.



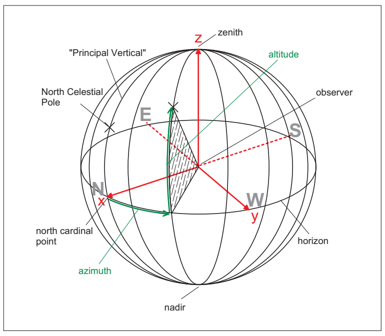
(a) mathematical



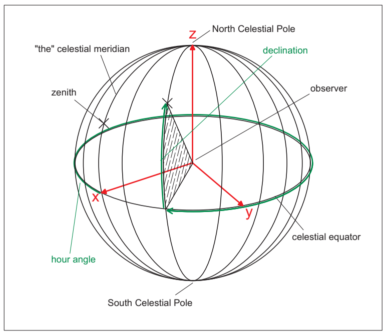
(b) electromagnetic



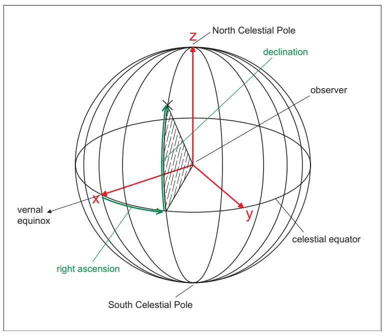
(c) geographic



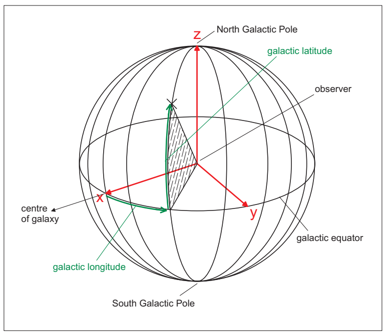
(d) horizontal



(e) local equatorial



(f) second equatorial



(g) galactic

Figure B.1: Common coordinate systems

B.1.6 The First Equatorial = Local Equatorial Coordinate System

The horizontal system depends on *place* (because the sky appears different from different points on Earth) and on *time* (because the Earth rotates, and each star appears to trace out a circle centred on the North Celestial Pole).

The first equatorial coordinate system is a first step towards a coordinate system that is fixed on the sky, independent of the observer's time and place. For this, the fundamental circle is no longer the horizon but the celestial equator. The celestial equator lies directly above (in the same plane as) the Earth's equator.

Any great circle between the NCP and the SCP is now a meridian. The one which also passes through the zenith (as seen by the observer) and the nadir is called *the* celestial meridian. It is identical to the principal vertical in the horizontal coordinate system (section B.1.5). The point where the celestial meridian crosses the *southern* half of the equator is the zero-point for the first equatorial system.

The direction of an object X can now be described by drawing a meridian through X. The declination of X is then the angular distance from the celestial equator to X, measured from -90° at the South Celestial Pole to $+90^\circ$ at the North Celestial Pole. The Hour Angle of X is the angular distance between the meridian of X and *the* celestial meridian, measured *west*wards and normally expressed in *hours* (0h – 24h), hence the name Hour Angle.

The first equatorial coordinate system is still tied to the observer's here-and-now, but over short periods of time the object's declination will not change. This fact is used for observing stars with telescopes mounted on so-called *equatorial mounts*.

Relationship to the Cartesian Coordinate System

For maximum similarity with the 'mathematical' spherical coordinate system, we define the relationship between the first equatorial coordinate system and the Cartesian coordinate system as shown in figure B.1: The x-axis points towards the southern intersection of celestial equator and *the* celestial meridian, the z-axis points towards the North Celestial Pole.

B.1.7 The (Second) Equatorial Coordinate System

The second equatorial coordinate system finally reaches the aim of being independent from the observer's position on Earth. To accomplish this, the second equatorial system still uses the celestial equator as its fundamental circle, just like the first equatorial coordinate system described in section B.1.6. However, the zero-point is now no longer defined by the observer's zenith, but by a fixed point on the celestial equator called the *vernal equinox*, or the *First Point of Aries*[1].

Like before, the declination of an object X is the angular distance from the celestial equator to X, measured on a meridian through X and ranging from -90° at the South Celestial Pole to $+90^\circ$ at the North Celestial Pole.

However, the second coordinate is now called *Right Ascension* (RA), and this is the angle along the celestial equator measured *east*wards from the vernal equinox to the meridian of X. Like HA, RA is measured in hours (0h – 24h). Note that it goes in the opposite direction, compared to the HA in the first equatorial coordinate system (section B.1.6).

Relationship to the Cartesian Coordinate System

For maximum similarity with the 'mathematical' spherical coordinate system, we define the relationship between the second equatorial coordinate system and the Cartesian coordinate system as shown in figure B.1: The x-axis points towards the vernal equinox, the z-axis points towards the North Celestial Pole.

B.1.8 The Galactic Coordinate System

The galactic coordinate system is no longer based on any of the properties of the Earth. Galactic coordinates describe the directions of objects relative to the galactic plane. The fundamental great circle is therefore the intersection of the galactic plane with the celestial sphere. This circle is called the *galactic equator*. The only fact where the Earth still comes in is in the definition of the Galactic Poles. The North Galactic Pole is defined as that pole in the same hemisphere as the North Celestial Pole.

The galactic latitude b^{II} can now be defined as the angular distance from the galactic equator to the object X, measured along a great circle through the Galactic Poles and X. Galactic latitude b^{II} ranges from -90° at the South Galactic Pole to $+90^\circ$ at the North Galactic Pole.

The zero-point for galactic longitude is the centre of the galaxy. The galactic longitude l^{II} of object X is the angular distance around the galactic equator from the centre of the galaxy to the great circle through X, measured *east*wards from $0^\circ - 360^\circ$.

[1]Despite the name, the vernal equinox is not fixed within the zodiac of Aries, but will itself move slowly through the sky [Walb].

Relationship to the Cartesian Coordinate System

For maximum similarity with the ‘mathematical’ spherical coordinate system, we define the relationship between the galactic coordinate system and the Cartesian coordinate system as shown in figure B.1: The x-axis points towards the centre of the galaxy, the z-axis points towards the North Galactic Pole.

B.1.9 The Geomagnetic Coordinate System

This coordinate system is used to describe positions relative to the magnetic field of the Earth. Lots of different definitions for geomagnetic coordinates exist, and generally they depend on time (epoch) and sometimes on complex magnetic models. Consequently, the use of geomagnetic coordinates can lead to much confusion, and as these coordinate systems are of no further consequence to this thesis, we will not discuss them further here. [GPP92] and [TUP$^+$87] are good starting points.

B.2 Converting Between Coordinate Systems

In order to correctly convert between the different coordinate systems mentioned in section B.1, it is obviously not sufficient to just change the scales of the different coordinates. As different systems have different origins, it is necessary to express the base vectors of one system in terms of the base vectors of another system. In addition to that, a position vector is needed to specify how the two origins are related to each other.

B.2.1 Relation Between Horizontal and Geographic Coordinates

We illustrate this for the most often required example of converting between geographic longitude/latitude (geographic coordinate system, see section B.1.4) and the horizontal coordinate system of an observer on the surface of the Earth.

We use the underlying Cartesian coordinate system of both systems as described in the respective sections above. Now the problem shows itself as depicted in figure B.2.

We are looking for the base vectors $\overrightarrow{e_x}$, $\overrightarrow{e_y}$ and $\overrightarrow{e_z}$, expressed as multiples of the base vectors $\overrightarrow{x}$, $\overrightarrow{y}$ and $\overrightarrow{z}$ of the basic geographic coordinate system. The observer is located at a position described by $\overrightarrow{X}$. $\overrightarrow{e_x}$, $\overrightarrow{e_y}$ and $\overrightarrow{e_z}$ can then be determined by using the following relationships:

$\overrightarrow{e_z}$	Same direction as $\overrightarrow{X}$, unit length.

$\overrightarrow{e_x}$	Lies in the same plane A as $\overrightarrow{X}$ and $\overrightarrow{z}$. If A intersects z-axis in positive half-axis, then $\overrightarrow{e_x} = \frac{\overrightarrow{PX}}{\left\|\overrightarrow{PX}\right\|}$, otherwise $\overrightarrow{e_x} = -\frac{\overrightarrow{PX}}{\left\|\overrightarrow{PX}\right\|}$.
$\overrightarrow{e_x}$	Forms a right-handed Cartesian coordinate system together with $\overrightarrow{e_z}$ and $\overrightarrow{e_x}$.

From these relationships, a function **azeltriad()** was developed that returns the base vectors $\overrightarrow{e_x}$, $\overrightarrow{e_y}$ and $\overrightarrow{e_z}$ as multiples of $\overrightarrow{x}$, $\overrightarrow{y}$ and $\overrightarrow{z}$, given the position of the observer $\overrightarrow{X}$. This function is part of the RIOSIM package as described in chapter 6.

B.2.2 First to Second Equatorial Coordinate System

As sections B.1.6 and B.1.7 explain, the first equatorial coordinate system still depends on the rotation of the Earth, whereas the second equatorial coordinate system does not. Declination δ is the same in both systems, and hour angle h and right ascension α are related as follows:

$$h = \theta - \alpha \qquad \text{(B.1)}$$

where θ is the *local apparent sidereal time* and α is the (calculated) apparent right ascension of the object in question at the given time. The local apparent sidereal time is determined as described in section C.5.

B.2.3 First Equatorial to Horizontal Coordinate System

The final conversion from first equatorial to horizontal coordinates is straightforward, as the relation between the two coordinates remains fixed for any given location of the observer on Earth. Horizontal coordinates follow from first equatorial coordinates by means of rotation. The same rotation applied in the opposite direction converts from first equatorial to horizontal coordinates.

B.2.4 Catalogued Star Coordinates to Current, Observable Coordinates

The position of stars (or more generally astronomical objects) is normally catalogued using a certain coordinate system. The most widely used catalogue is the FK5 catalogue [FSL+88], and together with many others, it gives the positions of stars in mean Right Ascension α and mean declination δ, thus coordinates in the so called (second) equatorial coordinate system, see section B.1.7.

Epochs

As described in section B.1.7, this coordinate system uses the plane of the equator as its fundamental plane, and the *vernal equinox* as the fixed point in space specifying the origin of α. As has been briefly mentioned already, both these positions are not fixed in space. Therefore, FK5 coordinates have to be specified together with the date the coordinate system was based on, for example ‘equinox J2000’ where J2000 specifies the *Julian epoch* 2000, with ‘epoch’ being another word for ‘instant in time.’ Specifically, epoch J2000 is defined to be 2000 January 1.5 in the TT timescale (see appendix C for more information about timescales).

Proper Motion

On top of that, most astronomical objects are not fixed in space with respect to the ‘fixed background.’ Instead, ’proper motion’ can be observed, and is normally catalogued as well. Proper motion $\mu_\alpha = \dot{\alpha}$ is the observed change of α, usually specified in radians per century. Similarly, $\mu_\delta = \dot{\delta}$ is the change in δ.

Now, one obviously not only has to specify position, coordinate system and proper motion, but also the date when the astronomical object was observed at the specified position, since it will have moved on since then.

Therefore, a full star catalogue entry contains (α, δ) in (second) equatorial coordinates of a given year (e.g. ‘$\alpha = 22^h34^m10^s.761$, $\delta = +276^\circ.281$ equinox J2000’), the proper motion terms (μ_α, μ_δ) and a date when the object was observed at the given position, e.g. ‘epoch J2000.’

Luckily, most catalogues reduce both dates to the same date. All values given in the FK5 catalogue, for example, are valid for epoch J2000 in the coordinate system based on equinox J2000.

Precession and Nutation

Unfortunately, as mentioned under ‘Epochs’ above, the position of the points that the equatorial coordinate system is based on varies with time, i.e. the coordinate system is constantly moving relative to the distant sky background. By far the largest influence is caused by the so-called ‘precession of the equinoxes’ [Walb]. The positions of the equinoxes change with time due to the precession of the Earth’s axis. Confusingly, two terms have been established for this: *luni-solar precession* describing the low frequency (one period in 26,000 years) main effect, and *nutation* describing the smaller high frequency terms. Note that precession and nutation are simply different frequency components of the same physical effect.

In addition, the planets in our solar system influence the Earth-Moon system, causing the ecliptic to rotate slowly. This is called *planetary precession*.

Planetary precession and *luni-solar precession* together are called *general precession*, and this is what is still included in the various 'mean' coordinates, whereas the influence of *nutation* is not. See below.

Mean Place

As mentioned above, catalogues usually contain the 'mean place' of the object in question, the term 'mean place' meaning that the low frequency component of precession ('general precession', see section B.2.4) has been included in these coordinates, while the higher frequency components have been removed. So the object in question might never have been observed at the specified place!

Aberration

The finite speed of light combined with the motion of the observer around the sun during the year causes apparent displacements of the positions of the stars. This effect is called *annual aberration*. In addition, there is a small contribution called *diurnal aberration* due to the rotation of the Earth. Also, the so-called *E-terms* describe the influence of the Earth's orbit being elliptical instead of circular.

Conversion FK5 to Horizontal

Let's assume we know the coordinates of an object in terms of $(\alpha, \delta, \mu_\alpha, \mu_\delta)$, with equinox and position both reduced to the same epoch, e.g. J2000. In order to find the coordinates where the object can be observed at a given time in the horizontal coordinate system of an observer on Earth, we need to follow the following steps:

1. apply proper motion from catalogued data — this will leave us with new coordinates (α^*, δ^*) specifying the position of the star for the time in question, but still in the coordinate system of, say, J2000.

