Ship's Radar data acquisition system and polar spectral analysis of sea wave patterns

van de Laar, J.

Award date:
2000

Ship's Radar data acquisition system and polar spectral analysis of sea wave patterns

Master's Thesis

Jakob van de Laar
Id. Nr: 420076

Supervisors:
P.C.W. Sommen (TUE)
J.C.M. Kleijweg (TNO-FEL)
T.J.M. Jeurens (Tech 5 B.V.)

Date : 17-08-2000

Summary

Radar waves reflected from the sea surface can be used to obtain oceanographic information. They are governed by the sea surface roughness that in turn is determined by the wind, waves, oil-slick contamination and even by topographic features of the sea bottom. SHIRA (an acronym for Ship's Radar) is a remote sensing system that has been developed by TNO-FEL to satisfy the need for measurements of directional wave spectra, water currents and water depths. The digitized data registered with the SHIRA is subjected to spectral analysis, which results in (directional) wave spectra. Together with a PC, a digitizer and an interface, SHIRA images the surrounding area for every rotation of the antenna. The back-scattered signals of a user-specified region are digitized by a data acquisition board and the data is transferred to the PC for further processing and storage. This way a series of images is registered, in which not only spatial patterns can be seen but also their temporal behavior. The PC is not only used for data handling and analysis, but it is also a controller of the system. With the PC, the location and size of the selected region, pixel spacing, etc. can be adjusted. Limits on these parameters are set by the capacity of the data acquisition system.

The project described in this report consisted of two parts. Firstly, because the current data acquisition system cannot deal with high pulse repetition rates, a new high-speed (40 MHz) data acquisition system has been implemented. Drivers have been written for the three main data acquisition components: a data acquisition board, a counter board and a time stamp board. A high level of modularity has been achieved because a separate driver has been programmed for each data acquisition component, instead of making one big monolithic driver as in the current data acquisition system. In addition to the driver software, user-mode software that interacts with these drivers has been developed. Tests have revealed that the data acquisition system complies with the given requirements and specifications. It can deal with pulse repetition rates up to 4 kHz while the sweep length (the number of samples per pulse return) is 4096, whereas the old system can deal with pulse repetition rates up to 2 kHz while the sweep length is 2048. Moreover, in contrast with the old system that can only deal with sweep lengths that are powers of two, the new system can deal with any sweep length up to 524288.

The second part of the project concerned the spectral analysis of the data obtained with SHIRA. The data is available on a polar grid that is nonuniform in the azimuth (angular) direction only. At present, the individual images out of a time series are first scan converted, i.e. they are resampled from the nonuniform polar grid onto a Cartesian grid, and next the two-dimensional FFT is computed. For the resampling the Nearest Neighborhood (NN) interpolation method is used. However, TNO-FEL observed that the spectra obtained this way seemed to be distorted, especially for higher spatial frequencies. They expected that this distortion was caused by the Nearest Neighborhood interpolation of the available polar samples onto the Cartesian grid. This expectation has been examined and it was found that indeed some distortion was present in the spectra for high spatial frequencies (i.e. large wave numbers). It is possible to eliminate the distortion totally because methods exist that ideally reconstruct two-dimensional signals from nonuniform samples in polar coordinates if the function is band limited. However, this involves an unacceptably high complexity. In order to try to diminish the distortion due to the NN interpolation and to obtain better spectra in general, an alternative method named Discrete Polar Fourier Transform (DPFT) has been examined and its performance regarding the quality of the spectra has been compared to the currently used method. The comparison was mainly based on the amount of distortion in the spectra computed with both methods. It has been shown that the DPFT performs better than the current method, especially when windows are used to smoothly taper the data to zero at the edges of the data window. However, in case windowing is used, the price paid for better signal-to-noise ratios by both the DPFT and the current method is a reduced spectral resolution.

In addition to the comparison, a method for indirectly computing the DPFT via Hankel transforms has been examined because with this method the DPFT can be computed within reasonable time. From a complexity analysis, it can be concluded that the new method is probably faster than the old one for large analysis areas (this still has to verified). This can be a great advantage since the user probably wants to analyze areas as large as possible and the currently used analysis areas are restricted by computation times.

Acknowledgments

I would like to thank Ir. J.C.M. Kleijweg from TNO-FEL for giving me the opportunity to work on a project from TNO-FEL, the well-known Dutch company for applied scientific research. In addition, I would like to thank him for his explanations, advice, suggestions and all other kinds of support.
I'm also grateful to my supervisor Ir. T.J.M. Jeurens from Tech 5 B.V., the company that worked on the project for TNO-FEL, for his advice, valuable suggestions, and for critically reviewing the first part of the manuscript of this report. I am also indebted to Ing. J. van Heesen, also from Tech 5 B.V., for his continuous support during difficult programming tasks (especially the development of drivers).
I am also most grateful to Dr. ir. P.C.W. Sommen, my supervisor from the Eindhoven University of Technology (TUE), for his continuous support, guidance, advice, suggestions, critical questions and comments, critical reviews of the entire manuscript of this report, and many other helpful inputs.
In addition, I would like to thank Prof. dr. ir. J.W.M. Bergmans and Prof. dr. ir. A.G. Tijhuis (both from the TUE) for critically reviewing the entire manuscript and for their valuable suggestions and advice.
Last, but by no means least, I am most thankful to my wife Ria for her patience, encouragement and all other kinds of support. I am indebted to her because of the sacrifices that were made during the last months of the project.

Table of contents

1 Introduction

As a part of the electrical engineering study at the Eindhoven University of Technology, a final graduation project for obtaining the Master of Science degree has been carried out at a company named Tech 5 B.V. This company is involved in a project for TNO-FEL, which is a company performing applied scientific research. The supervising research group from the Eindhoven University of Technology, named "Signal Processing Systems", is a subsection of the "Measurement and Control Systems (MBS)" group, which in turn is a part of the department of Electrical Engineering.

TNO-FEL uses a standard navigation radar to obtain information about sea wave patterns. Radar waves reflected from the sea surface can be used to obtain oceanographic information. They are governed by the sea surface roughness that in turn is determined by the wind, waves, oil-slick contamination and even by topographic features of the sea bottom. SHIRA (an acronym for Ship's Radar) is a remote sensing system that has been developed to satisfy the need for measurements of directional wave spectra, water currents and water depths. The digitized data registered with the SHIRA is subjected to spectral analysis, which results in (directional) wave spectra. From these spectra, other quantities can be derived, such as the water current vector and the water depth. It is also possible to obtain information about stationary features like oil spills, sea bottom topography (under certain hydro-meteo conditions), etc. by integrating images of the same area during a certain time (see [Kleijweg and Greidanus, 1994] for more information). SHIRA is based on a pulsed navigation radar, which can monitor the sea wave pattern surrounding the antenna if the clutter suppression is switched off and if the radar meets certain criteria such as sensitivity, rotation speed, pulse length, etc. When the radar is used as a navigation radar, the clutter suppression is switched on in order to see objects like ships, buoys, etc. instead of waves. Together with a PC, a digitizer and an interface, SHIRA images the surrounding area for every rotation of the antenna, i.e. the back-scattered signals of a user-specified region are amplified, demodulated by a logarithmic detector and digitized by a data acquisition board. This way a series of images is registered, in which not only spatial patterns can be seen but also their temporal behavior. The data is transferred from the data acquisition board to the PC for further processing and storage. The PC is not only used for data handling and analysis, but it is also a controller of the system. With the PC, the location and size of the selected region, pixel spacing, etc. can be adjusted. Limits on these parameters are set by the capacity of the data acquisition system.
The data acquisition system obtains the sample values on a nonuniform polar grid. While the radar antenna rotates, pulses are emitted and the corresponding pulse returns are sampled. The angle at which each pulse is emitted is recorded with each sweep. These angles and the distances from the radar at which samples are taken define the polar sampling grid (i.e. the data is collected in range-azimuth coordinates). Because the angular velocity of the radar antenna is not constant, due to the wind for example, the sampling of the sweeps is non-equidistant in the azimuth (angular direction); the sampling in the range direction however, is equidistant.

The purpose of the final graduation project is bipartite. In the first place, because the current data acquisition system cannot deal with high pulse repetition rates, a new high-speed data acquisition system to digitize and store the radar reflections has to be implemented. This implementation mainly takes the form of developing drivers for the data acquisition components of the radar system and software for interfacing with the user. The first 6 chapters of the report deal with this subject. Chapter 2 describes the requirements of the data acquisition system, while chapters 3 and 4 describe the design and implementation of the system. In Chapter 5, the results of tests are given and in Chapter 6 conclusions regarding the data acquisition system are drawn.
The second purpose of the project concerns the spectral analysis of the acquired raw data. At present, the individual images out of a time-series are first scan converted, i.e. they are resampled from the nonuniform polar grid onto a Cartesian grid, and next the two-dimensional FFT is computed. However, the spectra obtained this way seem to be distorted, especially for higher frequencies. Because using the polar-grid samples directly intuitively seems to exploit the given information better, the performance of the Discrete Polar Fourier Transform (DPFT) regarding the quality of the spectra is investigated. This subject is covered in the second part of the report starting with Chapter 7, which

gives an overview of the current SHIRA signal processing operations. Next, the DPFT is discussed in Chapter 8. Chapter 9 explains how the quality of the spectra is measured and in Chapter 10 a comparison between the current method and the DPFT is made. Finally, conclusions and recommendations regarding the spectral analysis are given in Chapter 11. In the appendices, a summary of used parameter definitions and the specifications of the SHIRA data acquisition system can be found. In addition, the most important Matlab simulation files are listed.

2 The SHIRA data acquisition system

2.1 Overview

SHIRA is a pulsed navigation radar. The basic principle of such a pulsed radar [Lynn, 1987] is as follows: the transmitter emits a continuous train of short radio-frequency pulses. The target range can then be found by measuring the time for echoes to return to the receiver. Moreover, the radar antenna rotates so as to image a two-dimensional area. For each pulse, transmitted and received in one specific direction, the information obtained is considered to be one-dimensional in the range direction. An angular encoder, mounted on the (vertical) axis of the radar antenna, provides information about the angular position of the antenna with respect to a reference point that is fixed with respect to the ship (or with respect to the land, in case the SHIRA is operated from land). Combining the information from all pulses transmitted at different angles with the digitized range information yields a two-dimensional data structure. In normal operation a time series of images of the same area is registered. The individual images contain information about the two-dimensional spatial structure of the sea surface, while the time series also contains information about its temporal behavior.
The introduction pointed out that this part of the report deals with the data acquisition system of the SHIRA, i.e. the system that digitizes and records a time series of images together with other relevant information. The radar system provides four signals that are relevant for the data acquisition system. The most important signal is the received back-scattered echo signal of a pulse transmitted by the radar. This signal is called *sweep* or *pulse return*. Secondly, the radar provides *trigger* pulses that are generated at the moments the pulses are emitted. The angular encoder supplies the last two signals. Firstly, it generates a pulse for each fixed angular displacement (the size of which is determined by the number of pulses that is generated in one complete revolution of the antenna). Secondly, for each full rotation of 360 degrees it also generates a reset pulse at the reference point mentioned above, called *North Reset*, indicating a complete rotation of the antenna. All digitized information is transferred to a host PC, which is the controller of the system. With the PC, the location and size of the selected region, sample frequency, etc. can be adjusted. Limits on these parameters are set by the capacity of the data acquisition system.
The four signals described in this section, being the input signals for the data acquisition system provided by the radar system, will be used for describing the main tasks and components of the data acquisition system in the following sections.

2.2 The data acquisition system

2.2.1 Requirements of the data acquisition system

In order to be able to design a system, the specifications of the system to be designed have to be known. The quantitative specifications are given in terms of parameter values. The parameter definitions used in this section and throughout the first part of the report can be found in Appendix A. The quantitative specifications of the data acquisition system are listed in Appendix A. Both the parameter definitions and the quantitative specifications are described more elaborately in [Laar, 1999]. The quantitative requirements for the data acquisition system can be described by these parameters and specifications. Additionally, for the design of the data acquisition system, a qualitative description is necessary to develop the architecture. The operation and requirements of the data acquisition system are described mainly in a qualitative way. The quantitative values will only be given when necessary.
The requirements for the data acquisition system can be stated as follows: **Digitize and record raw echo data of each sweep that is located in a user-defined region and store angular and time stamp information for each acquired sweep. The acquisition parameters must be set by software that runs on the Windows NT 4.0 operating system. The system must also be able to record a time series of image scans, where each scan corresponds to one revolution of the antenna.** These

requirements will be elaborated on in the next three sections and the software will be discussed in the next chapter.

2.2.2 Digitization of the sweeps

The main task of the data acquisition system is to *digitize* and store the sweeps from the radar that are in a user-defined region in real-time without loss of data. The number of samples acquired per sweep (the sweep length N_{RW}, see Figure 2-1 and Appendix A) has to be adjustable. This parameter relates to the range depth of the region of interest. The samples of the sweeps have to be transferred either to memory or to both memory and disk. The start range of the recording window (S_R), which determines at what distance from the antenna the radar starts "looking", also has to be adjustable (see Figure 2-1). This start range can be adjusted by delaying the trigger pulses generated by the radar each time a pulse is emitted. Simple hardware can be used for this purpose. The analog-to-digital conversion of a sweep can then be started upon the occurrence of such a delayed trigger pulse. However, this feature will not be realized in this project (only the required hardware is indicated in the next section). The start and end azimuths (S_A and E_A respectively) of the area to be registered also have to be adjustable. S_A and E_A define the angles between which sweeps have to be acquired, and they are defined with respect to the *North Reset* position (see Section 2.1). From the above, it follows that the user-defined region or recording window can be adjusted by adjusting N_{RW}, S_R, S_A and E_A (see figure). In the figure, it is shown that the last two parameters determine the azimuth span of the recording section, i.e. the number of sweeps between the start and end azimuth (this number is defined as N_M, the number of sweeps per scan).

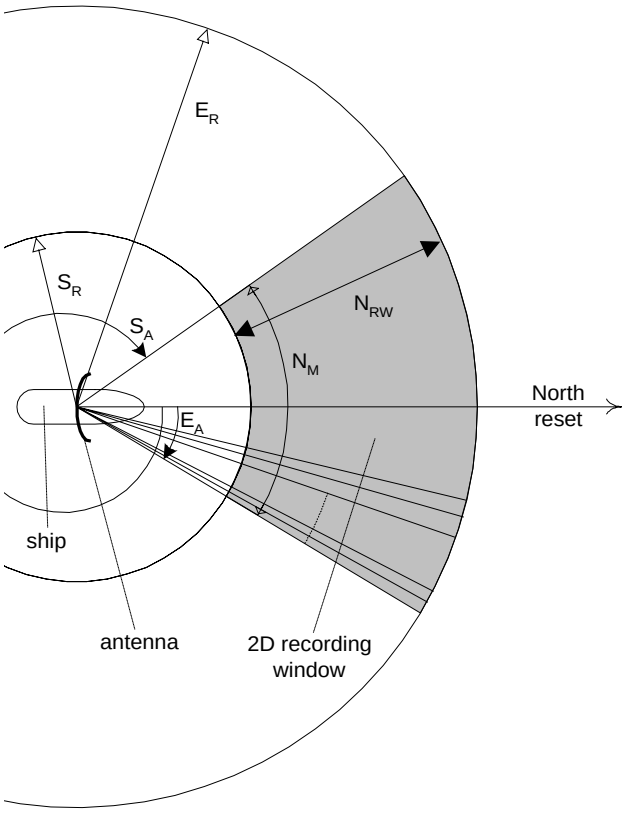

Figure 2-1 Recording window parameters

2.2.3 Recording angular information

The angle at which each pulse is emitted (and the corresponding pulse return is received) has to be recorded for each sweep because this information is necessary for reconstructing the two-dimensional image corresponding to one rotation of the antenna. Because the angular velocity of the antenna is not constant (due to the wind, for example), the angular displacement between two successive sweeps is

not constant. This means that the sampling is not equidistant in the azimuth direction. As has been pointed out already, the angular encoder that is mounted on the antenna axis generates pulses for fixed angular displacements, and for each full rotation of 360 degrees it generates a (north) reset pulse. Therefore, by *counting* these displacement-indicating pulses, the angle with respect to the *North Reset* position can be determined and then stored into memory (and to disk).

2.2.4 Recording time stamps

Time stamps have to be recorded for each sweep for the following reasons. Firstly, the time stamps are necessary to extract the temporal information (see the second part of the report). Secondly, they are needed in order to determine whether sweeps have been lost. This can be done in real-time or after a measurement has been completed. Furthermore, when synchronization of data acquired by different independent radar systems with equal data acquisition systems is required, satellite-provided time signals (GPS times) can offer a solution.

Given specific hardware components that realize the functions described above, a data acquisition system was designed and implemented. The next chapters describe this design and implementation process, and some test results.

3 Design and hardware implementation

3.1 Introduction

Several hardware and software components are needed to realize the data acquisition system.
In this chapter, the employed acquisition hardware components, their drivers and the other involved software components are described. Also the interrelationships between the components and drivers will be pointed out. Firstly, the operation and characteristics of the used hardware devices are described. Specific features of the device(s) that have a (great) impact on the design of the driver(s) will be elucidated. Next, given the functionality of the used hardware, a high-level system will be designed. This means that the operation and timing of the system is described in general terms in a descriptive way. The high-level design will then be refined during the discussion and development of the software. It will be pointed out what kinds of software components are required: user-mode software and kernel-mode software (drivers). Finally, the software is discussed together with the implementation and design decisions.

3.2 The data acquisition hardware

3.2.1 Required hardware

From the sections in the previous chapter, it can be determined what components are needed to realize the data acquisition system. The most important component is the *data acquisition board*, which performs the analog-to-digital conversion. The board must contain at least one channel and it must be able to start the conversion upon the occurrence of a trigger signal provided by the radar system. Figure 3-1 shows the connection of the radar signals to the data acquisition board (and the other data acquisition hardware components).

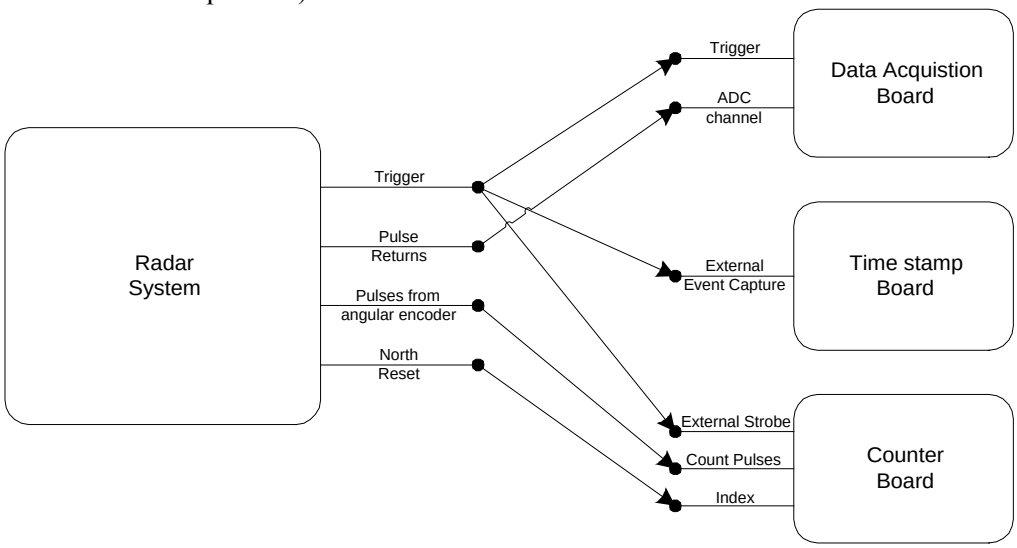

Figure 3-1 Connection of radar signals to the data acquisition components

As has been explained in Section 2.2.2, the start range of the recording window can be made adjustable by delaying the trigger pulse the radar generates for each transmitted pulse by an amount of time that corresponds to the desired distance (denoted S_R in Figure 2-1). This can be implemented by a *programmable delay line*, for example.
Time stamps can be recorded with a so-called *time stamp board*. Such a board is able to generate accurate time stamps from a satellite provided (GPS) signal or from an internal crystal clock. Furthermore, the board must be able to latch the time upon the occurrence of a trigger signal from the radar. Another possibility for time stamping is software time stamping. In this case, time stamps are provided by the host computer that controls the data acquisition.

However, this is not an option when synchronization of data acquired by different independent radar systems with identical data acquisition systems is required. The angular information for each sweep can be obtained by counting the pulses from the angular encoder. The pulse and the *North Reset* signals are fed into a *counter board*, which counts the pulses and resets the counter upon the occurrence of a reset pulse. This counter board must also be able to latch the current counter value upon the occurrence of a trigger signal because these values must be stored.
The next three sections describe the employed data acquisition hardware because knowledge about the operation of these devices and their peculiarities is required for the design of the system.

3.2.2 The data acquisition board

Because the nominal sampling frequency of the system is 40 MHz (see Appendix A), a high-speed data acquisition board has to be used. For the application the ICS650 board, made by ICS in Canada, is selected. The hardware characteristics of this board that are important for the SHIRA application are described now. A more extensive description can be found in [ICS650 Operating Manual]. The ICS650 is a 2-channel, 12 bit, 65 MHz/Channel PCI board that has been designed for a wide range of data acquisition applications and it is particularly well suited for applications in radar, communications, ultrasound, image data capture, etc. A simplified block diagram of the ICS650 is shown in Figure 3-2.

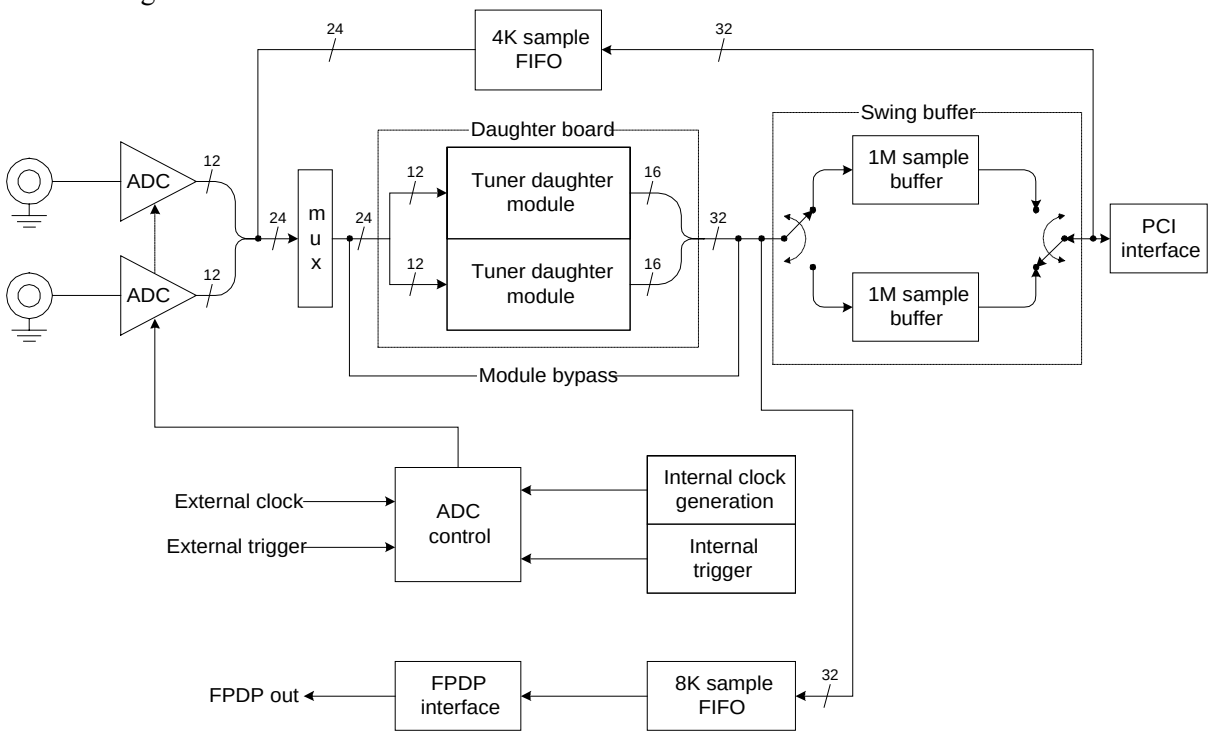

Figure 3-2 ICS650 block diagram

The figure shows that the ICS650 includes a 2 MegaSample "swing buffer" (2^{20} samples in each side of the swing buffer) for ADC data storage. The samples can be transferred from the dual-ported memories constituting the swing buffer to the host computer memory via the PCI interface and the PCI bus. The main advantage of this mechanism is that it is possible to read out one (side of the swing) buffer while the other is being filled with samples from the A/D converter. This allows for the high pulse repetition rates up to 4 kHz that are required for the application (see Appendix A). The data acquisition board of the current system only has one buffer and therefore lacks this advantage. If this buffer is filled, it must first be transferred to memory before the acquisition can continue. This is the reason it cannot satisfy the (new) specifications listed in Appendix A.
The samples of the two ADCs, which are clocked simultaneously, are combined to produce a 24-bit word which is placed either in the in the 8K sample FIFO and / or in the PCI swing buffer. This means that always two channels are recorded, even when only one channel is needed. On output, the data of the two channels is reorganized into 32-bit words in which pairs of 12-bit samples are located in the

most significant 12 bits of each 16-bit portion. Together with the fact that only one channel is required for the application, this implies that the amount of data that has to be transferred to the host memory is a factor $8/3$ times the memory occupied by the actual data of interest, which is an inevitable disadvantage.
The on-board buffers behave like FIFO type memory. In other words, random access to samples in memory is not available and data access is always sequential. Each time a data word is read from the buffer, the data is removed from memory and the buffer pointers are modified. A reset of the memory should be performed after programming the ADC configuration and before enabling acquisition. This is necessary to ensure that the buffer pointers are correctly aligned prior to buffer access by the ADC circuits.
The external trigger that is provided by the radar system must be a rising-edge TTL signal that must remain high for at least one clock cycle. The trigger is synchronized on the board to the sampling clock of the ICS650 and the conversion is initiated on the second rising edge of the sampling clock after the application of the trigger.
Figure 3-2 also shows that the outputs of the two ADCs may be passed to a digital demodulator (tuner) daughter board, if installed. Two different daughter board options are available: a module for wideband demodulation and a module for narrowband demodulation. The modules allow for the selection of any signal band for demodulation. However, no (further) demodulation is required for the SHIRA application, since the signal to be digitized has already been demodulated. Therefore the daughter board options are bypassed.
The ICS650 provides a number of options regarding operating modes, data buffering and data paths to and from the board. For the SHIRA application, only the capture mode without pre-trigger storage is relevant. In this mode, data is acquired for a programmable number of samples (N) following each application of a trigger. Because the size of the memory buffer (M) and the count of samples acquired per trigger are both programmable, multiple sweeps may be stored in the ICS650 memory before data is read from the board. When using this capture mode without pre-trigger storage, the programmed memory buffer length M must always be an integral number of acquisition counts. This is illustrated in Figure 3-3.

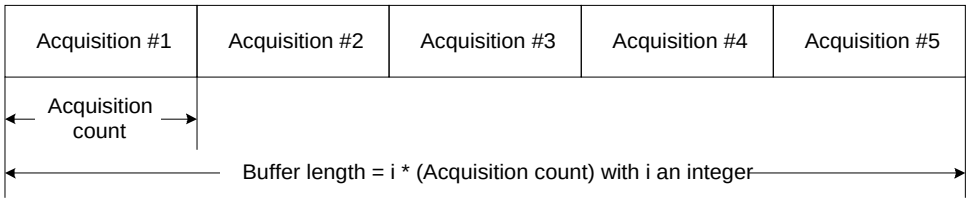

Figure 3-3 Capture mode without pre-trigger storage

If enabled, an interrupt (called ADC done interrupt) will occur after the number of samples acquired is equal to the programmed buffer length. In Section 3.3 a design decision will be made regarding the number of sweeps that is stored in one side of the swing buffer before the data is transferred to the host memory.
The ICS650 uses PCI interrupts to indicate swing buffer swap or A/D conversion completion, tuner module overflow, and DMA transfer completion. Two bits in the ICS650 Interrupt Mask register allow the user to enable the interrupts for swing buffer swap or A/D conversion completion and tuner module overflow. Corresponding bits in the status register indicate the status of these conditions. For the SHIRA application, the tuner overflow bit is disabled since no tuner module is present. *A PCI interrupt will occur when one of the bits in the interrupt mask register (IMR) is set, i.e. the interrupt is enabled, and the condition occurs.* The interrupt handler must then clear the corresponding IMR bit to prevent a further interrupt from occurring. If enabled, an ADC done interrupt is generated when the number of samples acquired per channel is equal to the value programmed in the buffer length register. A DMA read done interrupt is generated when a DMA read-out of a swing buffer has been completed; in addition, the pending ADC interrupt is cleared by the hardware at this moment.
In the used mode, i.e. the capture mode without pre-trigger storage, acquisition will stop when the number of samples acquired per channel is equal to the value programmed in the buffer length register. All data, i.e. the exact length of the data corresponding to the value programmed in the buffer length

register, must be read from the full side of the swing buffer before the ADC interrupts are re-enabled. If only a part of the data is read, immediately another ADC interrupt will occur, indicating that the read-side of the swing buffer is not empty.

3.2.3 The counter board

The requirements for the counter board, which is used to count the pulses from the angular encoder on the axis of the radar antenna, are not very demanding. The maximum frequency of the pulses to be counted can be computed as the quotient of the number of pulses from the angular encoder per revolution of the radar (N_{AE}) and the minimum rotation time of the radar antenna (T_R) (see Appendix A). Thus, it equals $N_{AE}/T_R = 4096/1.2 \approx 3413$ Hz. The PA1700-2 ISA counter board, made by Addi-Data in Germany, easily fulfills this requirement. It can count pulses up to 1.25 MHz [Addi-Data, 1999]. The three inputs of the counter board that are relevant for the SHIRA application are the *counter pulse* input, the *external strobe* input and the *index* input. The pulses to be counted from the angular encoder are fed into the pulse input. The counter board counts these pulses and stores the counter value in its count register, which is cleared upon the occurrence of a *North Reset* pulse, which is connected to the index input (the board is jumpered to clear the counter contents when the index input becomes active). The current counter register contents is latched into a data latch upon the occurrence of a trigger pulse on the external strobe input; in the meantime the counting operation continues to run in the background. As will be pointed out in Section 4.5.2, the driver for the data acquisition board will request this latched value from the counter board driver after each A/D conversion of a valid sweep (i.e. a sweep within the user-defined region).
The connection of the radar signals to the PA1700-2 counter board, and the interrelationship between the PA1700-2 and the other boards has been illustrated in Figure 3-1.