2. apply rotation caused by precession/nutation of the Earth's axis since the catalogued epoch, and allow for 'annual aberration' due to the movement of the Earth relative to the object in question

3. convert the resulting coordinates to the first equatorial coordinate system by taking into account the rotation of the Earth

4. adjust for 'diurnal aberration'

5. convert the resulting coordinates to the horizontal coordinate system which is fixed in relation to the first equatorial coordinate system, given a fixed location of the observer on the Earth.

6. Observations at high frequencies would now have to take into account refraction effects as well. However, waves of the frequency in question in our case (around 38MHz) pass through the Earth's atmosphere without being conceivably refracted.

B.2.5 Horizontal Coordinates to Galactic Coordinates

This conversion is needed for example if we have a digital representation of a background sky temperature map (see section 6.4) and want to calculate how the sky temperature is seen by a particular observer on Earth. The 'native' coordinate system of the observer is the horizontal coordinate system, and we have to convert these coordinates all the way to galactic coordinates which are normally used for sky maps.

Referring to the previous section, the following steps need to be followed:

1. convert horizontal coordinates to first equatorial coordinates, given the location of the observer on Earth

2. adjust for 'diurnal aberration'

3. convert to second equatorial coordinate system

4. apply rotation caused by precession/nutation and allow for 'annual aberration'

5. convert the resulting FK5 J2000.0 coordinates to galactic coordinates

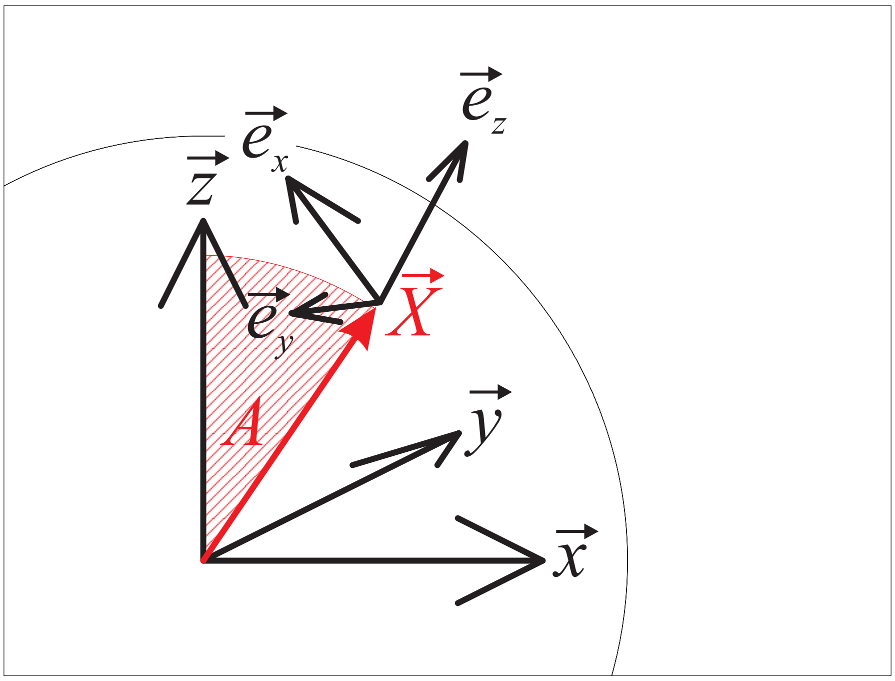


Figure B.2: Conversion from geographic to horizontal coordinates

Appendix C

Timescales

Coordinate systems are not the only issue that can cause a lot of confusion when it comes to running exact simulations. A variety of different scales for measuring time exist, and their relationships to each other are described in this appendix. Timescales are also important for the design of the ARCOM operating software (see chapter 8).

Tables C.1 and C.2 list the most widely used timescales together with their basic properties. The timescales of relevance for the current work will be discussed in more detail in the following sections. Large parts of these sections are compiled from information in [Walb]. For a real-time display of several timescales see [RTC].

C.1 International Atomic Time TAI

This timescale is established by a number of atomic clocks around the world. The basic unit is the SI second [SI], which itself is defined in terms of a defined number of wavelengths of the radiation produced by a certain transition in the Caesium 133 atom.

TAI is a continuous timescale, meaning that there are no leap seconds.

C.2 Coordinated Universal Time UTC

UTC shares the same basic unit with TAI. UTC is the basis of civil timekeeping, most time zones differ from UTC by an integer number of hours. As opposed to TAI, UTC keeps in sync with the sun. It does so, even though the Earth's rotation is slightly variable, by occasionally introducing a leap second. The International Earth Rotation Service [IER] determines whether or not a leap second will be inserted. It can only be inserted at the end of the months of December and June.

UTC cannot be expressed in Julian Date, as ambiguities would arise during leap seconds.

C.3 Universal Time UT=UT1

UT1 is a continuous timescale that keeps in sync with the mean sun. As the Earth's movement is slightly variable, the basic unit of the UT timescale is variable as well. UT is obtained by looking up the value of $\Delta = \mathrm{UT1} - \mathrm{UTC}$ in tables published by the International Earth Rotation Service [IER] for the date concerned. This quantity is kept in the range $\pm 0^s.9$ by means of UTC leap seconds, as described in section C.2 above. It is possible to predict Δ with an accuracy sufficient for pointing telescopes etc.

C.4 Greenwich Mean Sidereal Time GMST

Sidereal Time is the 'time of day' relative to the stars rather than to the Sun. After one sidereal day the stars come back to the same place in the sky, apart from sub-arcsecond precession effects. Because the Earth rotates faster relative to the stars than to the Sun by one day per year, the sidereal second is shorter than the solar second; the ratio is about 0.9973.

The Greenwich Mean Sidereal Time GMST is linked to UT1 by a numerical formula. There are, therefore, no leap seconds in GMST, but the second changes in length along with the UT1 second, and also varies over long periods of time because of slow changes in the Earth's orbit. This makes the timescale unsuitable for anything except predicting the apparent directions of celestial sources.

C.5 Local Apparent Sidereal Time LAST

Local Apparent Sidereal Time (LAST) is the apparent right ascension of the local meridian, from which the hour angle of any star can be determined knowing its Right Ascension α. It can be obtained from GMST by adding the East longitude (corrected for polar motion in precise work) and the equation of the equinoxes. The latter is an aspect of the nutation effect described in section B.2.4.

C.6 Network Time Protocol (NTP) Timescale

The Network Time Protocol is a standard internet protocol to synchronise clocks of different, spatially distributed, hosts [NTPb]. The NTP timescale is based on UTC and therefore shares the SI second as its basic unit. An NTP timestamp is a 64bit integer value with the high order 32 bits representing the number of seconds since the start of the current era and the lower 32 bits representing fractions of seconds. The current NTP era began with a counter value of 0 at 0h on the 1st of January 1900. However, the NTP timescale does not know anything about leap seconds. This means that, whenever a UTC leap second is inserted, a new NTP timescale is effectively established. During a potential leap second, the current NTP time remains constant.

C.7 GPS Timescale

The GPS timescale itself (also known as GPS Composite Clock or GPS time) is broadcast by the Global Positioning System (GPS) in the GPS navigation message and can be received with a simple GPS receiver [HFM02]. This is known as one-way GPS time transfer. A GPS receiver locks onto one satellite and receives a broadcast message. The broadcast message contains the satellite-specific offset between the received GPS clock information and the basic GPS clock as well as the current offset between GPS time and UTC(USNO). UTC(USNO) stands for the current UTC as determined by the United States Naval Observatory (USNO). With this information, the GPS receiver can derive UTC(USNO).

GPS time is measured in full weeks and seconds since start of the GPS timescale. To convert between GPS and UTC timescales, it is good to know that a fixed relationship exists between TAI and GPS time, see for example [Obsb]. GPS time 0 was at 00:00:00h on January 6th, 1980. At that time, the difference between GPS timescale and UTC timescale was 0 seconds. From [Obsa] it can be seen that the TAI timescale was offset by 19 seconds from UTC at that time ($\mathrm{TAI}-\mathrm{UTC}=19.0$). A table with the complete history of leap seconds is published by the United States Naval Observatory, see [Obsa]. Based on this table, RIOSIM (see chapter 6) implements a pair of functions — **utc2gps()** and **gps2utc()** — to convert between the two timescales. Note that these functions will need to be updated as new leap seconds are introduced. It should be noted that such conversions are usually only necessary after the dataset in question has been recorded, so that up-to-date knowledge of leap seconds is not needed for data recording, especially as the GPS transmitters continually transmit the current $\mathrm{TAI}-\mathrm{UTC}$ value.

abbrev.	name	basic unit	main properties
TAI	International Atomic Time	SI second	Continuous (no leap seconds). Therefore (increasing) time lag between TAI and UTC.
UTC	Coordinated Universal Time	SI second	Basis for civil timekeeping. Leap second to keep in sync with the sun. If those leap seconds are inserted or not is determined by the International Earth Rotation Service [IER]. They may only be introduced at the end of the months of December and June.
UT=UT1	Universal Time	variable 'second'	Mean solar time. Continuous. Needed for computing sidereal time. Derive UT1 from UTC by looking up (UT1-UTC) in tables published by the International Earth Rotation Service [IER].

Table C.1: Timescales (1)

abbrev.	name	basic unit	main properties
GMST	Greenwich Mean Sidereal Time	≈ 0.9973 variable 'seconds' = solar second	Mean time relative to stars. Continuous. Linked to UT1 by a numerical formula.
LAST	Local Apparent Sidereal Time	as above	Apparent right ascension of the local meridian. Obtained by adding East longitude of location and equation of equinoxes to GMST.
TT	Terrestrial Time	SI second	Currently: TT=TAI+32.184s
TDB	Barycentric Dynamical Time	variable 'second'	Differs from TT by an amount which cycles back and forth by between 1 and 2 milliseconds due to relativistic effects.
NTP	Network Time Protocol Timescale	SI second	Standard protocol to distribute time information between computers. See [NTPa]. NTP timescale is constantly re-synchronised with UTC. Therefore, a 'new' NTP timescale is established after each UTC leap second.
GPS	Global Positioning System Timescale	SI second	GPS time is the time that is distributed through the Global Positioning System (GPS). It uses the same basic unit as UTC, but does not contain leap seconds. Instead, the current offset from UTC is transmitted as part of the GPS navigation message, allowing the derivation of UTC from the GPS time.

Table C.2: Timescales (2)

Appendix D

ARIES System Diagrams

The following pages contain diagrams outlining various aspects of the Advanced Rio-Imaging Experiment in Scandinavia (ARIES) system. The following diagrams are provided:

- ARIES block diagram (figure D.1, courtesy of Peter Chapman).
- ARIES receiver block diagram and photograph (figures D.2 and D.3, courtesy of Peter Chapman).
- ARIES FPGA data flow (figure D.4, courtesy of Keith Barratt).
- ARIES physical layout, numbering scheme and coordinate system orientation (figure D.5).

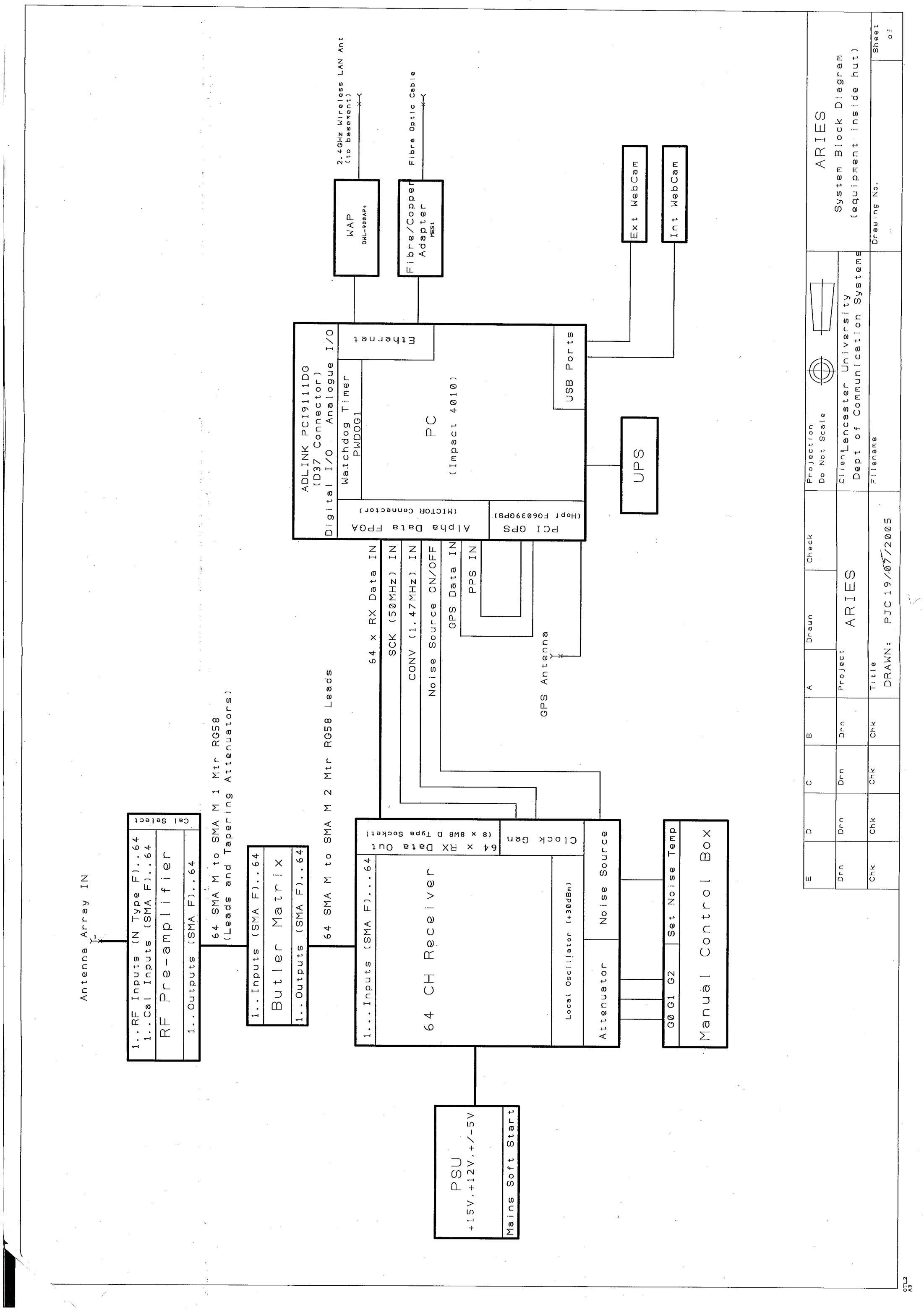


Figure D.1: ARIES block diagram. Courtesy of Peter Chapman.