3.2.4 The time stamp board

The BC630AT board, made by Datum Inc. in the USA, is used for time stamping. It is an ISA-bus board designed to provide a precision real time clock, decode serial time code signals, digitally synchronize multiple PC's and provide a number of other valuable timing features; it also incorporates a battery backed real time clock IC which maintains settable time during power loss and can be synchronized to an external time code signal [Datum, 1996].
The two inputs of the time stamp board that are relevant for the SHIRA application are the *external time code* and the *external event capture* inputs. On the external time code input, a serial time code signal (for example from a satellite) can be applied. This signal can be decoded and used for time stamping. Also the real time clock IC and the internal crystal clock can be used to generate time stamps with microsecond precision. A trigger pulse on the external event capture input latches the current time into a set of latch registers. The board requires approximately 150 microseconds for transferring the time data to the data transfer address space in response to a time request firmware command. Compared to the minimum PRI (see Appendix A) of the data acquisition system, which is 250 microseconds (see Appendix A: PRF_{max} = 4000 Hz), this is a rather long time. Besides that, the BC630AT cannot be used for a PRF of 4 kHz anyway because the maximum event pulse frequency at the external event capture input is about 2 kHz. Therefore, the board can only be used for PRFs below 2 kHz. Above this PRF, software time stamping has to be used. At present, a PCI board named BC635/637PCI that can handle frequencies up to 4 MHz is available. However, when the data acquisition system was implemented, only the BC630AT board was available. The connection of the radar signals to the BC630AT and the interrelationship between the BC630AT and the other boards has been illustrated in Figure 3-1.

3.3 High-level design of the data acquisition system

Given the components described in the previous sections, a high-level scheme for the data acquisition system can be designed. High-level in this context means that the operation and timing of the system are described in general, qualitative terms. In order to start the design process, first the global operation of the data acquisition system is described. Next, the timing of the signals that are provided or generated is described.

After enabling the interrupts on the data acquisition board, the system is armed and the A/D conversion is started by the hardware upon the occurrence of an external trigger pulse. After the A/D conversion has completed, the board switches to the other side of the swing buffer (see below) and generates an ADC done IRQ (Interrupt Request). In the corresponding Interrupt Service Routine (ISR), many actions are carried out. The most important actions will be described here (also see Figure 3-4). More details can be found in the sections dealing with the drivers.
Firstly, the ISR disables the ADC interrupts in order to prevent another ADC interrupt, indicating that the read-side of the swing buffer is not empty, from occurring. Next, the ISR requests the time stamp and counter boards for their latched time stamp and counter values respectively (which belong to the just acquired sweep). As the counter value relates to the angular position of the radar it can be determined whether the sweep is in the region of interest. If this is true the DMA action, which transfers all samples from the full side of the swing buffer to the host memory via the PCI bus, is started. After the DMA is ready, a DMA done interrupt is generated by the hardware. In the ISR corresponding to this interrupt, the ADC interrupts are re-enabled and the system is armed waiting for the next trigger pulse. If the sweep is not in the region of interest, it is discarded.
Now, the timing of the actions that have to be carried out and the signals that occur in the system will be described in more detail. This information has been illustrated in Figure 3-4 as a timing diagram showing the chronological order in which the signals occur.

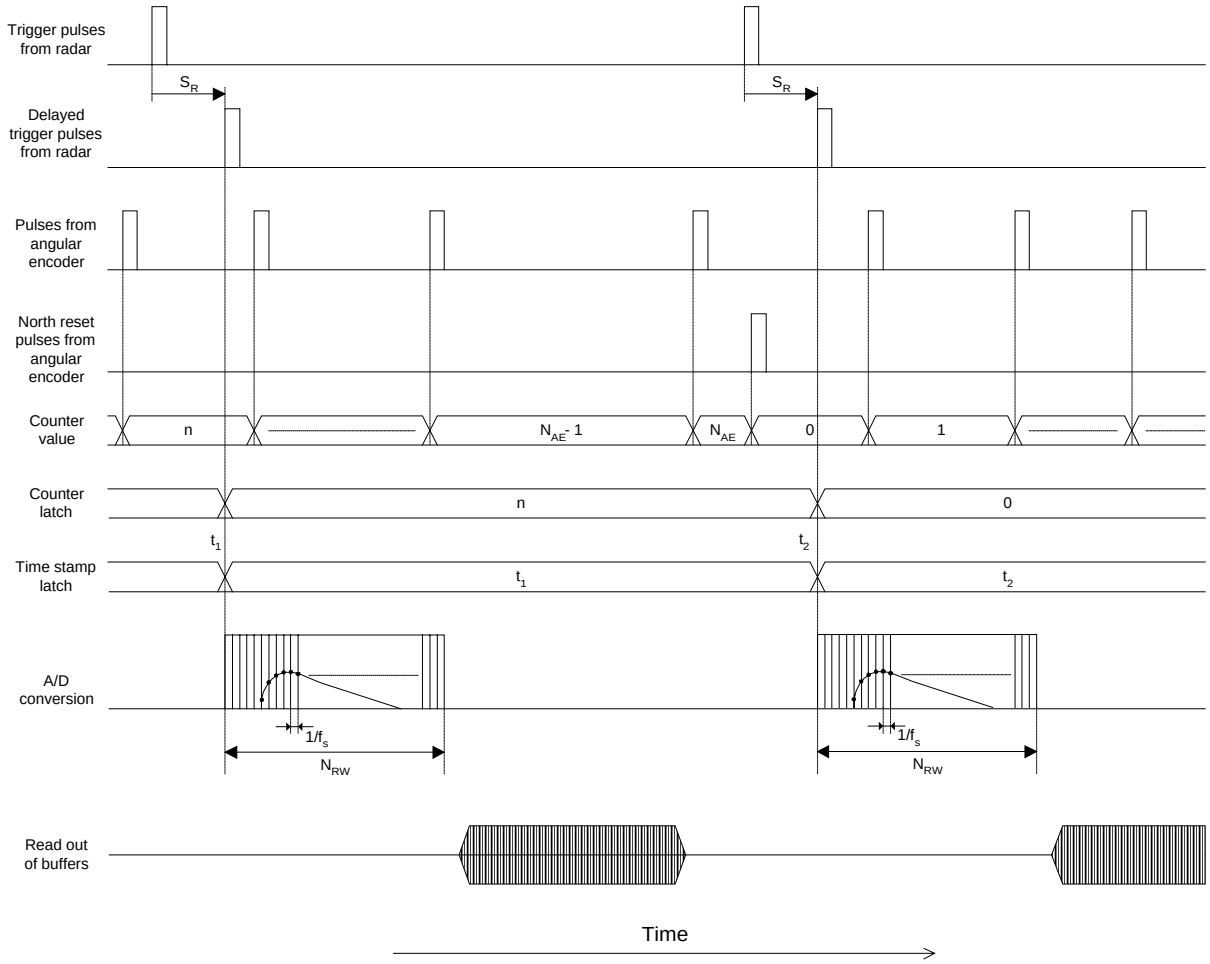

Figure 3-4 Data acquisition timing diagram

The first signal shows the trigger pulses from the radar fired at the moments the radar pulses are emitted, while the second gives the delayed pulses (used to realize the adjustable start range S_R). The hardware to create the trigger delay implementing the adjustable start range is not realized in this project. Therefore, the trigger pulses from the radar will directly be used to start an A/D conversion

process in the ICS650 data acquisition board. The delay can then be implemented later in software by discarding the samples that were taken at the ranges before the start range. Upon the occurrence of a (delayed) trigger pulse, the A/D conversion is started. At the end of the conversion, it is determined whether the corresponding sweep is valid (i.e. it lies between the start and end azimuth) and if it is, it is transferred to a circular host memory buffer. Figure 3-4 shows two valid sweeps, which are transferred to the host computer via the PCI bus by means of DMA. The third signal in the figure shows the pulses from the angular encoder, which have to be counted by the counter board and the fourth gives the angular encoder's *North Reset* signal, which is also fed into the counter board. The desired operation of the angular encoder and the counter board is illustrated by the next two signals. When a counter pulse arrives from the angular encoder, the counter board increments the value of its counting register. The relation between the counter value and the angular position of the radar antenna is determined by N_{AE}, being the total number of pulses generated per revolution. The value in this register is reset to zero upon the occurrence of a *North Reset* pulse and it is latched upon the occurrence of a (delayed) trigger pulse. This way, the pulses are counted relative to the *North Reset* reference point and therefore the angle with respect to this point is known.
The time stamps are also latched into the registers of the time stamp board upon the occurrence of the same trigger pulse. For example, it can be seen that at time t_1 a (delayed) trigger pulse is generated, which latches the counter and time stamp values at time t_1 (value n for the counter and value t_1 for the time stamp board). These latched counter and time stamp values must then be read from the latches before the next trigger pulse occurs, otherwise they will be overwritten by newer values resulting in incorrect angle and time stamp information. Therefore, the latched counter and time stamp values are requested by the data acquisition board driver from the counter board driver and time stamp board driver respectively, just after an A/D conversion has been completed and before the DMA is started. The latched counter value must be read before the DMA is started anyway because this value is needed in order to determine whether a sweep is valid or not.
The second signal from the bottom shows the A/D conversion process. When an A/D conversion has finished, the samples must be transferred to the host computer by means of DMA if the sweep is valid (bottom signal). In order to achieve the highest performance possible, the two dual-ported memories constituting the swing buffer have to be exploited fully. The main advantage of the double buffering mechanism is that it is possible to read out one buffer while the other is being filled with samples from the A/D converter. Virtually no time is lost transferring data into the host PC. As has been pointed out already, multiple sweeps may be stored in one side of the swing buffer on the ICS650 data acquisition board before data is read from the board. A decision has to be made regarding the number of sweeps that is stored in one side of the swing buffer before the data is transferred to the host memory. *The programmed buffer length M is chosen to be equal to the acquisition count N*. The main reason for this design decision is that it has to be possible to discard a sweep that is not in the recording section before it is transferred to the host memory in order not to waste memory space. If multiple sweeps were to be stored in the swing buffer and some of them were not valid (i.e. they were not in the recording section), they first would have to be transferred to the host memory before they could be discarded afterwards because it is impossible to discard a part of the buffer contents; only the contents of a whole buffer can be discarded. The reason for this is a peculiarity of the board that has been described in Section 3.2.2: the data buffers behave like sequential FIFO type memory and therefore, random access to samples in the memory is not available. Data access is always sequential regardless of the buffer address used for read or write. If the buffers are switched after each sweep, it is possible to decide for each sweep separately whether or not to discard it.
A second reason for gathering and storing data on a one-by-one sweep basis is that all data belonging to one sweep, i.e. the samples, the time stamp and the counter value, have to be stored in one logical contiguous block of memory or disk space, i.e. data from different streams have to be merged together. The data from other sources than the on-board memories of the data acquisition board (the time stamp, counter value and some additional information) are stored in a header. The memory is organized in the following way: it starts with the header data of a sweep. Then the sample values are stored, followed by the next sweep's header data, the next sweep's sample values, and so on. Since the sweeps of consecutive scans belonging to one measurement are stored contiguously in memory, the separate scans have to be extracted later on (during the signal processing). This can be done by detecting discontinuities in the angular information.

This section described the operation of the data acquisition system from a hardware point of view. Because all actions have to be controlled by software, also the software point of view is very important. This will be the subject of the next chapter.

4 Software implementation

4.1 Introduction

All the actions of the data acquisition system discussed in the above section have to be initiated and controlled by software. This software has to set up the memory, control the order in which the actions occur, initialize and control the hardware, etc. The software is composed of the so-called "kernel-mode" and "user-mode" software, as explained next. Under Windows NT, the operating system the software has to run on, the hardware can only be touched by "trusted system components", like device drivers. Device drivers (both the words "kernel-mode driver" and "driver" will be used in this text to refer to a device driver) will be used to manage the actual data transfer and control operations for the data acquisition components because they represent and control physical devices. Device drivers in Windows NT run in a privileged mode, called "kernel-mode", where they have complete access to all I/O devices and to the memory. User applications run in the so-called "user-mode" where they have restricted access to the different resources like I/O devices and memory. However, they can indirectly access I/O devices by calling a special kind of driver functions, called IOCTL (I/O control) functions or IOCTLs.
Since the overall goal is to acquire a time series of sweeps in a user-defined area and to store these sweeps into memory or onto a disk, it must be possible to set the acquisition parameters, such as the sweep length, the number of sweeps to acquire, the start and end azimuth, etc. from within a user-mode application. Therefore, an interface between the kernel-mode drivers, which control the acquisition hardware in order to execute the data acquisition, and the user-mode software must be created. This interface is composed of IOCTLs. The main architecture of the drivers and the user-mode software is dealt with in the following sections.

4.2 Overview of software structure

The main theme of this section will be the overall architecture of the device driver software, the user-mode software and their interaction.
In order to achieve a high level of modularity, three separate drivers are developed for the data acquisition device, the time stamp device and the counter device respectively (instead of integrating all the necessary driver code into one large driver). These drivers will have to work closely together and therefore they must expose well-defined interfaces through which other drivers can request services. The most important driver is the driver for the data acquisition board because this driver actually acquires the sweeps, and it requests services from the other two drivers (delivering time stamps and counter information, see Figure 4-1). Therefore, this driver will be called the "main driver".
The main driver is the only driver that will interface with the user-mode software. In order to achieve this interfacing, all necessary IOCTL functions that request services from the driver, are implemented, wrapped into convenient user-mode functions, and placed in a dynamic link library (DLL). User-mode applications or other DLLs can then request services from the driver by calling the functions from this DLL that perform low-level operations and therefore it is called "low-level DLL" in this report (see Figure 4-1). Because a lot of functions from this low-level DLL have to be called to perform one "high-level" action, for example to initialize, start and stop the device, another DLL is created, containing high-level functions, which in turn are composed of the low-level functions from the low-level DLL. This DLL is therefore called "high-level DLL". Figure 4-1 shows the relationship between the different software components. In order to make clear to which box in the above diagram a functions belongs (during the discussion of the software in this chapter), functions from the drivers will have the prefix "dr", functions from the low-level DLL "ll" and functions from the high-level DLL "hl".

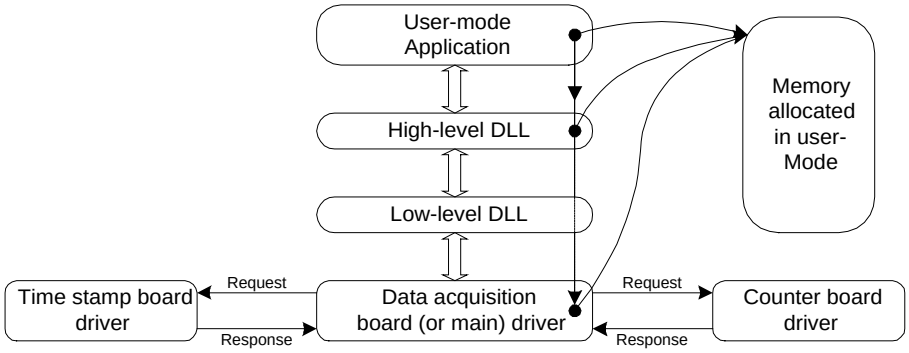

Figure 4-1 Relationship between software components

An important design decision is whether the memory buffer that will hold the DMA data and the additional data (the time stamps and the counter values) is allocated in the user-mode application, the high-level DLL, or in the main driver. One primary requisite is that the user-mode application must be able to access all data in the buffer because the user has to be able to process this data directly or store it to disk. Memory that is allocated within a driver (kernel-memory) is not available to a user-mode application without much effort; nevertheless it is possible to map kernel-memory into the user address space. However, this is not a safe solution because the user-mode application this way can directly touch "trusted" memory from the kernel. On the other hand, the memory allocated in a user-mode program can be passed on very simply to a driver, but not vice-versa. Therefore, it can be concluded that allocation of the data buffer in a user-mode program is preferred over allocation in a driver. Thus, the memory will be allocated in the user-mode program as has been illustrated in Figure 4-1. Pointers to this memory area will then be passed on to the driver by means of an IOCTL function from the low-level DLL.
The main architecture of the different software components and the interactions between them will be described in the following sections.