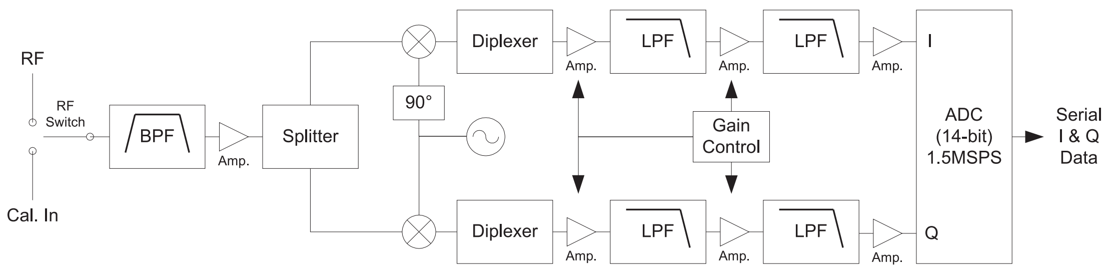


Figure D.2: ARIES receiver block diagram showing a single channel. A 64 channel, 14bit, 1.5MSPS simultaneously sampled receiver has been developed specifically for the ARIES project. The complex (in-phase & quadrature) digitised signals from each of the 64 receiver channels are output as multiplexed serial data; this helps reduce the wiring between the receiver rack and the PC-mounted FPGA. Courtesy of Peter Chapman.

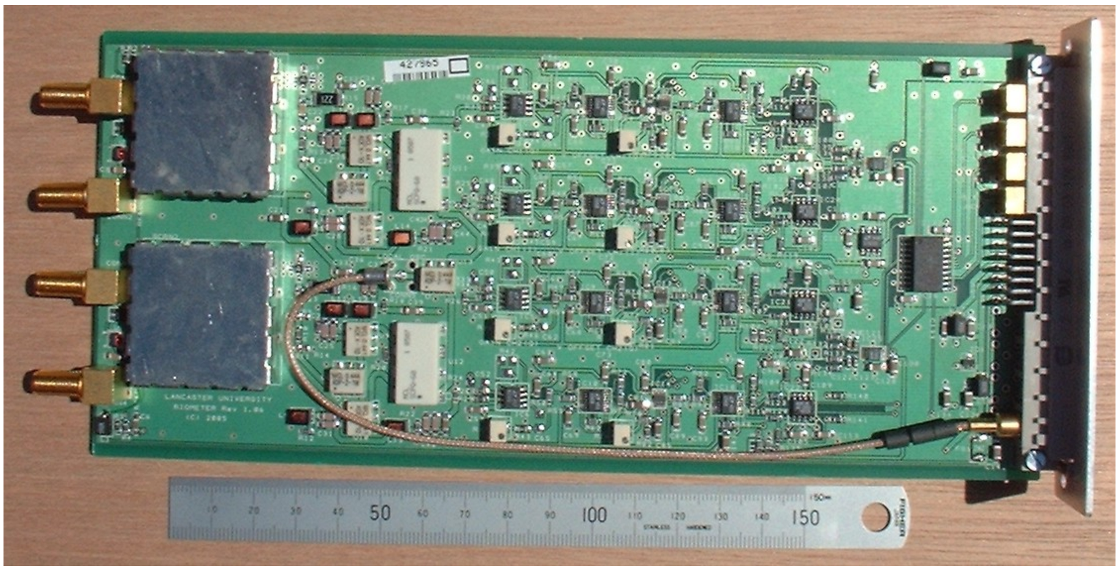


Figure D.3: ARIES receiver PCB. Two channels are co-located on a standard 220mm × 100mm Eurocard. 32 such receiver boards are required for the complete system and are housed, along with oscillators, calibration noise source and other peripheral electronics, in an industry standard 12U 19” sub-rack. Courtesy of Peter Chapman.

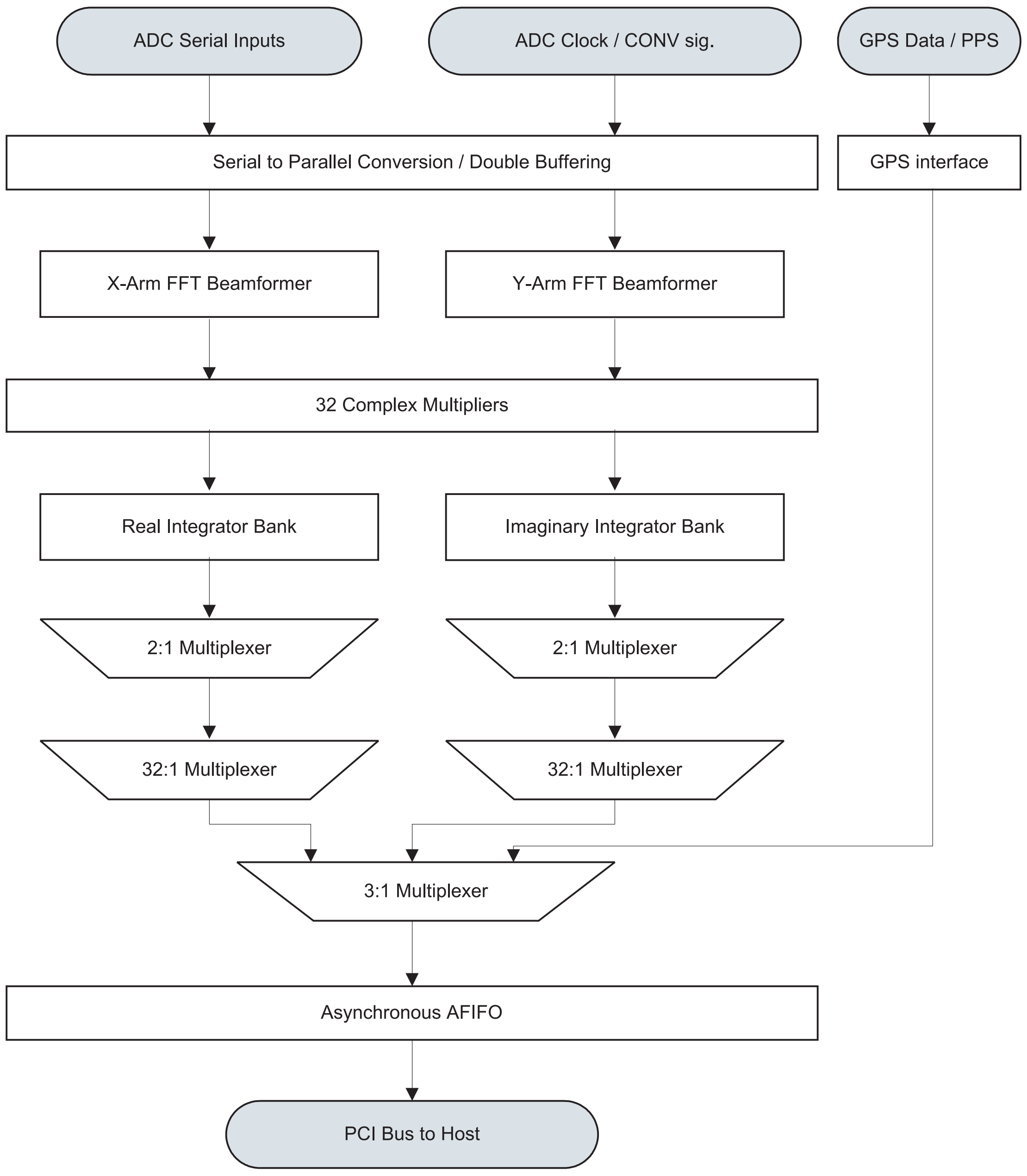


Figure D.4: ARIES FPGA data flow. Courtesy of Keith Barratt, taken from [Bar07].

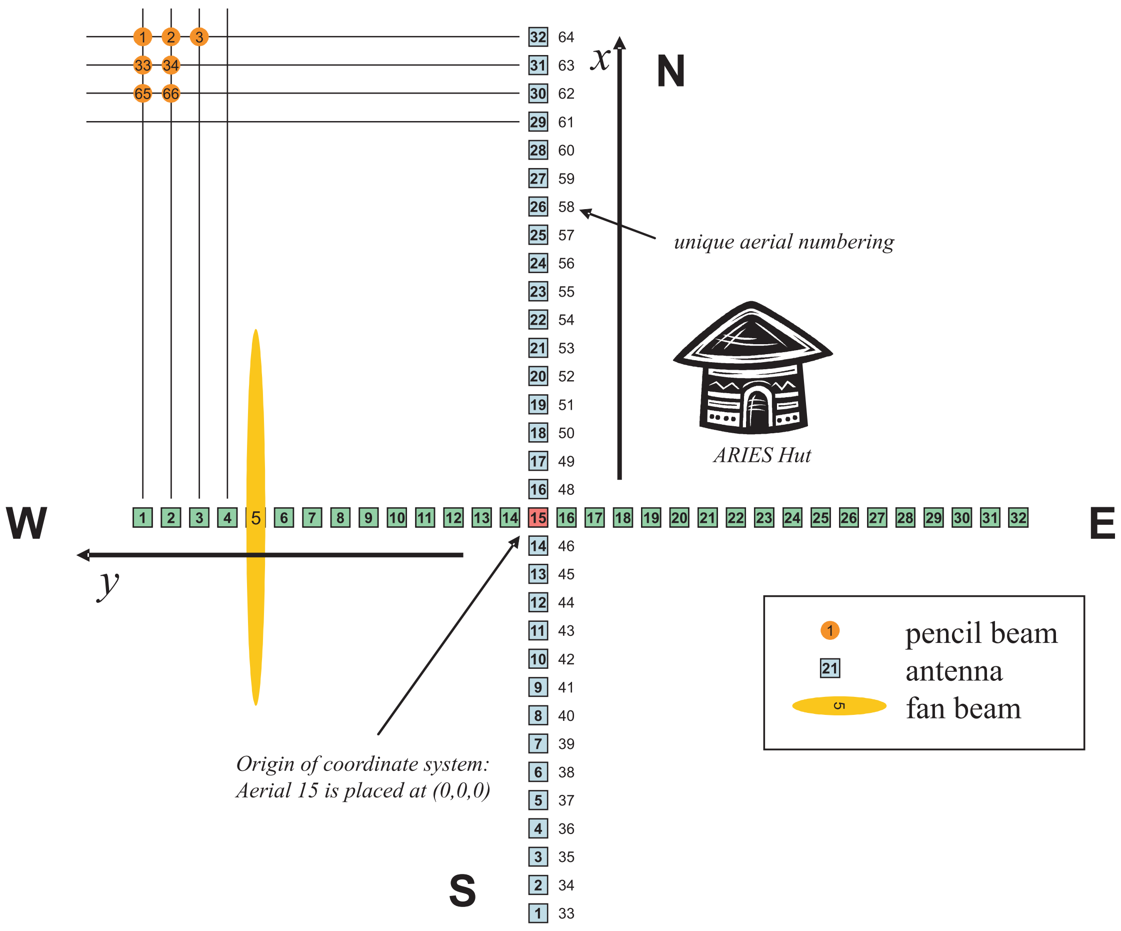


Figure D.5: ARIES physical layout, numbering scheme and coordinate system orientation as used throughout this thesis. Fan beams are numbered in the same way as aerials. The fan beam that points West is therefore number 1, the fan beam pointing North is number 64.

Appendix E

File Excerpts for ARIES Model

E.1 `run` Shell Script for ARIES Model

This is the simple shell script as mentioned in section 4.2.

```
#!/bin/sh

# build model files
cd cpp
make
make -f beamform.mak
make -f xcorr.mak
cd ..

# delete all previous output data
rm data/*

# run model
./model $*

# run butler matrices
./beamform 1
./beamform 33
```

```
# run cross correlator
./xcorr
```

E.2 Sample Source Definition File

This is an example of the format of the `*.source` files as used by the ARIES model. This particular `.source` file is actually the one that was used to create figure 4.6. The three sources defined herein were used to verify the correct orientation of the model's coordinate system.

```
# source definition file for ARIES model_v2
#
# lines starting with # are comments.
#
# each line has the following format:
# x y z a etc.
#
# where (x,y,z) = direction vector of source
# (a) = amplitude
# (etc.) = additional information (ignored)

1   0   1   1  to indicate x-axis
0   1 2.5   1  to indicate y-axis
0   0   1   1  to indicate z-axis (zenith)
```

E.3 Example Output of a Simulation Run

The following is the screen output of a simulation run. This particular simulation is the one that was run for section 4.3.5.

```
[grill@egb024000005 model_v2]$ ./run -f sources/ghost_sources.so
       urce -s 400000 -n 2
/home/aurora/grill/prg/model_v2/cpp
make: 'model' is up to date.
```

```
make: `beamform' is up to date.

make: `xcorr' is up to date.

Sourcefilename: sources/ghost_sources.source

NrOfSamples...: 400000

NoiseSrcType..: 2

FilterCoeff...: coeff/standard.coeff


Model Master Control proudly presents the model parameters:

  InternalSamplingFrequency = 1.2224e+09 Hz

  Maximum required delay buffer = 570 Samples

  We're aiming to calculate 400000 Samples

  The noise sources are of type 2.

  The IIR filter coefficients: coeff/standard.coeff.


  * creating aerials... Mills Cross antennas created.

  * reading sources from sources/ghost_sources.source...

    more than 0 sources so far...

    Amazing! We have 2 sources!


Running model...

  progress: Nearly finished, closing files... OK


Forming beams from aerials (1:32)...

  * opening files.

  * adding and delaying...

    nr. of lines is 400000

    progress: That's it! Beams are stored in 'data/beamXX'.


Forming beams from aerials (33:64)...

  * opening files.

  * adding and delaying...

    nr. of lines is 400000
```

```
        progress: That’s it! Beams are stored in ’data/beamXX’.


    Cross correlating 400000 Samples.

      progress: finished. Result stored in ’result’.


    [grill@egb024000005 model_v2]$
```

Appendix F

Specialbeams.txt

This is the parameter file that was used for the October 2002 experiment preparations. It is interpreted by the RIOSIM function **getspecialbeampattern()** as described in section 6.6.3.

```
#
# specialbeams.txt
#
#
# This file defines special beams that can be generated
# using the ARIES Mills Cross antenna array.
#
# This file is interpreted by GETSPECIALBEAMPATTERN.M.
#
# Lines starting with '#' are comments.
#
# Each line describes one beam and has the following fields:
#
# <nr> <type> <additional_parameters>
#
# <nr> is the unique number of the beam to be defined
# <type> is either A[dditive_array] or M[ultiplicative_array]
#
# For type A the <additional_parameters> are:
# <nr> A <sizex> <sizey> <dphasex> <dphasey>
# a linear phased array will be created with these parameters
#
# For type M the <additional_parameters> are:
# <nr> M <beam1> <beam2>
# a multiplicative array will be created from the two specified beams,
# these beams themselves need to be defined in the file, too.
#
#
# NOTES:
# o All beams defined in this file will be rotated to correct for
#   the slope of the ARIES site.
#
# o The individual radiation pattern of each single element is
#   a CXDipoleNielsenPattern.
#
# o Physically, the ARIES antenna array consists of two arms of
#   32 antennas each. Therefore, all type A beams should be arrays
#   of either size 1xX or Yx1.
#
# o refer to 20020906_MillsCrossCoordinateSystem.ppt for a description
#   of the coordinate system used.
#
# o It may prove useful to use the following number ranges
#   in order to avoid ambiguities:
# 2000:2999 for fan beams (formed by linear additive array)
# 3000:3999 for pencil beams (formed by multiplying two fan beams)
#
#
# SEE ALSO:
# GETSPECIALBEAMPATTERN, ARIES_contour*
#
# fan beams (2000:2999) ########################################
```

```
2000 A 32 1 0 0 # zenithal fan beam NS

2001 A 1 32 0 0 # zenithal fan beam EW

2002 A 32 1 0.490874 0.0 # this should give us trofan 46

2003 A 1 32 0.0 0.490874 # this should give us trofan 19

# beams > 2500 are for the 16+16 array

2500 A 16 1 0.0 0.0 # zenithal fan 1

2501 A 1 16 0.0 0.0 # zenithal fan 2

2502 A 16 1 0.490874 0.0 # a fan beam for the 16+16 array

2503 A 1 16 0.0 0.490874 # ditto

# pencil beams (3000:3999) #####################################

3000 M 2000 2001 # zenithal pencil beam

3001 M 2002 2003 # this should give us tro 595

# beams > 3500 are for the 16+16 array

3501 M 2502 2503 # a 16+16 beam at the same location as beam tro_595

3502 M 2500 2501 # a zenithal 16+16 pencil beam
```

Appendix G

ARIES ‘Earlobe’ Plots

These are the complex phase and amplitude plots for all ARIES beams, including 1-month average (black) ±15 days of 2007-03-23, single-day data (yellow) for 2007-03-23 and simulated data (red) for each beam. Each panel is a plot of power data over one sidereal day on the complex plane. X-axis is the real axis from left (negative) to right (positive), y-axis is the imaginary axis from bottom (negative) to top (positive), origin is at the centre. Axis scaling in arbitrary linear power units. Further details see chapter 10.

Figure G.1 is an overview of the central 676 beams, figures G.2 to G.7 are larger versions for all beams.