4.3 Software framework: the buffering mechanism

The framework for the data acquisition system software is constituted by the *buffering mechanism*. This mechanism describes the operation of the data acquisition system from a software point of view and is illustrated in Figure 4-2. In order to arrange for continuous acquisition, *the high-level DLL partitions the circular data buffer in the host memory into two equal parts called buffer pools* (see figure), each of which provides storage for an integer number of sweeps. The idea is as follows: the driver transfers the sweeps that are acquired into both sides of the on-board swing buffer alternately to the circular data buffer in the host memory. When the driver has filled one of the buffer pools with a fixed number of sweeps, it generates an event to the high-level DLL, decrements its own internal buffer pool count, and continues acquiring sweeps into the other buffer pool. When the high-level DLL receives such an event generated by the driver, it stores the data to disk if desired, it decrements its local buffer pool count, and then starts waiting for the next event to handle the data of the other buffer pool. From this it easily follows why the host memory has to be divided into two parts (buffer pools): while one half is filled with sweeps, the other is written to disk (if desired).
When the last sweep of a buffer pool has been acquired, the previous buffer pool should already have been read out if the data is stored to disk, otherwise wrong data will be written to disk. If data is only acquired into memory, earlier acquired data is overwritten and lost. In this case, it only makes sense to make the data buffer large enough to contain all data of a measurement.
When the buffer pool count reaches zero, both the driver and the DLL know that the acquisition is complete. The acquisition can also be terminated by the user at any time by means of the IOCTL function drICS650IoctlStopMultSweepAcq. In order to do this, a user-mode software component must call its wrapper llics650StoptMultipleSweepAcq.
The buffer pool count required for an acquisition can be calculated from the number of sweeps provided by the user-mode application and this number is passed on to the driver via the DLLs. If the

buffer length is L sweeps (see Appendix A), the buffer pool length is $L/2$. With N_S (see Appendix A) being the number of sweeps the user wants to acquire, the buffer pool count can easily be calculated as $\lceil 2N_s/L \rceil$. The buffer pool count can only be an integer number, since the driver only generates events to the high-level DLL when one of the two available buffer pools has completely been filled. Both the application and the driver have to keep track of the buffer pool count in order to be able to determine when an acquisition is complete. Moreover, they both have to know the places in memory where the sweeps are (to be) stored and which buffer pool is currently being filled. Both the data acquisition board driver and the high-level DLL use similar mechanisms like linked lists as data structures for storing all this information. Because they need to store different kinds of information, they use separate linked lists. Each element of the driver's linked list belongs to one sweep, and each element of the high-level DLL's linked list belongs to one buffer pool. Therefore, the number of elements in the driver's linked list equals the number of sweeps that fit in the host memory data buffer and the number of elements in the high-level DLL's linked list is two because there are two buffer pools. This is also illustrated in Figure 4-2, which is exemplified in detail in sections 4.5.2.3 and 4.4.3.2. Here, the first buffer pool is being filled with sweeps by the driver, and the high-level DLL is waiting for the event from the driver indicating that the first buffer pool is full. The pointer to the current buffer pool entry (i.e. the element of the high-level DLL's linked list) is updated after the data from the buffer pool have been written to disk (if acquiring to disk at all).

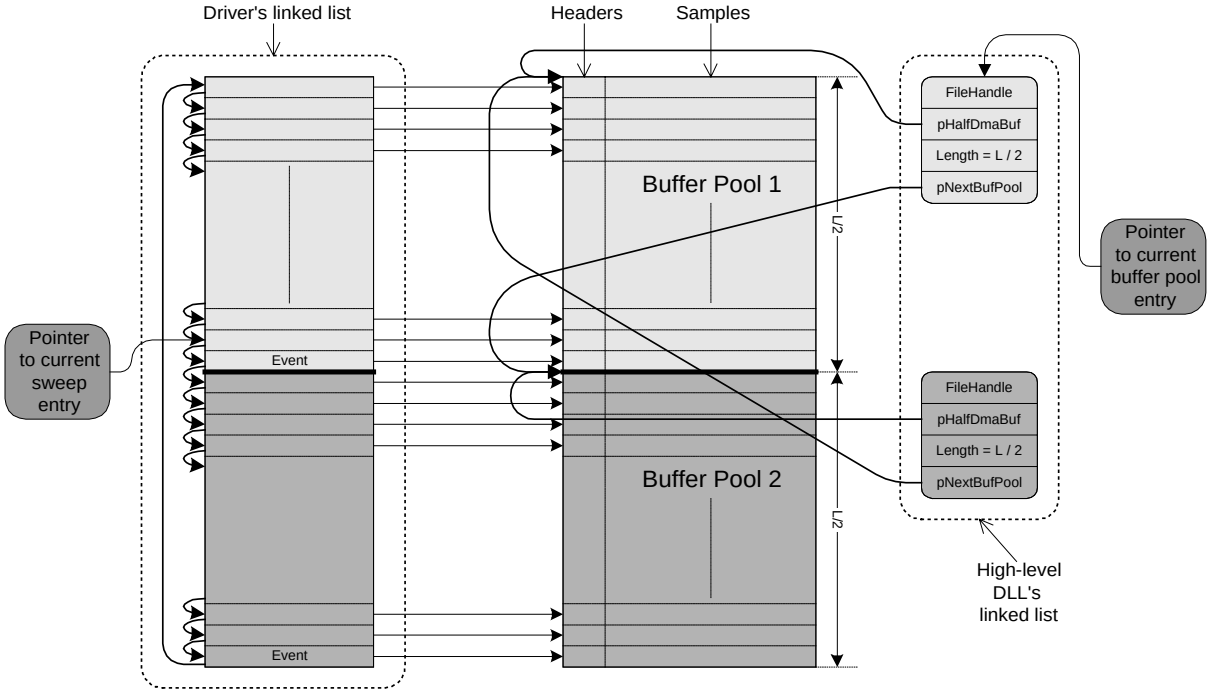

Figure 4-2 Buffering mechanism

The next two sections describe the driver's linked list and the DLL's linked list respectively in more detail. After that, the different drivers, the DLL and the user-mode application are discussed.

4.4 The user-mode software

4.4.1 Introduction

The user-mode software provides the data acquisition parameters and other information to the driver(s) and implements the user-mode part of the buffering mechanism. The user-mode software consists of three components (see Figure 4-1): a user-mode application, the high-level DLL and the low-level DLL. The functionality of the user-mode system has been spread over these levels (components) and the combination them realizes the desired functionality. The user-mode application is merely a graphical user interface (GUI) consisting of a dialog box, which enables the user to enter the

acquisition parameters and from which memory is allocated. Figure 4-3 shows a screenshot of the GUI.

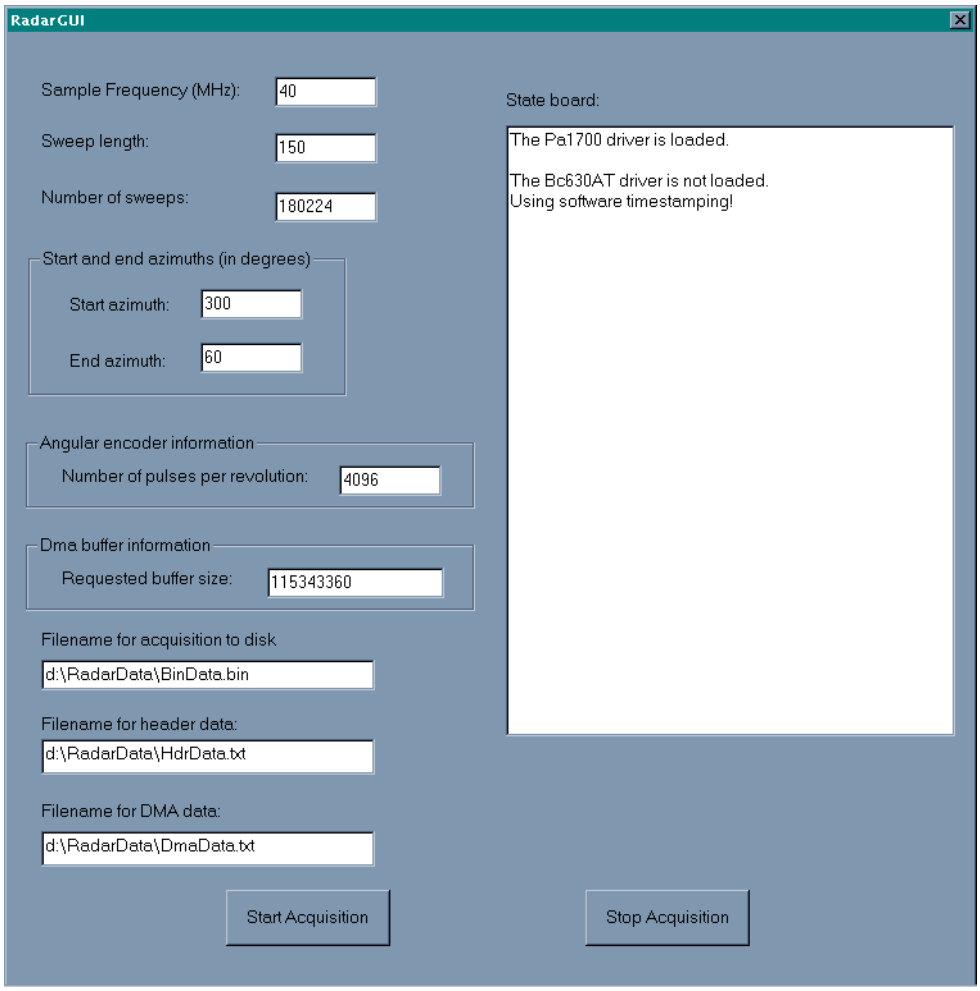

Figure 4-3 Screen shot of the test GUI

The active user-mode part of the buffering mechanism (and hence the acquisition) is handled by the high-level DLL. Therefore, the information regarding the parameters and the memory is passed from the user-mode application to the high-level DLL, which partitions the memory, provides the driver with the information it needs by calling functions from the low-level DLL, sets up the DLL's linked list and waiting loop, etc. As has been pointed out in Section 4.2, the low-level DLL contains convenient user-mode functions calling IOCTL functions in the driver of the data acquisition board. The next sections provide a more detailed explanation of the operation and software structure of the different software modules.

4.4.2 The user-mode application

The user-mode application is a test GUI that is required for testing the system and it is written in MSVisual C++ 6.0. The dialog box that forms the GUI of the user-mode application provides a number of edit boxes in which values for the acquisition parameters can be entered. In addition, the dialog box contains two buttons, one for actually starting the acquisition ("Start Acquisition") and another for stopping the acquisition prematurely ("Stop Acquisition"). The most important parameters that have to be entered by the user are: the sampling frequency in MHz, the number of samples per sweep (the sweep length N_{RW}), the number of sweeps that has to be acquired, the start azimuth (S_A) and end azimuth (E_A) in degrees, the number of pulses from the angular encoder per revolution of the radar (N_{AE},), and the size of the memory buffer in bytes (see Figure 2-1).
When the application is launched, the initialization method `CRadarGUIDlg::OnInitDialog` is executed. This

method first passes a pointer to a structure that will contain the runtime (acquisition) parameters to the high-level DLL in order to give it access to this data by calling the function hlSetPtrToRuntimeParams. Next, the event that will be signaled by the high-level DLL when the acquisition is complete, is created. The handle to this event is duplicated by the high-level DLL by a call to the function hlSetReadyEvent, so as to enable it to signal the event of the user-mode application. The duplication of the handle of the event is necessary because the same event is shared by different processes. Then, the application requests the high-level DLL to supply information about the counter board (driver) and the time stamp board (driver) by calling hlQueryDriverInfo. The high-level DLL in turn, calls the function llics650RequestInformation from the low-level DLL, which actually requests the information from the driver by means of an IOCTL. If the counter board driver is not loaded, the user-mode application (and the high-level DLL) is informed that no angular information is available and that sweeps will be acquired consecutively for one fixed angular position of the radar. If the time stamp board driver is not loaded, the user-mode application has no access to hardware time stamps and has to use software time stamps instead. After this, the initialization is complete and a measurement can be started. When the acquisition parameters have been entered by the user, pushing the "Start Acquisition" button causes the function CRadarGUIDlg::OnStartAcqBtn to be called. In this function, the following actions happen (only the most important ones are explained).
Firstly, the start and the end sweep numbers (i.e. the counter values corresponding to a certain sweep, also called sweep ID's) of a recording window are computed from the start and end azimuths respectively in the following way: the start sweep ID equals $(S_A \bmod 360) \cdot N_{AE} / 360$ and the end sweep ID equals $(S_E \bmod 360) \cdot N_{AE} / 360$. If the end azimuth is 360 degrees, the user obviously wants to acquire sweeps up to the *North Reset* position and therefore the end sweep ID is set to a very high number, which is much higher than the number of sweeps in one revolution of the antenna, in order to make all sweeps between the start sweep and the *North Reset* valid.
Next, the actual buffer size is calculated from the requested buffer size in such a way that it contains an even number of sweeps. The following piece of code shows how this is done:

```
ulTotSweepData = m_SweepLength * sizeof(int) + sizeof(SWEEPHEADER);

// Calculate the size of the buffer so that it contains an integer number of sweeps
ulDmaBufSize = ulDmaBufSize - ((ulDmaBufSize % ulTotSweepData));

// If the number of sweeps is odd, substract one sweep (result: even number of sweeps)
if ( (ulDmaBufSize / ulTotSweepData) % 2 )
{
        ulDmaBufSize -= ulTotSweepData;
}
```

The remainder of the integer division of the requested buffer size and the total amount of data per sweep (in bytes) is subtracted from the requested buffer size, in order to obtain a buffer size that contains an integer number of sweeps. Then, if the number of sweeps that fit into the buffer is odd, one more sweep is subtracted, resulting in an actual buffer size that equals the amount of data of an even number of sweeps. After the computation of the actual buffer size, the memory for the buffer is allocated. Next, the buffer pool count is calculated as $\lceil 2N_S / L \rceil$ (see Section 4.3).
After that, the high-level DLL function hlInitializeIcs650AndSetRecParams that initializes and configures the data acquisition board with the parameters specified by the user (see next section) is called. When the data acquisition board has been initialized and configured successfully, the function hlStartAcq from the high-level DLL that actually starts the acquisition is called. Just after this function has returned successfully, the user-mode application starts a thread, called WaitThread (it is also possible to start the thread before the acquisition is started). This thread waits for the event indicating that the acquisition is complete, to be signaled by either the high-level DLL or the user-mode application itself (in case the "Stop Acquisition" button has been pushed). Then, in case the acquisition has completed normally, the data that is currently in the memory buffer is saved to disk for testing purposes (due to the fact that during the final tests no fast disk system was available, the sweeps were first acquired into memory and afterwards written to a (slow) IDE disk, see Chapter 5). The saved data on the disk allow for inspection on correctness and completeness (no loss of data, etc.).

If, on the contrary, the “Stop Acquisition” button was pushed, the acquired data is discarded.