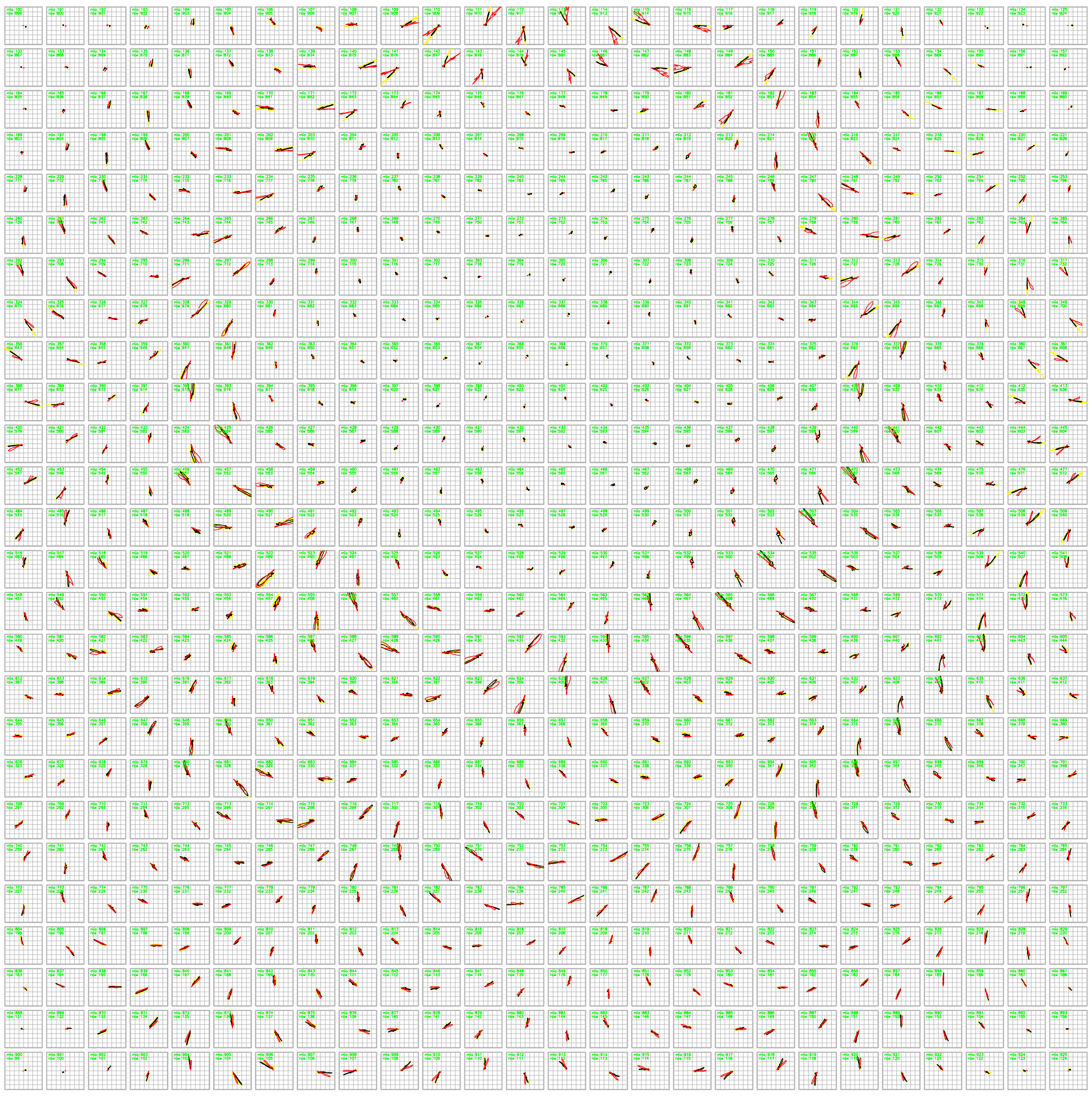


Figure G.1: ARIES 'earlobe' plots: all central 676 beams

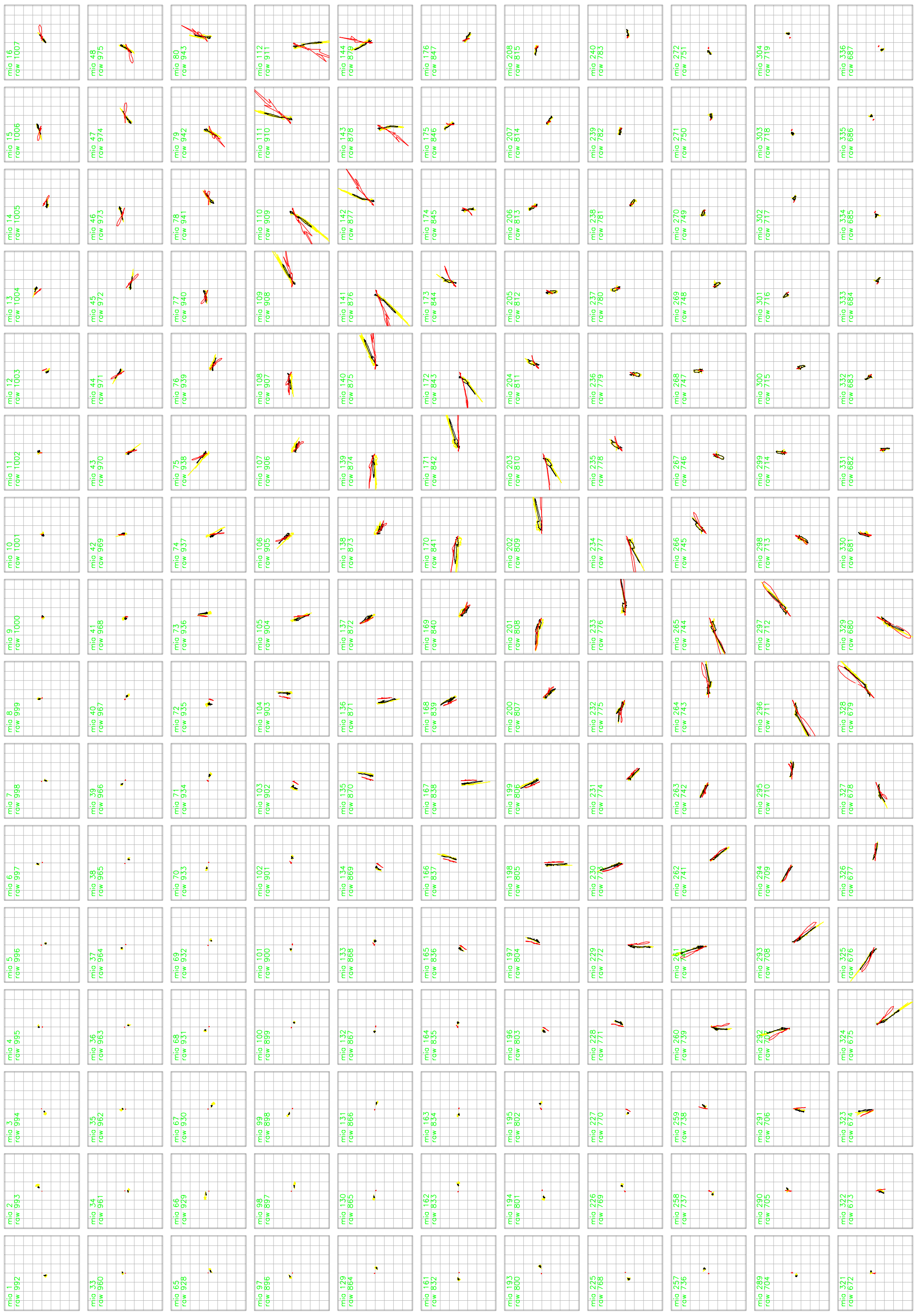


Figure G.2: Zoomed-in version of figure G.1, part 1

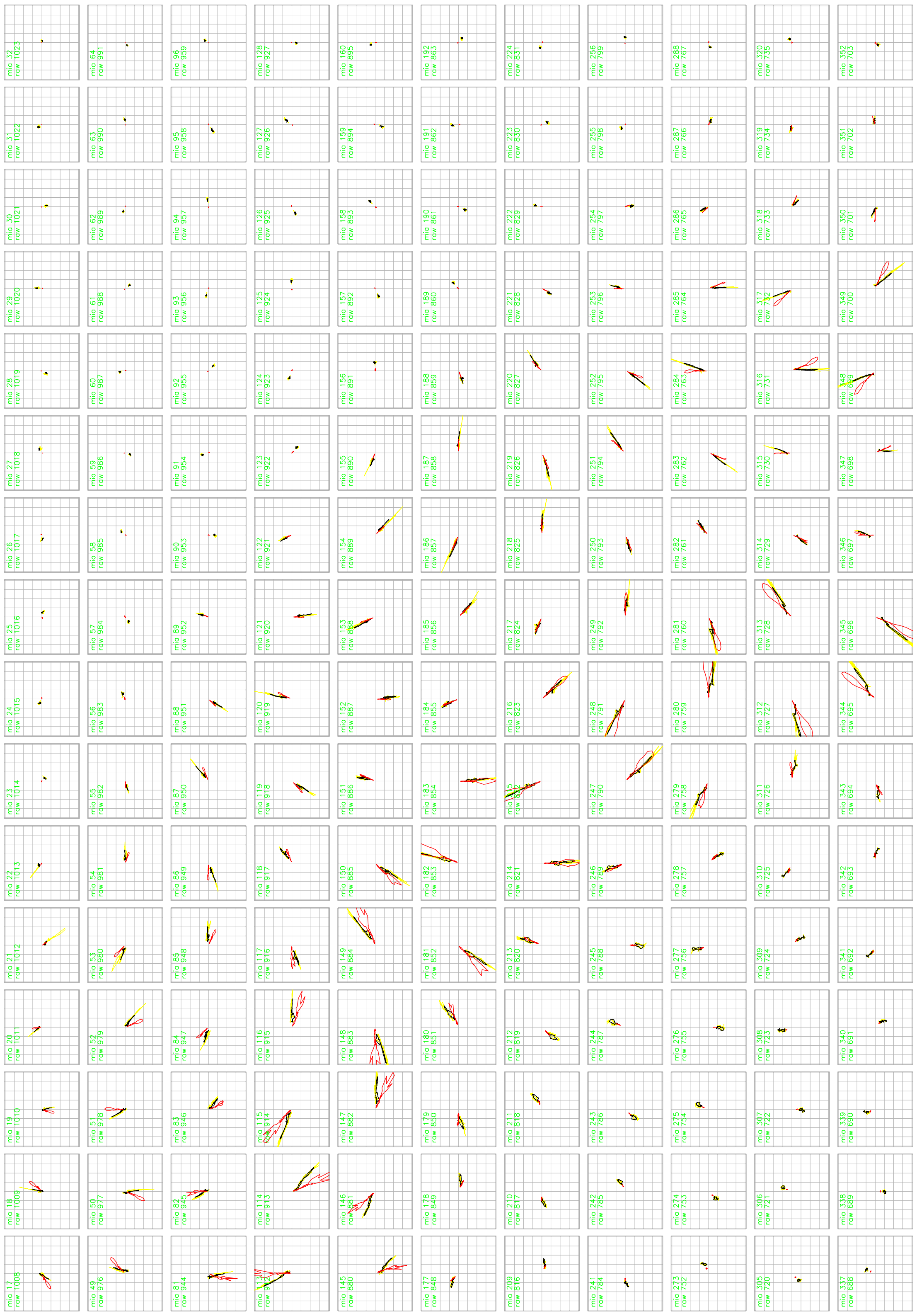


Figure G.3: Zoomed-in version of figure G.1, part 2

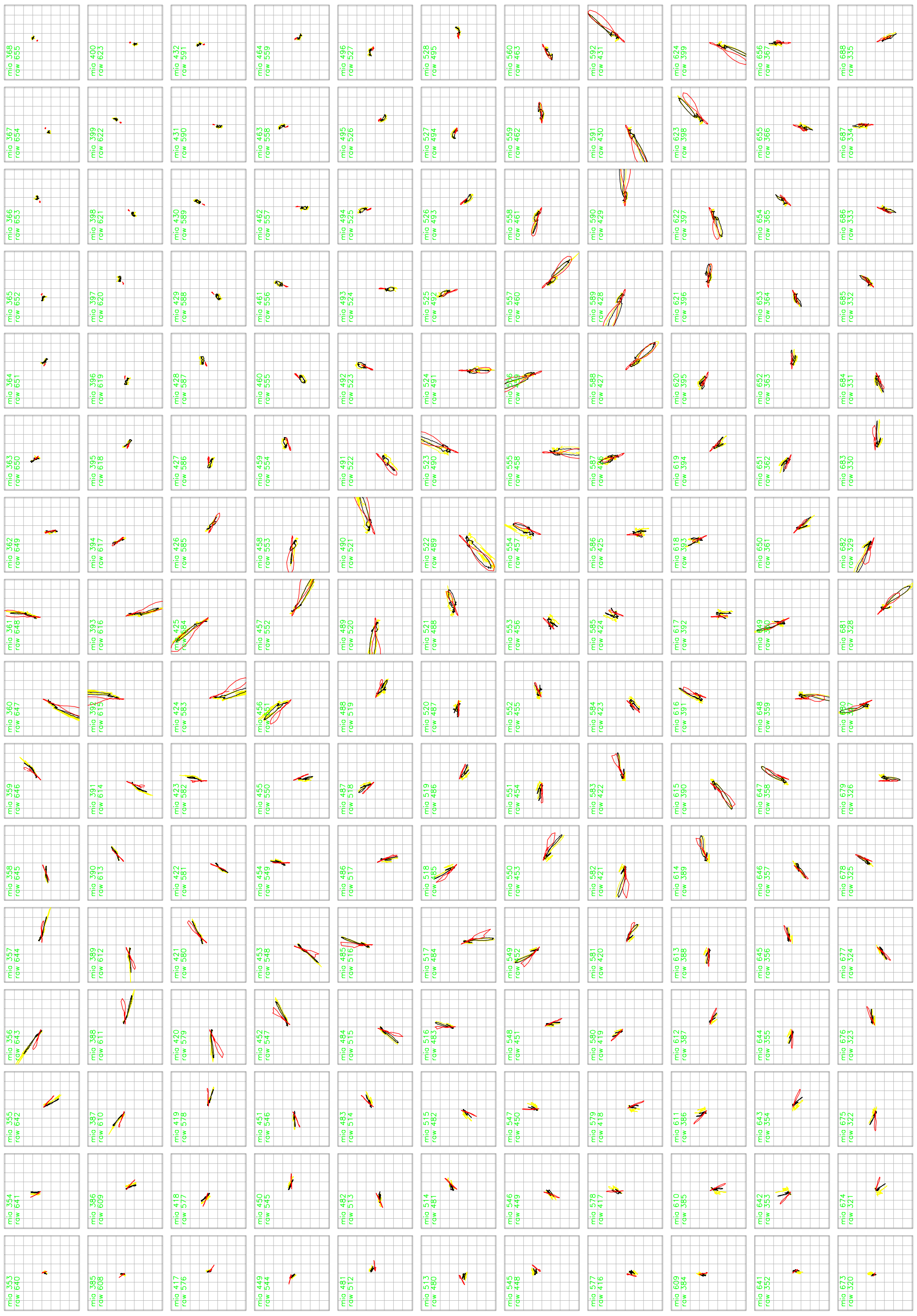


Figure G.4: Zoomed-in version of figure G.1, part 3

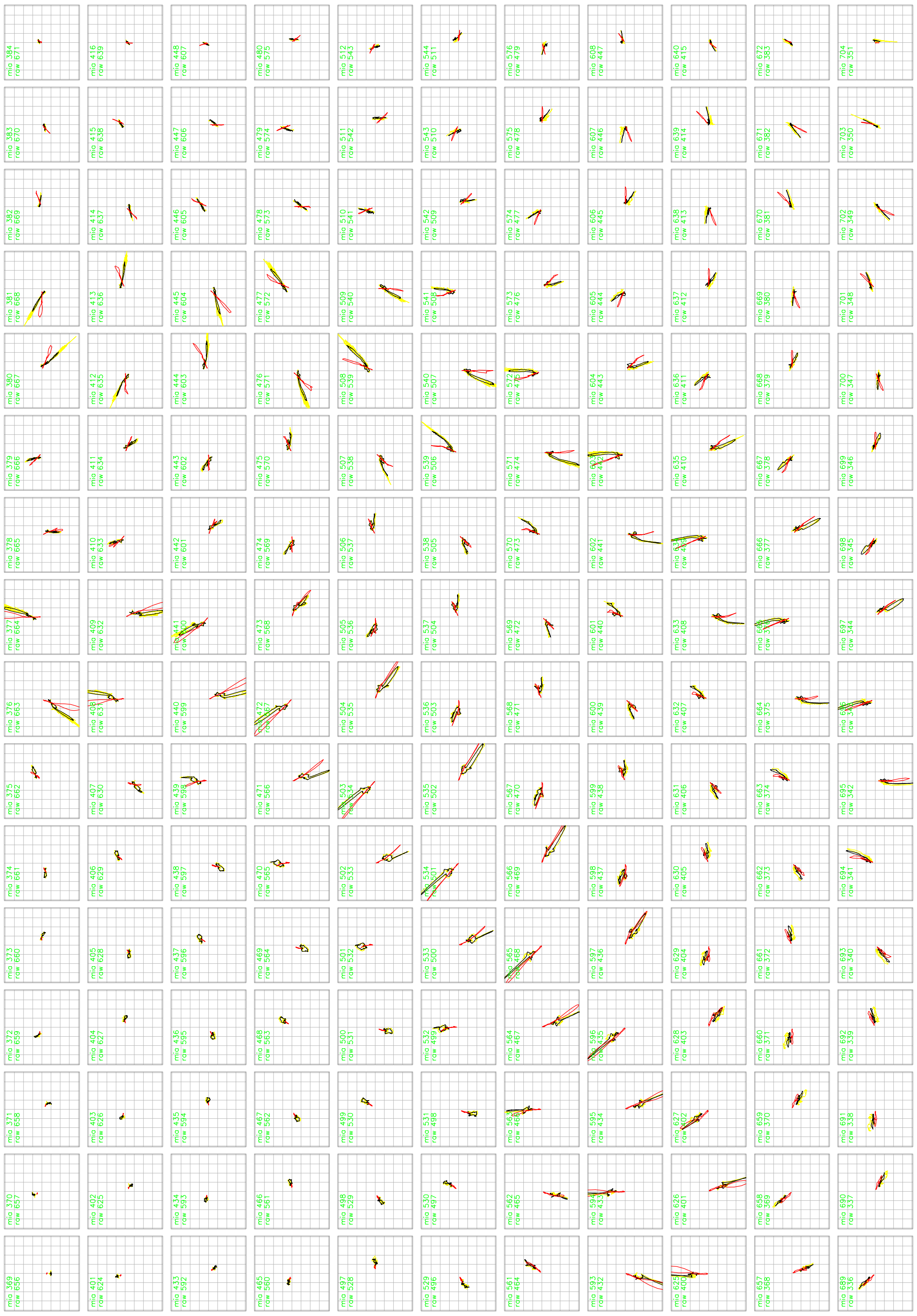


Figure G.5: Zoomed-in version of figure G.1, part 4

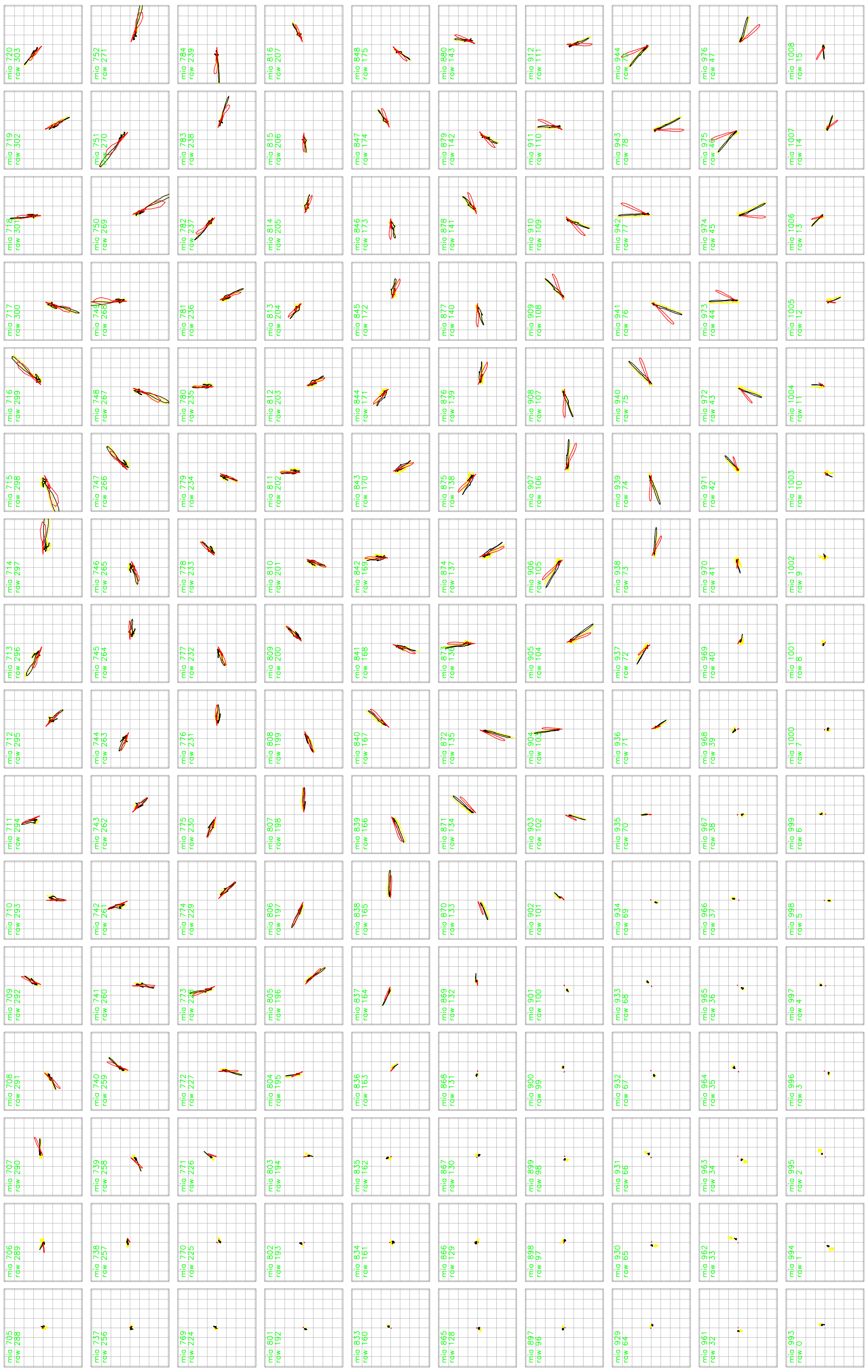


Figure G.6: Zoomed-in version of figure G.1, part 5

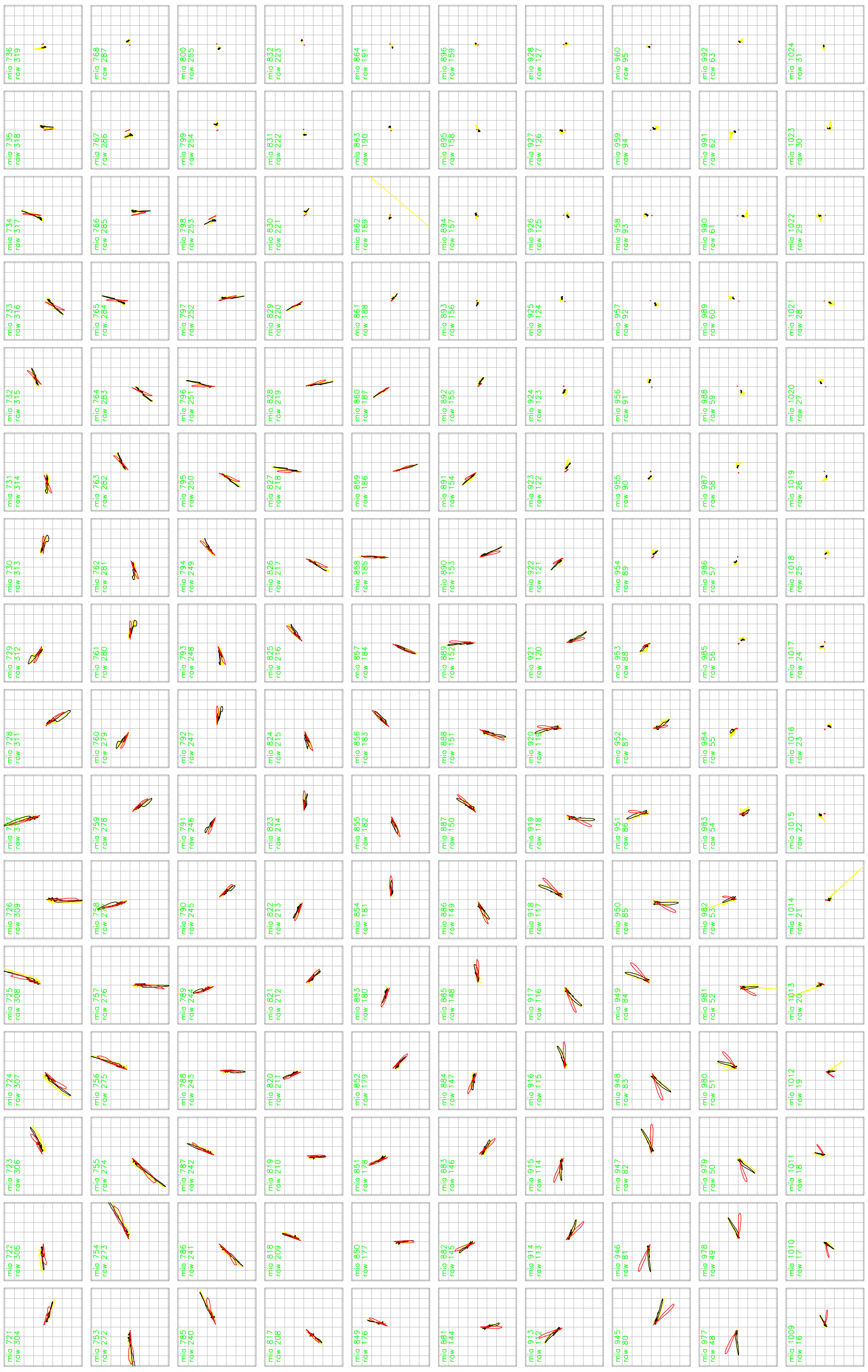


Figure G.7: Zoomed-in version of figure G.1, part 6

Appendix H

MIA's Asynchronous Processing System

The following paragraphs give a brief outline of the scheduling mechanism that is used by the WWW-based MIA data access facilities as originally developed by S. Marple.

It is important to note that when the MIA scheduling system was first conceived, it was not designed to be a scheduling system at all. Instead, as far as retrieving data was concerned, the only additional task it handled was logging all data requests. This information was needed by the funding body, and maintaining the data request logs manually was tedious and error-prone. Based on these requirements, Marple designed a set of database tables that would be used to log the details of each request made by users through the web interface. At this stage, the requests still had to be processed manually.

It was only afterwards, that the need for automatic processing arose due to a constantly increasing user base and therefore a constantly increasing number of data requests. Since all the information was stored inside the logging database anyway, a straightforward solution was implemented in the form of a backend process that continuously watched the database for new log entries and launched an automated processing stage whenever a new log entry was detected.

Therefore, a scheduling system had been created from the existing roots of the logging system, meaning that most of the existing logging code could be reused. Unfortunately, this system turns out to have some drawbacks, all based around the fact that it evolved from a simple logging system with scheduling capabilities added as an afterthought.

H.1 Request-specific Structure

The originally devised logging system included one separate table for each type of data request. This table would have columns according to the specific type of request. For example, the table for riometer image plot requests would have a column that specifies the requested mapping height, whereas a mapping height is not required for magnetometer data requests and is therefore not included in the magnetometer data request table.

However, this means that whenever a new type of data request needs to be added to the system, multiple changes need to be made by the administrator:

- A new table for the new kind of request needs to be added to the database.
- The processing backend needs to be informed about this new table, it needs to watch this table for changes and launch a processing tool whenever a new request is detected.
- The processing tool needs to be able to handle data in exactly the format supplied by the database table.

H.2 Centralised Organisation

As described, addition of a single type of request requires changes to nearly every part of the system. This may not be a problem in small systems. In fact, a single administrator can normally take care of this task, ensuring that the system remains dependable and free of errors.

However, as the system grows and several programmers need to add different kinds of processing engines, the advantages of such a centralised, integrated approach quickly turn into disadvantages. For a multi-developer system, it is advantageous to separate the complex functionality into independent subsystems that would then no longer rely on one single administrator whenever new types of processing engines need to be added.

H.3 Some Suggestions for Future Implementations

It is clear from the previous sections, that the currently implemented system has its limitations when it comes to dealing with multi-user (as in 'multi-developer') scenarios in an efficient way. For a further expanding data access facility, eventually a new system should be implemented, based on the long-term experience with the current one. The main parts of such a system would be

- *Web Frontends* to collect data from authorised end-users, encapsulate this data into a suitable message and send this message to the job queuing subsystem.
- A *Job Queuing Subsystem* to collect all authorised processing requests and send them to suitable processing engines, sorted by their respective priority.
- *Processing Engines* that process the job requests as managed by the job queuing subsystem and return the results in some standard format.
- *Web Frontends* that take the results as produced by the processing engine and display them to the end-user.

Note that in such a system the individual subsystems are no longer tightly coupled. A process submitting tasks to the job queuing subsystem would not have to know about any internal data formats of the job queuing subsystem. Inversely, the job queuing subsystem would no longer need to know about each individual type of job request and all the parameters associated with this particular request. Instead, it would merely act as an intermediary, passing the job request on to a suitable processing engine as soon as one becomes available, and informing some frontend when processing has finished.

Appendix I

DUNES Overview

DUNES (Dial-up NEtworking for remote Stations) provides the following functionality to a remote computer system (taken from the DUNES requirements document [Gri06b]), also included is a state diagram of the main DUNES state machine (figure I.1).

1. **Manually establish and keep up dial-up internet connection.** The local user can trigger dial-up to the internet. DUNES will try to establish a dial-up connection. The connection will be kept up (redialled) until manually terminated or until watchdog timeout.
2. **Automatically establish dial-up internet connection, transfer data, hang up.** Automated processes can ask for the internet connection to be established and taken down. Watchdog timeout will still be active.
3. **Local status information: Connecting/connected/disconnected/time-to-hangup.** Provides feedback to the local user: Is the connection up?
4. **Remote dial-in.** In offline periods, incoming phone calls can establish a TCP/IP connection with the system. Maximum connection time can be specified, after which the connection will be cut.
5. **Call back.** Remote dial-in causes system to dial up. A remote caller can dial in and trigger a dial-up. The system will hang up, dial up to the internet and stay online for a predefined period of time.
6. **Free call back.** Modem RING causes system to dial up. A RING on the modem is sufficient to trigger a short dialled-up period. Can be enabled/disabled.

7. **Scheduled regular online periods.** The system will go online during regular predefined time periods.

8. **Hang-up watchdog.** Auto hang-up after predefined time period (timeout), can be re-triggered=extended.

9. **Fallback internet providers.** The system will have a list of internet providers. If a connection with the preferred provider cannot be established, it will fall back to other providers.

10. **UDP Notification.** The system will send a UDP info packet to a list of predefined addresses whenever it comes online. This packet will include IP address, name, and additional information such as time, internet provider, etc.

11. **DynDNS registration.** The system will register with at least one dynamic DNS provider to allow easy access through fixed host names.

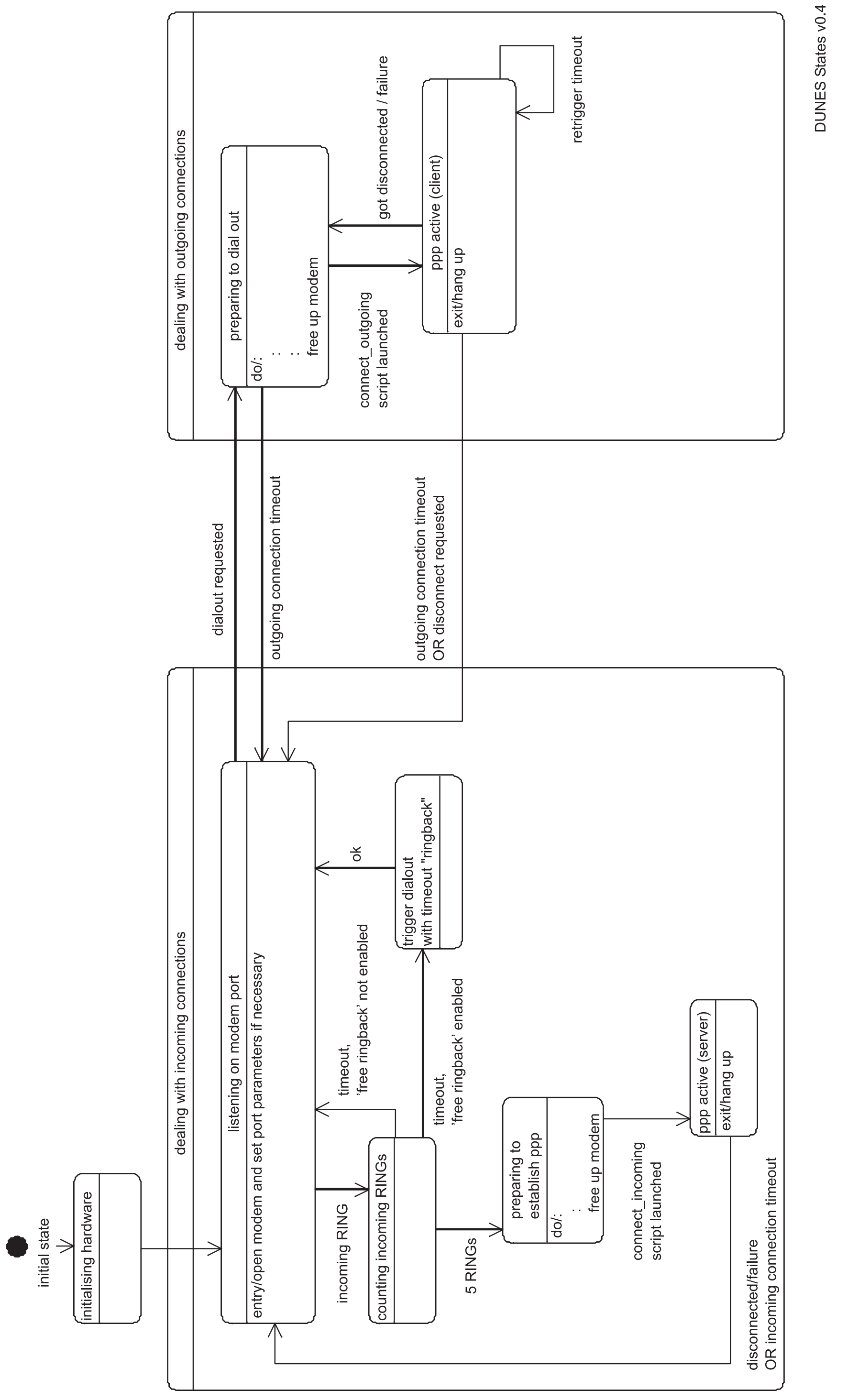


Figure I.1: State diagram of main DUNES state machine

Bibliography

[ADI] NCSA astronomy digital image library. Available from: `http://adil.ncsa.uiuc.edu/`.

[adl07] ADLINK Technology, Inc. *PCI-9812/9810 4-Ch 20MHz Simultaneous Analog Input Cards*, 2007. Available from: `http://www.adlinktech.com/PD/web/PD_detail.php?pid=33`.

[AGW72] Noach Amitay, Victor Galindo, and Chen Pang Wu. *Theory and analysis of phased array antennas*. Wiley-Interscience, New York, 1972.