4.4.3 The high-level DLL

4.4.3.1 Overview

The high-level DLL is written in C and it is not a *passive* dynamic link library in the sense that it only contains a set (library) of routines that can be called by programs at run time.
It also handles the *active* user-mode part of the buffering mechanism. This section explains the most important routines of the DLL and its interaction with the driver via the buffering mechanism in the order prescribed by the user-mode application and the chronological order in which actions are taken (see Section 3.3). Only the functionality that requires further explanation in addition to the explanation given in the previous section is elucidated.
As stated in Section 4.4.2, after computing the start and end sweeps, the buffer size, etc., the function CRadarGUIDlg::OnStartAcqBtn from the user-mode application calls the high-level DLL function hlInitializeIcs650AndSetRecParams that initializes and configures the data acquisition board with the parameters specified by the user. This function in turn calls functions from the low-level DLL to perform the requested operations. First, the board is reset by calling the function llics650BoardReset. Next, the board is configured, i.e. the trigger mode, the ADC clock mode, the capture mode, the sampling frequency, the acquisition count, the buffer length, etc. are programmed, by calling the functions llics650ConfigSet, llics650ADCClkSet, llics650AcquireCountSet, llics650BufferLengthSet and llics650DecimationSet. A description of all the IOCTL driver functions, provided by the nominal driver (see Section 4.5.2.1), that are wrapped by these low-level DLL functions can be found in [ICS650 Software Developers Kit User’s Manual].
The last function that is called by hlInitializeIcs650AndSetRecParams is llics650SetRecordingParams. This function supplies the runtime acquisition parameters (i.e. the memory pointers, the sweep length, the start and end azimuth sweeps, the number of buffer pools, etc.) to the driver, where they are stored in a driver dedicated memory area, called device extension, by means of the IOCTL function drICS650IoctlSetRecParams. As has been pointed out in Section 4.4.2, after hlInitializeIcs650AndSetRecParams has returned, the function hlStartAcq is called by the user-mode application. This function first initializes the high-level DLL’s linked list, i.e. it partitions the host memory buffer into two parts, initializes the pointers, etc. The next section describes the linked list in detail. After the initialization of the linked list, hlStartAcq starts a thread, named AcqThread, which handles the active part of the buffering mechanism, i.e. it starts the actual acquisition, runs through the waiting loop and finalizes the acquisition. This thread is described in Section 4.4.3.3. The pseudo code of hlStartAcq makes clear the context of the main themes of the next two sections:

```
hlStartAcq
{
        Initialize linked list
        Set current buffer pool to the first one in the list

        Create AcqThread
        Kick off AcqThread
}
```

4.4.3.2 The high-level DLL’s linked list

From Section 4.3 it follows that the high-level DLL has to know which of the buffer pools is being filled, where they are stored and which one can be saved to disk, etc. It must keep track of the buffer pool information in order to *synchronize with the driver operation and to determine when a measurement is complete*. Like the driver, it uses a linked list for this purpose. The right side of Figure 4-2 illustrates the organization of the high-level DLL’s linked list. The (C-language) definition of a linked list element is shown below:

```
typedef struct _Dma_Buffer_Pool_Entry
{
        HANDLE  hFile;                  // Handle to file if data must be stored to disk
        PUCHAR pHalfDmaBuf;             // Pointer to buffer pool
        ULONG   HalfDmaBufLength;       // Buffer pool length
```

```
        struct _Dma_Buffer_Pool_Entry *pNextDmaBufPoolEntry;     // Pointer to next buffer pool entry

} DMA_BUFFER_POOL_ENTRY, *PDMA_BUFFER_POOL_ENTRY;
```

The linked list elements are called *"DMA buffer pool entries"*. Each DMA buffer pool entry contains the information for one buffer pool and consists of four items. The handle called hFile is necessary to identify the file into which the data will be stored when acquiring to disk (if the data is acquired to memory only, this handle is not used). The DLL keeps track of the place in memory where the (current) buffer pool is stored by means of a pointer called pHalfDmaBuf, which points to the beginning of the buffer pool. The size of the buffer pool is stored in the variable HalfDmaBufLength. Finally, a pointer called pNextDmaBufPoolEntry to the next buffer pool entry is used to point to the next element of the linked list. Since there are only two buffer pools, the linked list contains two elements that point to each other. As stated in the previous section, the creation of the linked list and the initialization of its elements is done in the high-level DLL function hlStartAcq. The next piece of code from this function shows the initialization of the linked list in case the sweeps are acquired to memory only:

```
// Now get the values of pDmaBuffer and ulDmaBufSize
pDmaBuffer = pRunTimeParams->DmaBufData.pDmaBuf;
ulDmaBufSize = pRunTimeParams->DmaBufData.ulDmaBufSize;

// Initialize the Dma Buffer pool (the Dma Buffer pool is implemented as a linked list)
for(i = 0;i < DMA_BUF_POOLS; i++)
{
        Dma_Buffer_Pool[i].hFile = INVALID_HANDLE_VALUE;
        Dma_Buffer_Pool[i].pHalfDmaBuf= (PUCHAR) (pDmaBuffer + i * ulDmaBufSize / DMA_BUF_POOLS);
        Dma_Buffer_Pool[i].HalfDmaBufLength = ulDmaBufSize / DMA_BUF_POOLS;
        Dma_Buffer_Pool[i].pNextDmaBufPoolEntry = &Dma_Buffer_Pool[i+1];
}
Dma_Buffer_Pool[DMA_BUF_POOLS - 1].pNextDmaBufPoolEntry = &Dma_Buffer_Pool[0];
pCurrentDmaBuffer = &Dma_Buffer_Pool[0];
```

The code in the for-loop is executed twice because the number of buffer pools (DMA_BUF_POOLS) is 2. The hFile fields of the pools are set to a valid file handle when acquiring data to disk and to the value INVALID_HANDLE_VALUE when acquiring to memory only. Before entering the for-loop, the pointer pDmaBuffer is set to point to the beginning of the memory that has been allocated by the user and the variable ulDmaBufSize equals the total size of the allocated memory. After the for-loop has executed, the two pHalfDmaBuf pointers of the DMA buffer pool entries each point to the beginning of a buffer pool (see Figure 4-2). The variables named HalfDmaBufLength contain the size of a buffer pool (i.e. *L/2* in the figure). The pNextDmaBufPoolEntry pointers each point to the other buffer pool entry to indicate their next linked list element. The line of code just after the for-loop ensures that the last (i.e. second) element points back to the first one. Finally, the last line of code sets the current sweep buffer entry to be the first one in the linked list:

```
pCurrentDmaBuffer = &Dma_Buffer_Pool[0];
```

This ensures that when the acquisition starts, the DLL is waiting for the event signaling that the first buffer pool has been filled with sweeps.

4.4.3.3 The high-level DLL's thread

After the initialization of the linked list, hlStartAcq starts the thread AcqThread. This thread handles the active part of the buffering mechanism, i.e. it starts the actual acquisition, runs through the loop associated with the linked list during the measurement, and finalizes the acquisition. Since this thread is the heart of the user-mode part of the buffering mechanism during a measurement, the most important portions of its code are listed here:

```
BOOL DisplayThread(LPVOID ThreadParams)
{
        Declaration and initialization of variables …

        // Start multiple sweep acquisition ...
        if( !llics650StartMultipleSweepAcq (hIcs650Dev, &pRunTimeParams->DmaBufData) ){
                DllPrint(__LINE__, "ics650StartMultipleSweepAcq failed!\n");
                return false;
```

```
        }
        do
        {
                dwWaitResult =  WaitForMultipleObjects(
                        2,                      // number of handles in the handle array
                        HandleArr,              // pointer to the object-handle array
                        FALSE,                  // wait flag
                        INFINITE                // time-out interval in milliseconds
                        );
                Save to disk if desired …
                pRunTimeParams->RecParams.RecCount--;
                pCurrentDmaBuffer = pCurrentDmaBuffer->pNextDmaBufPoolEntry;
        }
        while ( (dwWaitResult != WAIT_TIMEOUT) && (pRunTimeParams->RecParams.RecCount != 0)
                        && (!bKillDisplayThread) );

        // Stop multiple sweep acquisition ...
        if (!Ilics650StoptMultipleSweepAcq (hIcs650Dev, &pRunTimeParams->OutBufData)){
                DllPrint(__LINE__, "ics650StoptMultipleSweepAcq failed!\n");
        }else {bRecording = FALSE;}

        Test if acquisition was cancelled by user. If yes, set flag …

        SetEvent(hRdyEvnt);                     // Signal application that acquisition is complete
        return true;
}
```

First, the low-level DLL wrapper Ilics650StartMultipleSweepAcq calls the IOCTL function drICS650IoctlStartMultSweepAcq in the driver, which actually starts the acquisition. After this driver function has returned, the acquisition is started and valid sweeps that are acquired into the on-board memories are transferred to first buffer pool in the host memory by means of DMA. AcqThread then starts the loop that waits for and handles the events from the driver indicating that a buffer pool has been filled with sweeps. It waits for either the event from the driver or the event from the user indicating that the acquisition must be cancelled, by calling the Windows API function WaitForMultipleObjects, which returns after one of these events has been signaled. As has been described in Section 4.3, after the reception of a "buffer pool full" event from the driver, AcqThread saves the data to disk if desired. Next, it decrements the high-level DLL's local buffer pool count. After that, it updates the pointer to the current buffer pool entry as illustrated at the right side of Figure 4-2. The final part of the loop concerns the tests for the continuation of the loop. Three conditions can force the loop to break. If none of these conditions is satisfied, the loop again starts waiting for one of the two possible events. If one of the three conditions is satisfied, the loop is exited and the acquisition is stopped (see further). Firstly, if a time-out has occurred, something is wrong, and consequently the loop is exited. Secondly, in case the flag bKillDisplayThread is set to true (this happens when the user presses the "Stop Acquisition" button), the loop is exited as well. Thirdly, when the buffer pool count reaches zero, the DLL knows that the measurement has completed normally and the loop can be stopped. After the termination of the loop, the low-level DLL wrapper Ilics650StoptMultipleSweepAcq calls the IOCTL function drICS650IoctlStopMultSweepAcq in the driver, which actually stops the acquisition. It stops the data acquisition system in the correct way and performs some de-initializations. After AcqThread also has handled some de-initialization details, it signals the event the thread WaitThread of the user-mode application is waiting for.

4.5 The drivers

4.5.1 Overview

Because it is not the purpose of this report to explain Windows NT device drivers, only their functional behavior that is important for the SHIRA application will be described here and not the driver-specific details. The programming concepts for developing Windows NT device drivers can be found in [Baker, 1997] and [Dekker and Newcomer, 1999]. As has been pointed out, the sweeps are transferred to the host memory by means of DMA. In order to use DMA in drivers, a lot of knowledge about DMA under Windows NT and about the Windows NT memory system is required. Again, since it is not the purpose of this report to explain these concepts extensively, they will not be dealt with in

this report. In addition, a lot of specific driver concepts have been used when programming the drivers. For example, the driver for the data acquisition board uses auto-detection to detect the PCI board, while the drivers for the time stamp board and the counter board have to report the hardware resources they want to use themselves, since they represent and control ISA devices. Furthermore, all drivers have logging mechanisms built in to report status information to the user. The next sections will emphasize the functionality of the drivers that is specific to the SHIRA application. The flow of operations a driver is supposed to perform is described and visualized in flow graphs. First, the driver for the data acquisition board (the main driver) is described, since this is the most important one. Next, the drivers for the time stamp board and the counter board are described.

4.5.2 The data acquisition board driver (main driver)

4.5.2.1 Introduction

From the previous sections (especially sections 3.3 and 4.3), the main structure for the data acquisition driver can be deduced. The design of the driver for the ICS650 data acquisition board starts with a nominal driver provided by the manufacturer of the ICS650 board, which supports only the standard data acquisition features. The main reason that the nominal driver is not suitable for the SHIRA application is the fact that only one sweep at a time can be captured. Therefore, if the current functionality also has to be preserved, code that allows for multiple sweep acquisition has to be added to the driver. The wrappers of the nominal IOCTL functions, which in fact form the API of the driver, are also contained in the low-level DLL. An extensive description of the API-functions provided by the manufacturer can be found in [ICS650 Software Developers Kit User's Manual]. The most important and time consuming part of the implementation of the data acquisition system is the modification of the data acquisition board driver in order to make it suitable for multiple sweep acquisition. Four IOCTL functions are added in order to enable the user-mode software to interface with the driver. The usage of the low-level DLL wrappers of these functions (drICS650IoctlStartMultSweepAcq, drICS650IoctlRequestInfo, drICS650IoctlSetRecParams and drICS650IoctlStopMultSweepAcq) has been described in the sections dealing with the user-mode software. So, at this point it is known what tasks the IOCTLs perform and in which context they are used. The most important parts of their implementation will be discussed. Since drICS650IoctlRequestInfo and drICS650IoctlSetRecParams are simple functions that request and pass information or parameters respectively, they are not explained. In addition to the IOCTLs, a lot of other internal driver functions (not exported) have been modified or added. As stated earlier, the main driver requests services from the counter and time stamp board drivers. In order to directly call routines from the other drivers, the main driver gets pointers to these drivers during its initialization.
The next sections explain the most important added or modified driver software and the interaction of the driver with the high-level DLL via the buffering mechanism in the order prescribed by the user-mode application, the high-level DLL and the chronological order in which actions are taken (see Section 3.3). Only the functionality that requires further explanation in addition to the explanation given in previous sections is elucidated. Because some parts of the code are rather complex, the corresponding flow graphs are sometimes divided into smaller parts, each of which is depicted in a different figure. First, the initialization and start of a measurement by means of the driver IOCTL function drICS650IoctlStartMultSweepAcq is explained. Then, the driver's linked list is dealt with. After that, the heart of the driver that enables multiple sweep acquisitions, namely the interrupt service routine (ISR), is discussed and finally the termination of a measurement by means of the function drICS650IoctlStopMultSweepAcq is elucidated.

4.5.2.2 Initialization and start of a measurement

When the data acquisition board has been initialized and configured successfully, and the acquisition parameters have been provided to the driver (boxes 1 and 2 in the flow graph of Figure 4-4) as described in Section 4.4.3.1, the function hlStartAcq from the high-level DLL calls drICS650IoctlStartMultSweepAcq which actually starts the acquisition. This function first performs a number of initializations and then starts the measurement. These actions are denoted by the numbers 3 through 6 in the flow graph of Figure 4-4.

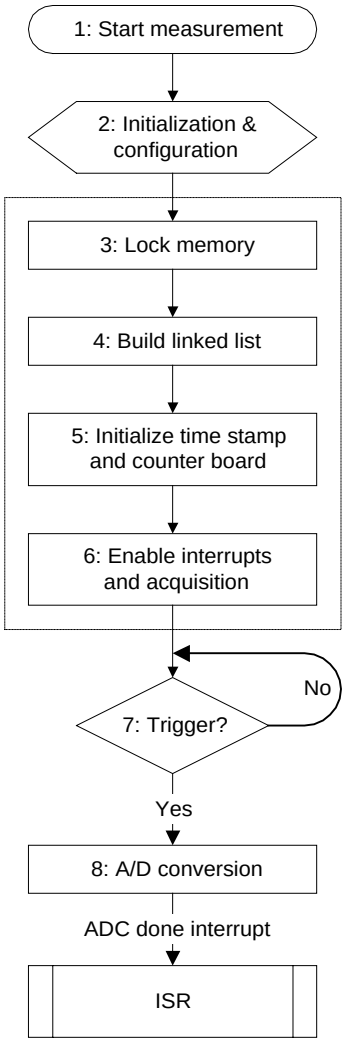

Figure 4-4 Flow graph of start of a measurement. The part surrounded by the dashed box indicates the actions of drICS650IoctlStartMultSweepAcq

First, drICS650IoctlStartMultSweepAcq locks the memory pages of the host buffer into memory (box 3), i.e. if pages of the allocated memory are still paged out on disk, they are transferred to the physical memory and the paging system is informed that these pages may not be removed from memory until they are unlocked. After the memory pages have been locked, the driver's linked list is built as described in the next section (box 4). Then, drICS650IoctlStartMultSweepAcq tests whether the time stamp and counter board drivers were loaded successfully during the data acquisition board driver's initialization (i.e. when the host computer is booted or before a measurement is started). If true, they are requested to initialize themselves (box 5). Next, the interrupts of the data acquisition board are enabled and finally the acquisition is enabled, i.e. the board is armed (box 6). After box 6 in Figure 4-4, drICS650IoctlStartMultSweepAcq returns. After drICS650IoctlStartMultSweepAcq has enabled the acquisition, the board starts the analogue-to-digital conversion upon the occurrence of a trigger pulse (boxes 7 and 8). When this conversion is completed, a PCI interrupt is generated to the host computer indicating that a new sweep has been acquired into one side of the swing buffer and that this data must be read out (ADC done interrupt). At this point, the board has switched to the other buffer because the programmed buffer length is equal to the sweep length. When the PCI interrupt is asserted, the driver's interrupt handler is executed. This is described in Section 4.5.2.4. After a measurement has been completed, the system must be de-initialized in the correct way. Section 4.5.2.7 describes how this is done.

4.5.2.3 The driver's linked list

From Section 4.3 it follows that the driver has to know where the data belonging to the sweeps has to be stored in the host memory buffer. Moreover, it has to know when to generate an event to the high-level DLL, i.e. when a buffer pool has been filled. This information is required in order to *synchronize with the operation of the high-level DLL and to determine when a measurement is complete*. Like the driver, it uses a linked list for this purpose. The left side of Figure 4-2 illustrates the organization of the driver's linked list. Like the high-level DLL, the driver keeps track of the sweep information in a linked list. The (C-language) definition of an element of the linked list used by the driver is shown below:

```
typedef struct _sweep_buf_entry
{
        PUCHAR pSweepHeaderSystemVA;    // Sweep header system virtual address
        PUCHAR pSweepDataVA;            // Transfer virtual address for DMA
        PKEVENT pEvent;                 // Event for signalling Win32 application
        struct _sweep_buf_entry *pNext; // Pointer to next sweep buffer entry

} SWEEP_BUF_ENTRY, *PSWEEP_BUF_ENTRY;
```

Each list element, called a *"sweep buffer entry"*, contains the information for one sweep and consists of four items: a pointer called pSweepHeaderSystemVA to the memory location in the host memory buffer where the header data (time stamp, counter and additional information) of the sweep is stored; a pointer called pSweepDataVA to the memory location where the incoming samples are to be stored, a pointer called pEvent to the event that will signal the high-level DLL when a buffer pool has been filled, and finally a pointer called pNext to the next sweep buffer entry. This pointer is used to update the current sweep buffer entry after a sweep has been stored in the host memory buffer. From the information in this next sweep buffer entry (linked list element), the driver knows where to store the data belonging to the next sweep.
As stated in the previous section, the creation of the linked list and the initialization of its elements is done in the driver function drICS650IoctlStartMultSweepAcq. The next piece of code from this function shows the initialization of the linked list:

```
……
// Get system virtual address of header data
pSweepHeaderSystemVA = (PUCHAR) MmGetSystemAddressForMdl (pDeviceExt->pMdl);

// Get virtual address of first sweep data
pSweepDataVA = (PUCHAR)MmGetMdlVirtualAddress(pDeviceExt->pMdl) + sizeof(SWEEPHEADER);
……
// Initialize first half of DMA buf sweep entry linked list
for(i = 0; i < ulSweepsInDmaBuf / 2; i++)
{
        pSwBufLinkList[i].pSweepHeaderSystemVA = pSweepHeaderSystemVA + i * TotDataPerSweep;
        pSwBufLinkList[i].pSweepDataVA = pSweepDataVA + i * TotDataPerSweep;
        pSwBufLinkList[i].Event = NULL;
        pSwBufLinkList[i].pNext = &pSwBufLinkList[i + 1];
}
// Signal when first buffer pool has been filled
pSwBufLinkList[ ulSweepsInDmaBuf / 2 – 1 ].Event = pDeviceExt->pEvent;

// Initialize second half of DMA buf sweep entry linked list
for(i = ulSweepsInDmaBuf / 2; i < ulSweepsInDmaBuf; i++)
{
        pSwBufLinkList[i].pSweepHeaderSystemVA = pSweepHeaderSystemVA + i * TotDataPerSweep;
        pSwBufLinkList[i].pSweepDataVA = pSweepDataVA + i * TotDataPerSweep;
        pSwBufLinkList[i].Event = NULL;
        pSwBufLinkList[i].pNext = &pSwBufLinkList[i + 1];
}
// Signal when second half has been filled
pSwBufLinkList [ ulSweepsInDmaBuf – 1 ].Event = pDeviceExt->pEvent;

// Last entry points back to first:
pSwBufLinkList[ ulSweepsInDmaBuf – 1 ].pNext = &pSwBufLinkList[0];

// Set current buffer entry
pDeviceExt->pCurrentSweepEntry = &pSwBufLinkList[0];
```

Before the first for-loop is entered, the pointer pSweepHeaderSystemVA is set to point to the first header in the buffer and the pointer pSweepDataVA is set to point to the first sample value of the first sweep in the buffer. In the first for-loop, the first half of the linked list that refers to the first buffer pool is initialized. As can be seen in Figure 4-2, the memory is organized in the following way: it starts with the header data of a sweep containing the time stamp, counter value and some additional information. Then the sample values are stored, followed by the next sweep's header data, the next sweep's sample values, and so on. The variable TotDataPerSweep equals the size of the total amount of data in one sweep, i.e. the size of the header plus the size of the samples. After each iteration through the loop, pSweepHeaderSystemVA points to the header data of the next sweep, and pSweepDataVA points to the sample data of the next sweep. During the acquisition, the driver needs the pointer values pSweepHeaderSystemVA and pSweepDataVA to determine where to store the data of the sweep it is acquiring. The last line of the for-loop makes sure that each sweep buffer entry points to the next one thus creating a *linked list*. When the first for-loop has finished, the Event field of the last sweep buffer entry is set to the event that is shared with the high-level DLL. All the event fields of the other sweep buffer entries are set to NULL, which means that they have no event they can signal. Thus, by testing the event field of the sweep buffer entry of a sweep that is (currently) being acquired for a valid event, the main driver can determine whether the current sweep is the last one in a buffer pool, and if it is, it is signaled (by the ISR of the main driver, see Section 4.5.2) in order to let the high-level DLL know that one buffer pool has been completely filled. The same initialization is done for the second buffer pool in the second for-loop. When the total memory buffer has been filled with sweeps, the next sweeps must be acquired again into the first buffer pool. Therefore, the last sweep buffer entry points back to the first one:

```
// Last entry points back to first:
pSwBufLinkList[ ulSweepsInDmaBuf – 1 ].pNext = &pSwBufLinkList[0];
```

When the last sweep buffer entry of a buffer pool is reached during acquisition, the previous buffer pool should already have been read out if the data is stored to disk. If data is only acquired into memory, earlier acquired data is overwritten and lost. Obviously, when acquiring data only to memory, the buffer must be large enough to hold all the data of one measurement. The last line of code sets the current sweep buffer entry to be the first one in the linked list:

```
// Set current buffer entry
pDeviceExt->pCurrentSweepEntry = &pSwBufLinkList[0];
```

This ensures that when the acquisition starts, the memory buffer is filled from the top. Figure 4-2 shows the two data pointers of each list element (the pointer to the header data and the pointer to the samples) as one arrow and the pointer to the next list element also as one arrow. Furthermore, it illustrates that the high-level DLL is signaled only after the last sweep in one of the two buffer pools has been acquired.

4.5.2.4 The data acquisition board's interrupt service routine

The interrupt handler of the data acquisition board driver, also called interrupt service routine (ISR), has also been modified drastically from the original one supplied by the manufacturer. All the original functionality, designed for acquiring single sweeps, has been preserved. By means of the flag bAcqMultSweeps, it is determined whether or not the current acquisition is a multiple sweep acquisition. Once the measurement has been started, the ISR controls all actions. Therefore, the ISR is the heart of the data acquisition board driver and will be explained in detail.

The flow graph in Figure 4-5 gives an overview of the operations performed by the ISR. The ICS650 data acquisition board can generate three PCI interrupts (see Section 3.2.2). Firstly, an ADC done interrupt is generated when the number of samples acquired after a trigger equals the programmed buffer length. Secondly, a DMA read done interrupt is generated when the acquired samples have been read from the swing buffer and thirdly a DMA write interrupt is generated when a transfer from the host to the 4K sample FIFO (see Figure 3-2) has completed. Only the first two interrupts are of

importance for the SHIRA application. Figure 4-5 shows that when the ISR is entered, it is first tested whether the interrupt is an ADC done interrupt, a DMA read done interrupt, or a DMA write interrupt. If the interrupt is an ADC done interrupt, the interrupt is first acknowledged (box 6) and then further ADC interrupts are prevented from occurring by disabling them in the interrupt mask register (box 7). This is necessary because another ADC interrupt, indicating that the read-side of the swing buffer is not empty, will occur if the ADC interrupts are not disabled (see Section 3.2.2). Next, it is tested whether the driver is performing a multiple sweep acquisition or a normal single-sweep acquisition. If it is a single-sweep acquisition, the interrupt handler returns after executing the original driver code (boxes 10 and 11). If a multiple sweep acquisition is being performed, which is the case for the SHIRA application, the newly added functionality, denoted by flow graph A in Figure 4-5, is executed. This functionality is explained in the next section.
If the interrupt is a DMA read done interrupt, the interrupt is first acknowledged (box 12) and then it is tested whether the driver is performing a multiple sweep acquisition or a normal single-sweep acquisition. If it is a single-sweep acquisition, the interrupt handler returns after executing the original driver code (boxes 15 and 16). If a multiple sweep acquisition is being performed, the newly added functionality, denoted by flow graph B in Figure 4-5, is executed. This functionality is explained in Section 4.5.2.6.
If the interrupt is a DMA write done interrupt, the original driver code is executed and the ISR returns (boxes 17 and 18). Finally, if the interrupt was not generated by the data acquisition board at all, the ISR returns immediately.

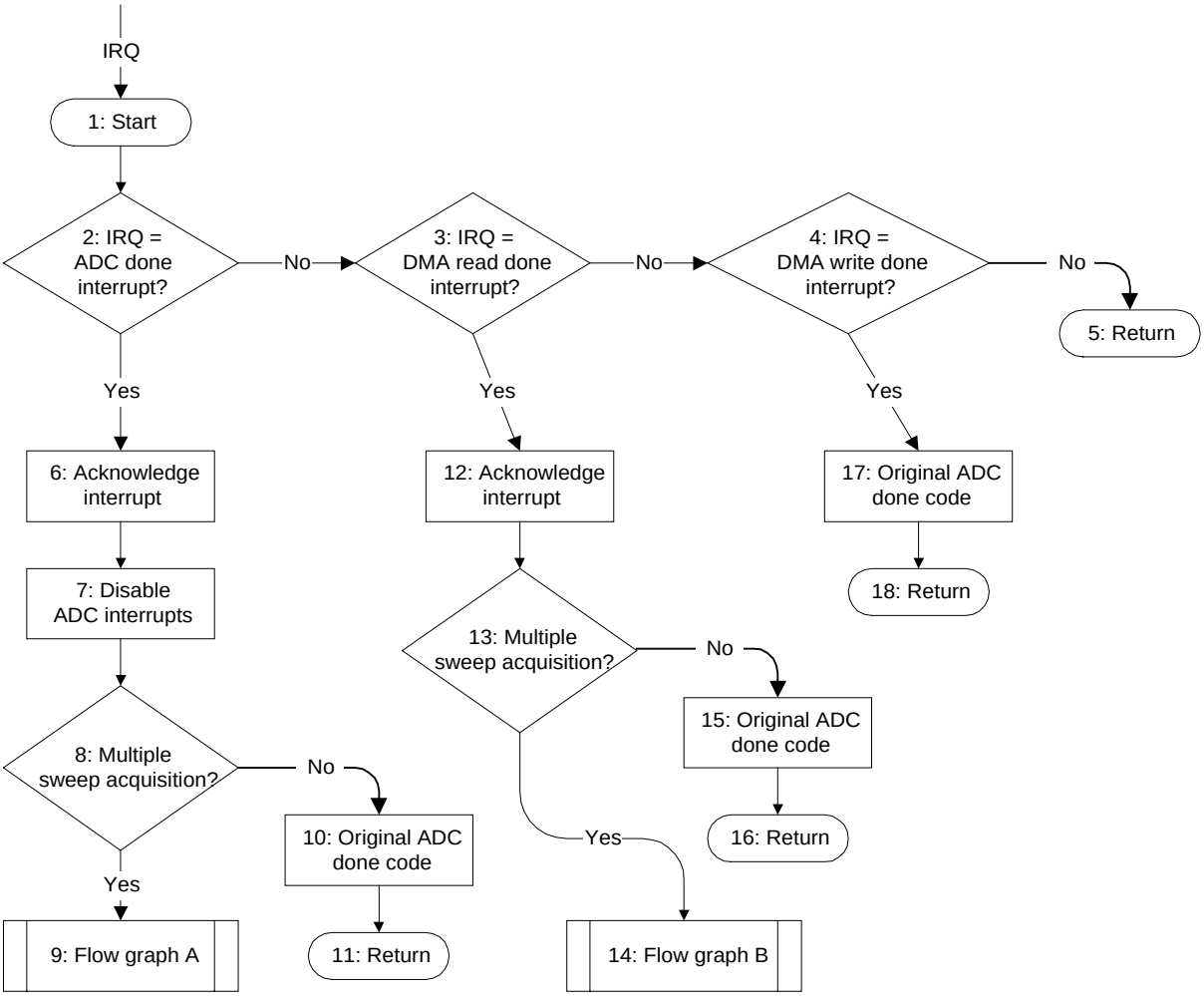

Figure 4-5 Overview flow graph of ISR of data acquisition board driver

4.5.2.5 Response to an ADC done interrupt for a multiple sweep acquisition

In response to an ADC done interrupt for a multiple sweep acquisition (as is the case with SHIRA), the ISR must (after acknowledging the interrupt, disabling further ADC interrupts, and determining whether the current acquisition is a multiple sweep acquisition) first request the last latched counter value from the counter board driver if the driver is loaded. Figure 4-6 shows the actions of flow graph A in Figure 4-5.