[AKK05] M. Ashrafi, M. J. Kosch, and K. Kaila. Height triangulation of artificial optical emissions in the F-layer. In Kosch [Kos05], pages 8–16.

[Ans65] Z. A. Ansari. A narrow-beam antenna array for radio wave absorption studies in the auroral zone. *Proceedings of the IEEE*, 53:530–532, May 1965.

[AR02] Bertram Arbesser-Rastburg. Space weather — effects on navigation and communication systems, July 2002. Presentation at the Alpbach Summer School 2002.

[AST] Radio astronomy tutorial. Available from: `http://fourier.haystack.mit.edu/urei/tutorial.html`.

[Bar02] Les Barclay. Ionospheric effects and communication systems performance. In *Proceedings of the 10th Ionospheric Effects Symposium*, page 1. JMG Associates, Ltd., May 2002.

[Bar07] Keith Barratt. ARIES FPGA documentation. Technical report, Lancaster University, 2007.

[BGPW77] J. W. M. Baars, R. Genzel, I. I. K. Pauliny-Toth, and A. Witzel. The absolute spectrum of CAS A - an accurate flux density scale and a set of secondary calibrators. *Astronomy and Astrophysics*, 61:99–106, October 1977. Available from: `http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=1977A%26A....61...99B&db_key=AST`.

[BGS97] Bernhard F. Burke and Francis Graham-Smith. *An Introduction to Radio Astronomy*. Cambridge University Press, 1st edition, 1997.

[BHGC04] K. Barratt, F. Honary, M. Grill, and P. Chapman. Advanced Rio-Imaging Experiment in Scandinavia—ARIES: Technical implementation aspects. In Kosch [Kos04], page 65. Poster presentation.

[BHH95] S. Browne, J. K. Hargreaves, and B. Honary. An imaging riometer for ionospheric studies. *Electronics and Communication Engineering Journal*, 7(5):209–217, October 1995.

[BL61] J. Butler and R. Lowe. Beam-forming matrix simplifies design of electronically scanned antennas. *Electronic Design*, 12:170–173, 1961.

[BP77] G. J. Burke and A. J. Poggio. *Numerical Electromagnetic Code (NEC) — Method of Moments. A User-Oriented Computer Code for Analysis of the Electromagnetic Response of Antennas and other Metal Structures.* Naval Ocean Systems Center San Diego CA, 1977. Technical document.

[BPSM$^+$06] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maier, François Yergeau, and John Cowan. Extensible markup language (XML) 1.1. Technical report, The World Wide Web Consortium (W3C), 2006. Available from: `http://www.w3.org/TR/xml11/`.

[BRJ00] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Complete UML Training Course*. Prentice Hall PTR, 25th May 2000.

[Bro36] George H. Brown. The “turnstile” antenna. *Electronics*, 9:15+, April 1936.

[BSB05] Daniel J. Barrett, Richard E. Silverman, and Robert G. Byrnes. *SSH, The Secure Shell: The Definitive Guide*. O’Reilly, Sebastopol, CA, 2nd edition, 2005.

[Can75] H. V. Cane. Low frequency maps of the Galaxy. In *Astronomical Society of Australia, Proceedings*, volume 2, pages 330–331, October 1975.

[Can78] H. V. Cane. A 30 MHz map of the whole sky. *Australian Journal of Physics*, 31:561–565, 1978.

[Cas76] J. L. Caswell. A map of the northern sky at 10 MHz. *Monthly Notices of the Royal Astronomical Society*, 177:601–616, 1976.

[CHHW97] P. N. Collis, J. K. Hargreaves, W. G. Howarth, and G. P. White. Joint imaging riometer – incoherent scatter radar observations: A four-dimensional perspective on energetic particle input to the auroral mesosphere. *Advances in Space Research*, 20(6):1165–1168, 1997.

[CM58] W. N. Christiansen and D. S. Mathewson. Scanning the sun with a highly directional array. *Proceedings of the Institution of Radio Engineers*, 46:127+, 1958.

[Coo06] Mendel Cooper. *Advanced Bash-Scripting Guide*, 2006. Revision 4.2.01. Available from: `http://linuxreviews.org/beginner/abs-guide/en/`.

[CT04] John Cowan and Richard Tobin. XML information set. Technical report, The World Wide Web Consortium (W3C), 2004. Available from: `http://www.w3.org/TR/xml-infoset/`.

[Dav96] Chris Davis. An introduction to the EISCAT dynasonde, June 1996. Available from: `http://www.eiscat.rl.ac.uk/dynasonde/description.html`.

[Dea02] John Deacon. *Design patterns*. London, UK, 2002. Course material.

[Dea05] John Deacon. *Object-Oriented Analysis and Design*. Addison-Wesley, Harlow, 2005.

[dPKH02] C. F. del Pozo, M. J. Kosch, and F. Honary. Estimation of the characteristic energy of electron precipitation. *Annales Geophysicae — Atmospheres, Hydrospheres and Space Sciences*, 20(9):1349–1359, 2002.

[DR90] D. L. Detrick and T. J. Rosenberg. A phased-array radiowave imager for studies of cosmic noise absorption. *Radio Science*, 25(4):325–338, July–August 1990.

[DR94] D. L. Detrick and T. J. Rosenberg. *System Manual for the Imaging Riometer for Ionospheric Studies (IRIS)*. Institute for Physical Science and Technology, University of Maryland, 1994.

[DS90] G. R. Drevin and P. H. Stoker. Riometer quiet day curves determined by the density method. *Radio Science*, 25(6):1159–1166, November–December 1990.

[DSS95] K. S. Dwarakanath, N. Shankar, and T. S. Shankar. All sky survey at 34.5 MHz from GEETEE. *Astronomy Data Image Library*, page 1, December 1995. Available from: `http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=1995ADIL...KD...01D&db_key=AST`.

[DU90] K. S. Dwarakanath and N. Udaya Shankar. A synthesis map of the sky at 34.5 MHz. *Journal of Astrophysics and Astronomy*, 11:323–410, September 1990. Available from: `http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=1990JApA...11..323D&db_key=AST`.

[Eng02] Chad English. *TIMEBAR function for MATLAB*, 2002. Available from: `mailto:cenglish@myrealbox.com`.

[ETS97] ETSI. Digital video broadcasting (DVB); a guideline for the use of DVB specifications and standards. TR 101 200. Technical report, European Telecommunications Standards Institute, 1997. Available from: `http://www.etsi.fr`.

[ETS03] ETSI. Digital video broadcasting (DVB); specification for service information (SI) in DVB systems. TR 101 211. Technical report, European Telecommunications Standards Institute, 2003. Available from: `http://www.etsi.fr`.

[Fos06] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.

[FSL$^+$88] W. Fricke, H. Schwan, T. Lederle, U. Bastian, R. Bien, G. Burkhardt, B. Du Mont, R. Hering, R. Jährling, H. Jahreiß, S. Röser, H.-M. Schwerdtfeger, and H. G. Walter. Fifth fundamental catalogue (FK5). Part 1: The basic fundamental stars. *Veröffentlichungen Astronomisches Rechen-Institut Heidelberg*, 32:1–106, 1988.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vissides. *Design patterns*. Addison-Wesley, Reading, Mass., 1995.

[GHM$^+$07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP version 1.2 part

1: Messaging framework. Technical report, The World Wide Web Consortium (W3C), 2007. Available from: `http://www.w3.org/TR/soap12-part1/`.

[GHN$^+$03] M. Grill, F. Honary, E. Nielsen, T. Hagfors, G. Dekoulis, P. Chapman, and H. Yamagishi. A new imaging riometer based on Mills Cross technique. In *7th International Symposium on Communication Theory and Applications*, pages 26–31, Ambleside, UK, 13th–18th July 2003.

[God] God. Paul's letter to the Ephesians. *The Bible*, Eph 3:20. New International Version (NIV). Available from: `http://www.biblegateway.com/passage/?search=eph3;`.

[GPP92] G. Gustafsson, N. E. Papitashvili, and V. O. Papitashvili. A revised corrected geomagnetic coordinate system for epochs 1985 and 1990. *Journal of Atmospheric and Terrestrial Physics*, 54(11–12):1609–1631, December 1992.

[Gre03] Robin Green. *Spherical Harmonic Lighting: The Gritty Details*. Sony Computer Entertainment America, 2003. Available from: `http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.html`.

[Gri06a] Martin Grill. *ARCOM Distribution Documentation*. Lancaster, UK, 2006. Generated by Doxygen. Available from: `ARCOM/doc/html/index.html`.

[Gri06b] Martin Grill. DUNES requirements document. Technical report, Lancaster University, 2006.

[Gri06c] Martin Grill. SPARKLE requirements document. Technical report, Lancaster University, 2006.

[Gro02] William Grosso. *Java RMI*. O'Reilly, Sebastopol, CA, 2002.

[GSH05] Martin Grill, Andrew Senior, and Farideh Honary. Two new approaches to spatial interpolation with inherent sidelobe suppression for imaging riometers. In Kosch [Kos05], pages 23–36.

[Hag01a] Tor Hagfors. Comparison of the performance of the cross correlation and the filled aperture imaging riometers. Written during his visit to the Lancaster University, November 2001.

[Hag01b] Tor Hagfors. Some notes on the cross correlation imaging riometer. Written during his visit to the Lancaster University, October 2001.

[Hai85] G. V. Haines. Spherical cap harmonic analysis. *Journal of Geophysical Research*, 90(B3):2583–2591, 1985.

[Hai88] G. V. Haines. Computer programs for spherical cap harmonic analysis of potential and general fields. *Computers and Geosciences*, 4(4):413–417, 1988.

[Har69] J. K. Hargreaves. Auroral absorption of HF radio waves in the ionosphere: A review of results from the first decade of riometry. *Proceedings of the IEEE*, 57(8):1348–1373, August 1969.

[Har95] J. K. Hargreaves. *The Solar-Terrestrial Environment*. Cambridge University Press, 1995.

[HCG$^+$05] F. Honary, P. Chapman, M. Grill, K. Barratt, S. Marple, E. Nielsen, and T. Hagfors. Advanced rio-imaging experiment in Scandinavia (ARIES): System specification and scientific goals. In *URSI General Assembly*, New Delhi, India, 2005.

[HCG$^+$06] F. Honary, P. Chapman, M. Grill, K. Barratt, S. Marple, E. Nielsen, and T. Hagfors. Advanced Rio-Imaging Experiment in Scandinavia (ARIES): First observations. In *2nd International Riometer Workshop*, page 5, Banff, Alberta, Canada, 25th March 2006.

[HD02] J. K. Hargreaves and D. L. Detrick. Application of polar cap absorption events to the calibration of riometer systems. *Radio Science*, 37(3):7/1–7/11, 2002.

[HFM02] Steven Hutsell, Matthew Forsyth, and Charles B. McFarland. One-way GPS time transfer: 2002 performance. In *Proceedings of the 34th annual precise time and time interval (PTTI) meeting*, pages 69–76, 2002. Available from: `http://tycho.usno.navy.mil/ptti/ptti2002/paper6.pdf`.

[HGBC07] Farideh Honary, Martin Grill, Keith Barratt, and Peter Chapman. The Advanced Rio-Imaging Experiment in Scandinavia. *submitted to Radio Science*, July 2007.

[HGH03] T. Hagfors, M. Grill, and F. Honary. Performance comparison of cross correlation and filled aperture imaging riometers. *Radio Science*, 38(6):17/1–17/5, 30th December 2003.

[HH68] P. A. Hamilton and R. F. Haynes. Observations of the southern sky at 10.02 MHz. *Australian Journal of Physics*, 21:895, 1968.

[Hie95] Jarkko Hietaniemi. Comprehensive Perl archive network (CPAN), 1995. Available from: `http://www.cpan.org/`.

[HMK$^+$04] F. Honary, S. R. Marple, M. Kosch, A. Senior, R. A. Makarevitch, and M. Grill. The past, present and future of imaging riometry at Lancaster. In Kosch [Kos04], page 22.

[Hob55] Ernest William Hobson. *The theory of spherical and ellipsoidal harmonics*. Chelsea Pub. Co., New York, 1955.

[Hon01] Farideh Honary. *High-Resolution Imaging Riometer, Case for support*. Lancaster University, 2001.

[HV99] Michi Henning and Steve Vinoski. *Advanced CORBA programming with C++*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[IEE07] IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, 2007. Available from: `http://www.swebok.org/`.

[IER] Homepage of the International Earth Rotation Service. Available from: `http://hpiers.obspm.fr/eop-pc/`.

[IRI] Imaging riometer for ionospheric studies. Available from: `http://www.dcs.lancs.ac.uk/iono/iris`.

[Jen66] Roger Clifton Jennison. *Introduction to Radio Astronomy*. Newnes, London, 1966.

[Jon02] Tudor Jones. Lecture notes: MSc in satellite communications and space environment, module 5: Solar terrestrial (satellite) environment, February 2002. Lancaster University.

[JT98] Michael K. Johnson and Erik W. Troan. *Linux application development*. Addison-Wesley, Reading, Mass., 1998.

[Kas88] N. E. Kassim. The Clark Lake 30.9 MHz Galactic plane survey. *Astrophysical Journal Supplement Series*, 68:715–733, December 1988. Available from: `http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=1988ApJS...68..715K&db_key=AST`.

[Kav02] Andrew John Kavanagh. Energy deposition in the lower auroral ionosphere through energetic particle precipitation. PhD thesis, University of Lancaster, UK, November 2002.

[KDR85] S. Krishnaswamy, D. L. Detrick, and T. J. Rosenberg. The inflection point method of determining riometer quiet day curves. *Radio Science*, 20(1):123–136, January–February 1985.