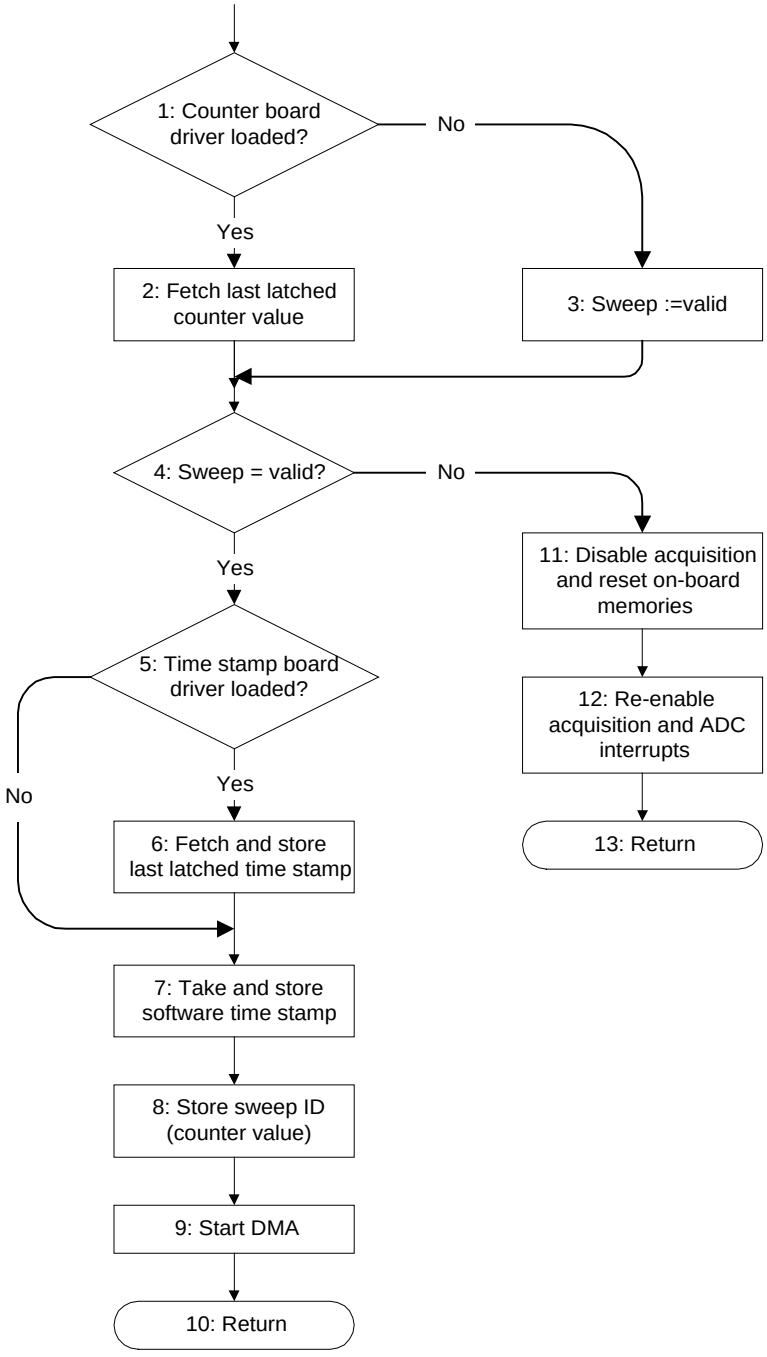

Figure 4-6 Flow graph A from Figure 4-5: response to an ADC done interrupt

How the counter board driver handles this request is explained in Section 4.5.3. The counter value that is fetched by a call to the counter board driver (this counter value is called the *sweep ID* of the current sweep) is used to determine whether or not the current sweep is valid, i.e. it lies within the user-defined recording window (box 4). Sweeps with a sweep ID that is not between the start and end sweeps of the recording window are discarded. If the counter board driver is not loaded, all sweeps are assumed to be valid and they will be acquired consecutively for one fixed angular position of the

radar.
Since a measurement starts at an arbitrary point in time, all sweeps belonging to the first (probably partial) scan are discarded. Once it has been determined that a sweep is invalid, the acquisition is disabled, the on-board memories of the data acquisition board are reset, and then the acquisition and interrupts are re-enabled (boxes 11 through 13). The reset of the memories is necessary in order to prevent another interrupt, indicating that the read-side of the swing buffer has not been read yet, from occurring when the interrupts are re-enabled.
If a sweep is valid, the operations indicated by boxes 5 through 10 in Figure 4-6 are carried out. If the time stamp board driver has been loaded, the driver asks it for the last latched time stamp and stores it into the buffer (boxes 5 and 6). This time stamp was taken at the moment the trigger occurred (see Section 4.5.4). Next, a software time stamp is taken and also stored in the buffer. Then, after the sweep ID also has been stored into the buffer, the DMA transfer of the samples from the board to the host memory buffer is started. The details of performing a DMA transfer on a Windows NT platform are explained in detail in chapters 12 and 17 of [Baker, 1997] and [Dekker and Newcomer, 1999] respectively.

4.5.2.6 Response to a DMA read done interrupt for a multiple sweep acquisition

In response to an DMA read done interrupt for a multiple sweep acquisition (as is the case with SHIRA), the ISR must (after acknowledging the interrupt and determining whether the current acquisition is a multiple sweep acquisition) first carry out some DMA specific de-initialization operations (box 1 in Figure 4-7), such as flushing buffers in order to ensure cache coherency, freeing DMA mapping registers, etc. Next, the pointer that points to the element of the driver's linked list corresponding to the current sweep is updated (box 2) in order to point to the next one (the linked list elements contain, among others, the destination addresses of the sweeps in the host memory buffer). The old pointer is saved because it is used in the rest of the routine.
Then the ISR checks whether all samples of the current sweep have been transferred to the host memory (box 3). If true, it is tested whether the current sweep is the last one in a buffer pool (box 4), and if this is the case, the event that signals the high-level DLL is set (box 5). If not all samples of the current sweep have been transferred, a new DMA transfer is initiated in order to transfer the remaining samples to the host memory buffer and the ISR returns (boxes 11 and 12). Next, if the user has not stopped the acquisition prematurely and if the acquisition is not yet complete, the ADC interrupts are re-enabled (boxes 6 and 7). From this it follows that the driver must also keep track of the buffer pool count. Then, it is tested whether the last sweep has been DMA-ed completely in case the user prematurely stopped the acquisition (box 8), and if this is true an event is generated to the IOCTL function that stops the board (box 9) and the ISR returns. This is necessary because the measurement can be interrupted at any time and a DMA action of a sweep can be busy at that moment. Otherwise, the ISR returns directly.

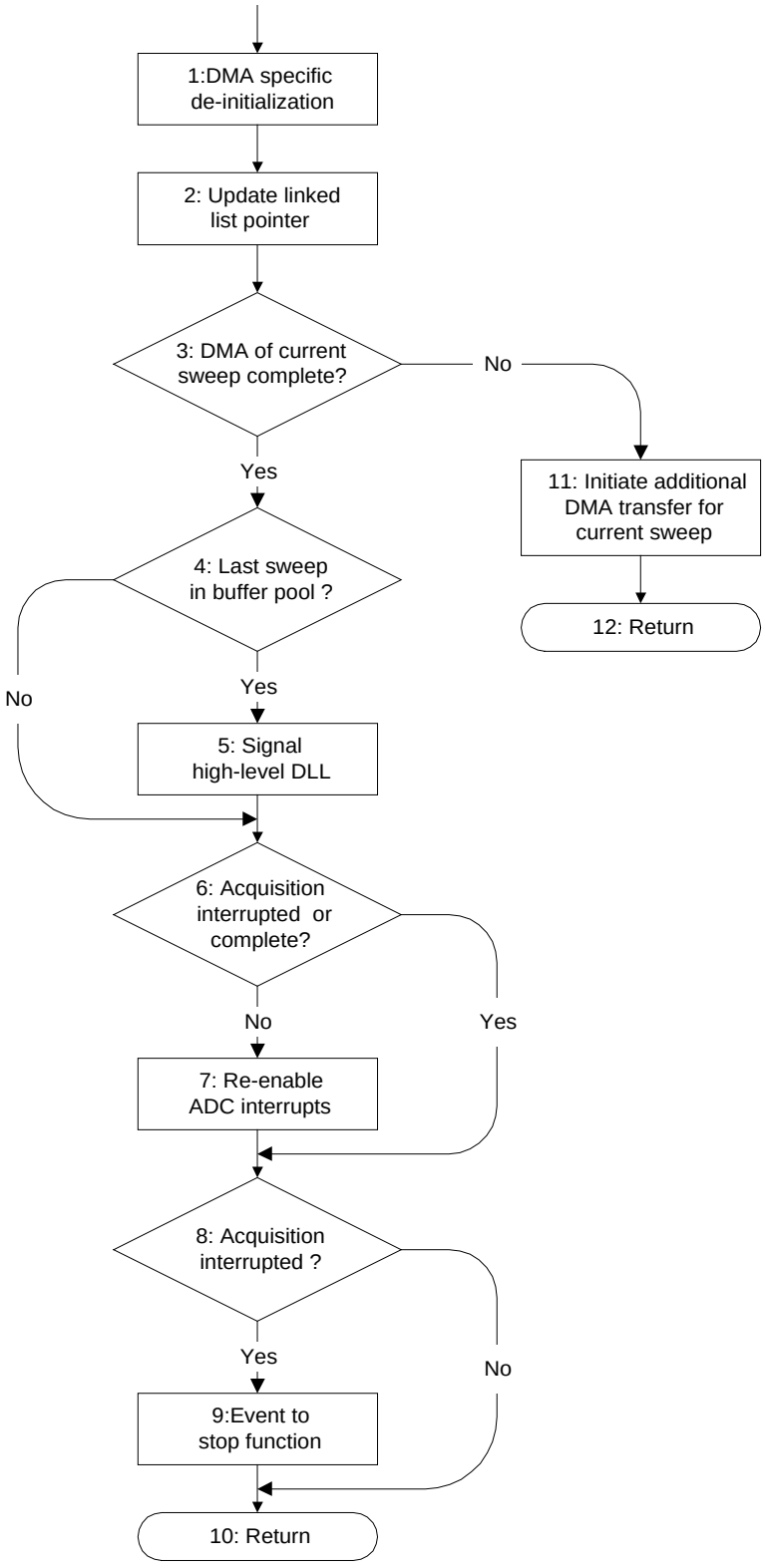

Figure 4-7 Flow graph B from Figure 4-5: response to a DMA read done interrupt

4.5.2.7 End of the measurement

In the part of the ISR that deals with the DMA read done interrupt, it is determined whether a measurement is complete. Like the high-level DLL, the ISR simply tests if the buffer pool count is zero (box 6 in Figure 4-7). If this is the case, the ADC interrupts are not re-enabled and therefore further acquisitions are blocked. The driver then waits until the high-level DLL calls drICS650IoctlStopMultSweepAcq (by calling the llics650StoptMultipleSweepAcq wrapper). This function is

also called if the user presses the “Stop Acquisition” button. It disables the acquisition and the interrupts, and after receiving the event indicating that the last sweep of a user-interrupted acquisition has been completely DMA-ed, it unlocks the user-allocated memory.

4.5.3 The driver for the counter board

The driver for the PA1700 counter board has been written from scratch. This means that in addition to the functionality specifically required for the SHIRA application, a lot of code for allocating the hardware, reporting the used hardware resources to Windows NT, releasing the hardware during driver unloads, etc. had to be written. Methods similar to the methods described in [Baker, 1997] and [Dekker and Newcomer, 1999] have been used for this purpose. This section only describes the functionality that is specifically required for the SHIRA application.
During the initialization of the counter board driver, values are programmed into several registers. The Control and Mode registers are part of the PA1700 hardware and are programmed with values that are entered in the Windows NT registry. The Control register is (among others) used to select the first counter module out of three available modules. Each counter module contains two 16-bit counters that can be cascaded internally by writing the proper values into the PA1700 registers. For the SHIRA application, a counting depth of 16 bits is enough since the maximum counter value equals the number of pulses the angular encoder generates per revolution of the radar antenna, which is 4096 (see Appendix A).
The Mode register is programmed in such a way that the 16-bit counting depth is selected and that counting direction is upwards.
When the main driver requests the counter board driver for the last latched counter value, the function named `Pa1700ReadCounterValue` in the PA1700 driver is called. This function reads the required value from the register and stores it at the memory location supplied by the calling main driver. Next, the driver tests whether a *North Reset* pulse has occurred. If this is the case the *index* bit (indication flag) is cleared in order to allow a new *North Reset* pulse to set the flag. Finally, the function returns. The index bit can also be cleared by means of the function `Pa1700SetIndex1Bit`. During the start of a measurement, the function `drICS650IoctlStartMultSweepAcq` from the main driver calls this function.

4.5.4 The driver for the time stamp board

Like the counter board driver, the driver for the BC630AT time stamp board has also been written from scratch. Again, only the functionality that is specific to the SHIRA application is described.
During the initialization of the time stamp board driver, values are programmed into several BC630AT hardware registers by the function `Bc630AtInitDevice`. The Time Code and Masks registers are programmed with values that are entered in the Windows NT registry. Because no external time source was available at the moment the driver was programmed, the time provided by the battery backed real time clock IC is used. However, the time (code) source can easily be modified by modifying the corresponding key value in the registry. The Masks register is (among others) used to control the External Event Capture (EEC) and to select the interrupt source. For the SHIRA application, the EEC mask is programmed in such a way that events on the EEC Input, in this case the trigger pulses, are captured on the rising edge of the input signal. The interrupt selection mask is programmed in such a way that an interrupt will be generated when a firmware command has completed.
When the main driver requests the time stamp board (BC630AT) driver for a time stamp, the function named `Bc630AtReqEvtTimeExp` in the BC630AT driver is called. This function first saves a pointer to the memory location where the time stamp has to be stored in the device extension. Next, the time stamp that was latched at the moment the last trigger pulse occurred is actually requested by a firmware command, and the function returns while the request is being dealt with. The BC630AT requires approximately 150 microseconds to load the data transfer address space (i.e. the registers that can be accessed by the host computer) with the latched time in response to a time request.
Because the main driver’s ISR has passed the pointer to the time stamp memory location to the BC630AT driver, this driver knows where to store the time stamp after it becomes available in the data transfer address space eventually. While the time stamp is transferred from the data latches to the data transfer address space, the main driver continues execution, i.e. the DMA is performed, etc. When the

firmware command has completed after about 150 microseconds an interrupt is generated, which is then serviced by the time stamp driver's ISR. The data is read from the device registers and stored at the time stamp memory location. Next, the "Clear Event Capture" firmware command is issued in order to reactivate the event capture circuitry.

5 Test results

The complete system has been tested extensively both by using artificial radar signals generated by pulse generators and by running tests with SHIRA itself. The final tests reported in this chapter only describe the tests performed with SHIRA, i.e. the real radar system. A lot of tests with the artificially generated radar signals have been carried out during the development and implementation of the data acquisition system and they yielded approximately the same results. So, they are only described in a qualitative way. During the final tests no fast disk system and no time stamp board were available, since they were borrowed and had to be returned. Therefore, the sweeps were first acquired into memory and afterwards written to a (slow) IDE disk, and software time stamps were used for the tests. During the tests with the artificially generated radar signals, it was proved that the data acquisition system also works well with a fast disk system and the time stamp board. So, the current operational system works with software timestamps and a slow IDE disk. However, the functionality for the hardware time stamps has been implemented (by means of the time stamp board driver) and the functionality for the fast disk system can be implemented by adding a few lines of code to the high-level DLL (this was done during the development of the system).

Test measurements have been carried out for different values of the acquisition parameters. It is determined whether sweeps are lost and whether the sweeps are digitized and recorded correctly within the user-defined region. During all tests, the PRF of the radar was fixed at about 1200 Hz and the revolution time T_R of the radar was approximately 1.5 s. Hence, the number of sweeps per revolution of the radar is approximately 1800.

The results of two measurements are reported. The parameters for the first measurement are $S_A = 0°$, $E_A = 360°$ (complete revolution of the radar antenna) and $N_{RW} = 512$. The parameters for the second measurement are $S_A = 270°$, $E_A = 90°$ and $N_{RW} = 1024$. Figure 5-1 shows the sweep IDs (counter values) of the first 6000 acquired sweeps for the first measurement.