[Kos04] Mike Kosch, editor. *31st Annual European Meeting on Atmospheric Studies by Optical Methods and 1st International Riometer Workshop*, Ambleside, UK, 22nd–28th August 2004.

[Kos05] M. J. Kosch, editor. *Proceedings of the 31st Annual European Meeting on Atmospheric Studies by Optical Methods and 1st International Riometer Workshop*, Ambleside, UK, December 2005. Lancaster University.

[KPW69] K. I. Kellermann, I. I. K. Pauliny-Toth, and P. J. S. Williams. The Spectra of Radio Sources in the Revised 3c Catalogue. *Astrophysical Journal*, 157:1, July 1969. Available from: `http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=1969ApJ...157....1K&db_key=AST`.

[Kra88] John D. Kraus. *Antennas*. McGraw-Hill, New York, 1988.

[La 74] La Jolla Sciences. *Solid State Riometer Manual*, April 1974. reprinted October 1980.

[LC01] Bo Leuf and Ward Cunningham. *The Wiki way: quick collaboration on the Web*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[LL59] C. G. Little and H. Leinbach. The riometer - a device for the continuous measurement of ionospheric absorption. *Proceedings of the IRE*, 47:315–320, February 1959.

[Mara] S. Marple. MIA function getdata(), type ‘help getdata’ in MIA to get the function description.

[Marb] S. Marple. MIA function setresolution(), type 'help mia_base/setresolution.m' in MIA to get the function description.

[Marc] S. Marple. Multi-Instrument Analysis (MIA) toolkit. Available from: `http://www.dcs.lancs.ac.uk/iono/mia`.

[Mard] S. Marple. personal communication 13/01/2003.

[Mare] S. Marple. Riometer and imaging riometer database. Available from: `http://www.dcs.lancs.ac.uk/iono/cgi-bin/riometers`.

[Mata] The Mathworks, Inc. *MATLAB Distributed Computing Toolbox*. Available from: `http://www.mathworks.com/products/distribtb/index.html`.

[Matb] The Mathworks, Inc. *MATLAB Function Reference*. Available from: `http://www.mathworks.com/products/matlab/functionlist.html`.

[Matc] The Mathworks, Inc. *MATLAB Signal Processing Toolbox*. Available from: `http://www.mathworks.com/products/signal/`.

[Max07] Maxim/Dallas. *1-Wire Devices*, 2007. Available from: `http://www.maxim-ic.com/products/1-wire/`.

[MBC65] D. S. Mathewson, N. W. Broten, and D. J. Cole. A survey of the southern sky at 30 Mc/s. *Australian Journal of Physics*, 18:665, 1965.

[MH04] S. R. Marple and F. Honary. A multi-instrument data analysis toolbox. *Advances in Polar Upper Atmosphere Research*, 18:120–130, September 2004.

[MH07] Steve Marple and Farideh Honary. Removal of solar radio emissions and determination of riometer quiet day curves. *In preparation*, 2007.

[MHMH04] R. A. Makarevitch, F. Honary, I. W. McCrea, and V. S. C. Howells. Imaging riometer observations of drifting absorption patches in the morning sector. *Annales Geophysicae — Atmospheres, Hydrospheres and Space Sciences*, 22(10):3461–3478, 3rd November 2004.

[Mic07] Microsoft. *File Systems*, 2007. Available from: `http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/prork/prdf_fls_pxjh.mspx`.

[Mil52] B. Y. Mills. The distribution of the discrete sources of cosmic radio radiation. *Australian Journal of Scientific Research*, 5(2):266–287, 1952.

[ML53] B. Y. Mills and A. G. Little. A high-resolution aerial system of a new type. *Australian Journal of Physics*, 6:272–278, 1953.

[MLSS58] B. Y. Mills, A. G. Little, K. V. Sheridan, and O. B. Slee. A high resolution radio telescope for use at 3.5m. *Proceedings of the Institution of Radio Engineers*, 46:67, 1958.

[MMK$^+$97] Yasuhiro Murayama, Hirotaka Mori, Shoji Kainuma, Mamoru Ishi, Ichizo Nishimuta, Kiyoshi Igarashi, Hisao Yamagishi, and Masanori Nishino. Development of a high-resolution imaging riometer for the middle and upper atmosphere observation program at Poker Flat, Alaska. *Journal of Atmospheric and Solar-Terrestrial Physics*, 59(8):925–937, 1997.

[Moo64] H. J. Moody. The systematic design of the Butler Matrix. *IEEE Transactions on Antennas and Propagation*, 12(6):786–788, November 1964.

[MS53] A. P. Mitra and C. A. Shain. The measurement of ionospheric absorption using observations of 18.3 mc/s cosmic radio noise. *Journal of Atmospheric and Terrestrial Physics*, 4:204, 1953.

[MTS73] Jelana Milogradov-Turin and F. G. Smith. A survey of the radio background at 38 MHz. *Monthly Notices of the Royal Astronomical Society*, 161:269–279, 1973.

[Mue72] Paul J. Muenzer. Properties of linear phased arrays using Butler Matrices. *Naturwissenschaftliche Technische Zeitschrift*, 9:419–422, 1972.

[Mur] Y. Murayama. High-resolution imaging riometer at Poker Flat. Available from: `http://www2.crl.go.jp/ck/ck421/riometer.html`.

[NBF96] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell. *Pthreads programming*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1996.

[New05] Cameron Newham. *Learning the Bash Shell*. O'Reilly, Sebastopol, CA, 3rd edition, 2005.

[NH97] E. Nielsen and T. Hagfors. Plans for a new rio-imager experiment in Northern Scandinavia. *Journal of Atmospheric and Solar-Terrestrial Physics*, 59(8):939–949, 1997.

[NHG04] E. Nielsen, F. Honary, and M. Grill. Time resolution of cosmic noise observations with a correlation experiment. *Annales Geophysicae — Atmospheres, Hydrospheres and Space Sciences*, 22(5):1687–1689, 8th April 2004.

[Nie91] Richard O. Nielsen. *Sonar Signal Processing*. Artech House, 1991.

[Nie01] E. Nielsen. Antenna system for a high resolution imaging riometer. Technical Report MPAE-W-03-01-04, Max-Planck-Institut für Aeronomie, Lindau, Germany, 28th February 2001.

[Nie02a] Erling Nielsen. Email "ARIES first results" to Peter Chapman, November 2002.

[Nie02b] Erling Nielsen. Time resolution of observations with HRIR. Technical report, Max Planck Institut für Aeronomie, 37191 Katlenburg–Lindau, Germany, April 2002.

[NTPa] NTP: The Network Time Protocol. Available from: `http://www.ntp.org/`.

[NTPb] NTP timescale and leap seconds. Available from: `http://www.eecis.udel.edu/~ntp/ntp_spool/html/leap.htm`.

[Obj05] Object Management Group, Inc. *The OMG's CORBA Website*, 2005. Available from: `http://www.corba.org/`.

[Obsa] U.S. Naval Observatory. Historical list of leap seconds. Available from: `ftp://maia.usno.navy.mil/ser7/tai-utc.dat`.

[Obsb] U.S. Naval Observatory. Leap seconds. Available from: `http://tycho.usno.navy.mil/leapsec.990505.html`.

[Owe] G. Scott Owen. 3D rotation. Available from: `http://www.siggraph.org/education/materials/HyperGraph/modeling/mod_tran/3drota.htm`.

[Par07] Chris Park, editor. *Submission of Theses*. Lancaster University, 2007. Available from: `http://www.lancs.ac.uk/users/gradschool/theses.html`.

[Paw05] Rich Pawlowicz. *M_Map Mapping Package for MATLAB*. University of British Columbia, Vancouver, Canada, 2005. Available from: `http://www.ocgy.ubc.ca/~rich/`.

[PFTV88] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vettering. *Numerical Recipes in C*. Cambridge University Press, 1st edition, 1988.

[PR00] Arno Puder and Kay Römer. *MICO*. Morgan Kaufmann Publishers and dpunkt.verlag, 3rd edition, 2000.

[Qui03] Liam Quin. Extensible markup language (XML), August 2003. Available from: `http://www.w3.org/XML/`.

[RDVvB91] T. J. Rosenberg, D. L. Detrick, D. Venkatesan, and G. van Bavel. A comparitive study of imaging and broad-beam riometer measurements: The effect of spatial strucure on the frequency dependence of auroral absorption. *Journal of Geophysical Research — Space Physics*, 96(A10):17793–17803, October 1991.

[Rei85] D. Reidel. *Maximum-Entropy and Bayesian Methods in Inverse Problems*. Fundamental theories of physics. Kluwer Academic Publishers, 1985.

[Ric77] Barney J. Rickett. Interstellar scattering and scintillation of radio waves. *Annual Review of Astronomy and Astrophysics*, 15:479–504, September 1977.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, MA, USA, 1999.

[RTC] Javascript application for real-time display of several timescales. Available from: `http://www.leapsecond.com/java/gpsclock.htm`.

[San92] A. De Santis. Conventional spherical harmonic analysis for regional modelling of the geomagnetic field. *Geophysical Research Letters*, 19:1065–1067, 1992.

[SCT03] *7th International Symposium on Communication Theory and Applications*, Ambleside, UK, 13th–18th July 2003.

[SCW] Eric Weisstein's world of science. Available from: `http://scienceworld.wolfram.com`.

[Sen] A. Senior, personal communication.

[Sha58] C. A. Shain. The Sydney 19.7-mc radio telescope. *Proceedings of the Institution of Radio Engineers*, 46:85+, 1958.

[SI] The international system of units (SI). Available from: `http://www.bipm.fr/enus/3_SI/si.html`.

[Sin50] G. Sinclair. The transmission and reception of elliptically polarized waves. *Proceedings of the IRE*, 38:151, 1950.

[SK05] Joshua Semeter and Farzad Kamalabadi. Determination of primary electron spectra from incoherent scatter radar measurements of the auroral E region. *Radio Science*, 40(RS2006), 2005. doi:10.1029/2004RS003042.

[SKVa] SkyView survey document. Available from: `http://skys.gsfc.nasa.gov/cgi-bin/survey.pl`.

[SKVb] SkyView, the internet's virtual telescope. Available from: `http://skys.gsfc.nasa.gov`.

[SLM$^+$90] G. Sironi, M. Limon, G. Marcellino, G. Bonelli, M. Bersanelli, G. Conti, and K. Reif. The absolute temperature of the sky and the temperature of the cosmic background radiation at 600 MHz. *The Astrophysical Journal*, 357:301–308, July 1990. Available from: `http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=1990ApJ...357..301S&db_key=AST`.

[Sma62] W. M. Smart. *Spherical Astronomy*. Cambridge University Press, 5th edition, 1962.

[Som04] Ian Sommerville. *Software Engineering*. Addison Wesley, 7th edition, 2004.

[ST97] A. De Santis and J.M. Torta. Spherical cap harmonic analysis: a comment on its proper use for local gravity field representation. *Journal of Geodesy*, 71:526–532, 1997.

[Ste98] W. Richard Stevens. *Unix Network Programming*, volume 2. Prentice-Hall PTR, 1998.

[Sun05] Sun Microsystems, Inc. *Java^{TM}Web Start Overview*, May 2005. Available from: `http://java.sun.com/developer/technicalArticles/WebServices/JWS_2/JWS_White_Paper.pdf`.

[Tao04] Huiyu Tao. Impact of ionospheric disturbance on HF radio communications. MPhil thesis, University of Lancaster, UK, 2004.

[TCC$^+$03] Angela C. Taylor, Pedro Carreira, Kieran Cleary, Rod D. Davies, et al. First results from the Very Small Array II – observations of the CMB. *Monthly Notices of the Royal Astronomical Society*, 341(4):1066–1075, June 2003.

[TMGWS86] A. Richard Thompson, James M. Moran, and Jr. George W. Swenson. *Interferometry and Synthesis in Radio Astronomy*. John Wiley & Sons, New York, London, Sydney, 1986.

[Tro] Erik W. Troan. UNIX MAN page for POPT. Available from: `man_3_popt`.

[TUP$^+$87] N. A. Tsyganenko, A. V. Usmanov, V. O. Papitashvili, N. E. Papitashvili, and V. A. Popov. *Software for computations of the geomagnetic field and related coordinate systems*. Soviet Geophys. Comm., Moscow, 1987.

[Uni07] University of Chicago. *The Globus Alliance Homepage*, 2007. Available from: `http://www.globus.org/`.

[vH06] Dimitri van Heesch. Doxygen, 2006. Available from: `http://www.doxygen.org`.

[Vin98] Fiona Vincent. Lecture notes on Positional Astronomy, February 1998. Available from: `http://star-www.st-and.ac.uk/~fv/webnotes/index.html`.

[Wala] Larry Wall. UNIX MAN page for the practical extraction and report language (PERL). Available from: `man_perl`.

[Walb] P. T. Wallace. *SLALIB — Positional Astronomy Library — Programmer's Manual*. Available from: `http://star-www.rl.ac.uk/star/docs/sun67.htx/sun67.html`.

[WCS96] Larry Wall, Tom Christiansen, and Randal L. Schwartz. *Programming Perl*. O'Reilly, Sebastopol, CA, 2nd edition, 1996.

[Wei02] Eric W. Weisstein. *CRC Concise Encyclopedia of Mathematics*. Chapman & Hall/CRC, 2nd edition, December 2002.

[Wel44] N. Wells. The quadrant aerial: An omni-directional wide-band horizontal aerial for short waves. *Journal of the Institution of Electrical Engineers*, 91(III):182, December 1944.

[Wil06] J. Wild. *Open letter to Prof. Keith Mason, Chief Executive, PPARC*. Lancaster University, May 2006.

[Yam] H. Yamagishi, personal communication.