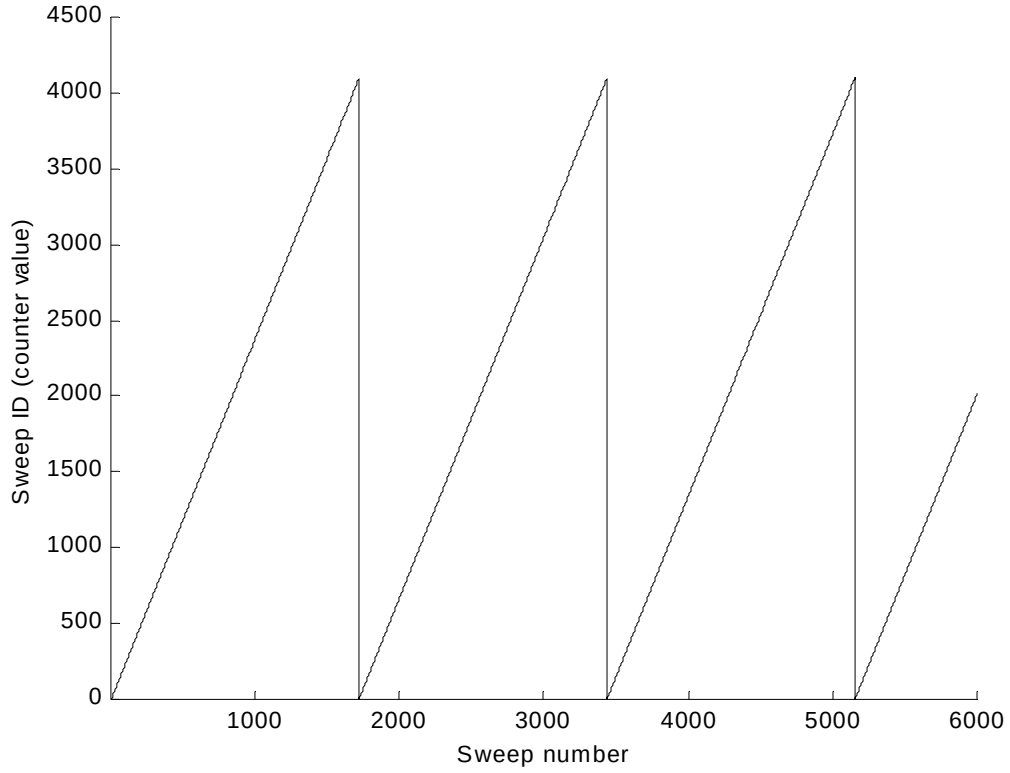

Figure 5-1 Sweep IDs for consecutively acquired sweeps ($S_A = 0°$, $E_A = 360°$ and $N_{RW} = 512$)

The figure clearly shows that all sweeps over the entire revolution of the radar antenna are acquired and that the counter values increase linearly, as expected. When the radar antenna reaches the *North Reset* reference point, the counter value is reset. It can also be seen that there are about 1800 sweeps in one revolution of the antenna. Since the angular encoder generates 4096 pulses per revolution of the radar antenna, the average difference of successive sweep IDs over one revolution is about $4096/1800 = 2.3$. In order to check whether sweeps have been lost, the time differences between successive sweeps are computed. If some of these time differences are much larger than the PRI (more than about 50 %), sweeps have been lost. Figure 5-2 shows a plot of the time differences for the first 6000 sweeps.

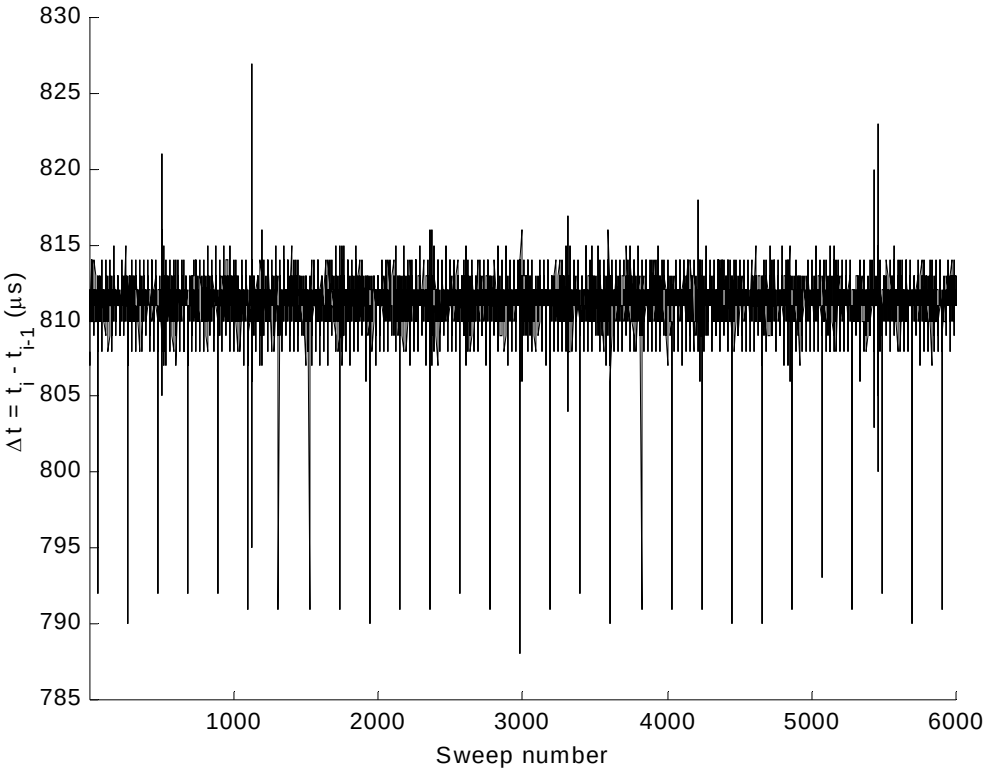

Figure 5-2 Time differences between successive sweeps for first test

The mean of the time differences (PRI) is $811.2\ \mu s$ which implies that the real PRF of the radar is about 1233 Hz. The standard deviation is $2.04\ \mu s$ and the maximum deviation from the mean is $24.2\ \mu s$ (3 %). As can be seen from these statistics and from the figure, none of the differences exceed the PRI very much and thus it may be concluded that no sweeps have been lost during the measurement.
The deviations from the mean are primarily caused by the Windows NT operating system. Since a lot of actions in the driver must be synchronized carefully, some actions block other actions by semaphores or by raising their interrupt request level. Therefore, the exact times at which actions happen (in this case the software time stamping) cannot be predicted exactly. If the time stamp board would be used (as has been done in earlier tests), the deviations would be in the order of a few microseconds. Now, the maximum deviation from the mean is about 3 %, which is not too high.

Figure 5-3 shows the 512 samples of one sweep. The signal shape is the same as the shape observed on an oscilloscope. With the artificially generated radar signals, different signal shapes were applied to the data acquisition system and in all cases the digitized signal shapes were as expected.

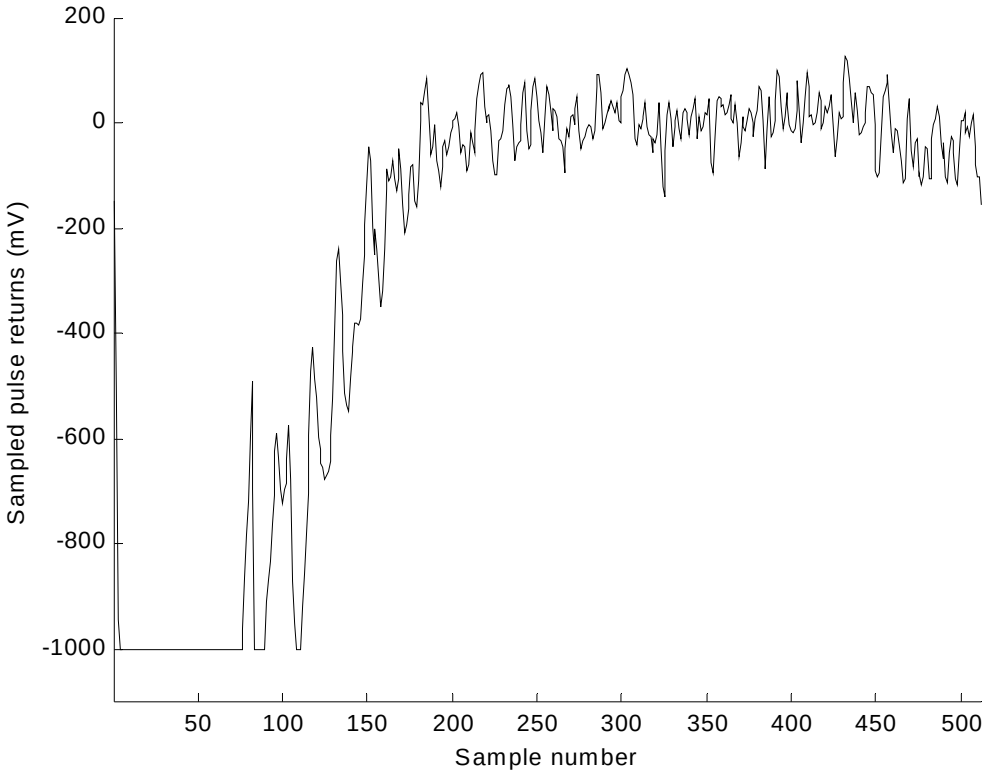

Figure 5-3 Sampled pulse return (sweep)

With the second test measurement it has been tested whether the acquisition of a user-defined region that starts before the *North Reset* reference point ($S_A = 270°$) and ends after it ($E_A = 90°$), is correctly acquired. The sweep length N_{RW} was 1024. Figure 5-4 shows the sweep IDs of the first 3000 acquired sweeps for this measurement.

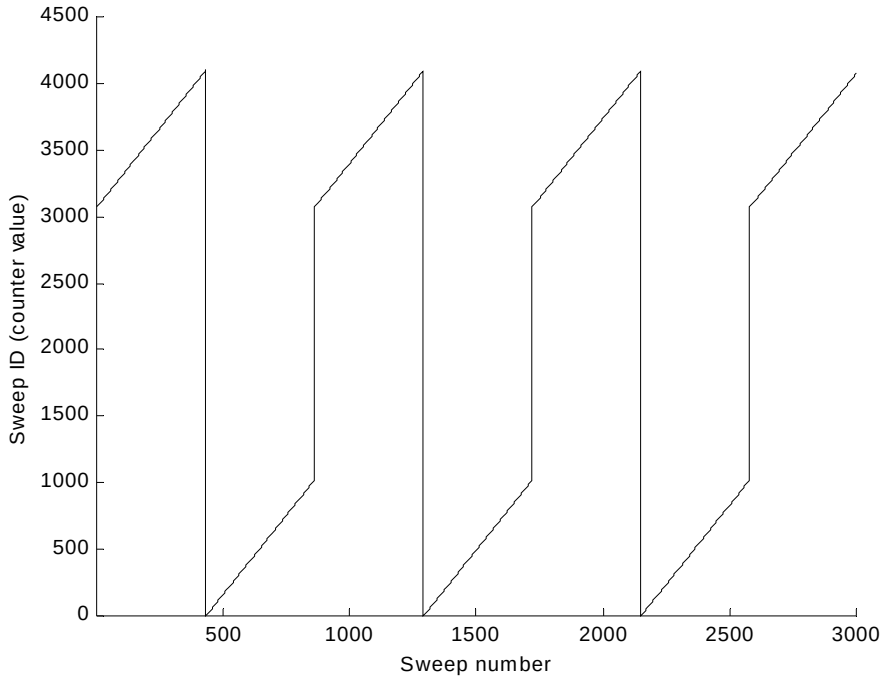

Figure 5-4 Sweep IDs for consecutively acquired sweeps ($S_A = 270°$, $E_A = 90°$ and $N_{RW} = 1024$)

Sweeps with a sweep ID between $(270 \bmod 360) \cdot 4096/360 = 3072$ and $(90 \bmod 360) \cdot 4096/360 = 1024$ are in the user-defined region. Figure 5-4 shows that the sweep ID of the first sweep in a scan is 3072 (or somewhat higher) and that sweeps are acquired up to the *North Reset*. Next, sweeps with sweep ID between zero and 1024 are acquired. After that, the area that is not in the user-defined region is skipped and the next scan begins. Figure 5-5 shows a plot of the time differences for the first 3000 scans.

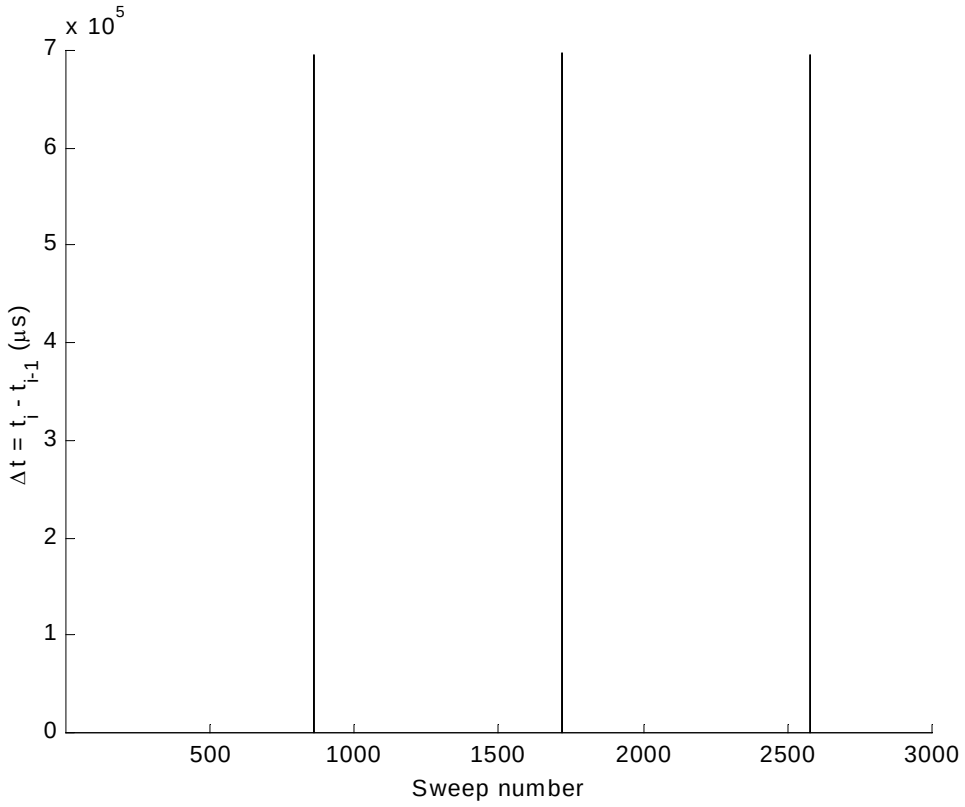

Figure 5-5 Time differences between successive sweeps for second test

The figure clearly shows large time differences between two successive scans. Since the azimuth span of the recording window is about $180°$, the time difference between the end of a scan and the beginning of the next scan is approximately $\frac{1}{2}T_R = 0.75$ s (0.70 s in the figure). Again, the mean of the time differences in one scan is $811.2\ \mu s$, which implies that the real PRF of the radar is about 1233 Hz. This is expected since the PRF was not changed. The standard deviation is $1.77\ \mu s$ and the maximum deviation from the mean is $14.2\ \mu s$ (1.75 %). Again, it can be concluded that no sweeps are lost.

As stated earlier, the tests with the artificially generated signals also revealed that the data acquisition system works well with a fast disk system. If the time stamp board is used, only PRFs up to 2 kHz can be used. Moreover, it has been tested what PRFs can be achieved. The highest PRF that can be achieved without losing data depends on the sweep length. The old system can deal with PRFs up to 2 kHz while the sweep length is 2048. The tests with the new system showed that, in contrast with the current (old) data acquisition system, the new system can deal with PRFs up to 4 kHz when the sweep length is 4096. Therefore, it may be concluded that the specifications are satisfied (see Appendix A).

6 Conclusions regarding the data acquisition system

Because the current data acquisition system cannot deal with high pulse repetition rates (PRFs), a new high-speed (40 MHz) data acquisition system had to be implemented. In order to do this, drivers have been written for the three main data acquisition components: a data acquisition board, a counter board and a time stamp board. A high level of modularity has been achieved because a separate driver has been programmed for each data acquisition component, instead of making one big monolithic driver as in the current data acquisition system. In addition to the driver software, user-mode software that interacts with these drivers has been developed. Tests have revealed that the data acquisition system complies with the given requirements and specifications. It can deal with PRFs up to 4 kHz while the sweep length (the number of samples per pulse return) is 4096, whereas the old system can deal with PRFs up to 2 kHz while the sweep length is 2048. Moreover, in contrast with the old system that can only deal with sweep lengths that are powers of two, the new system can deal with any sweep length up to 524288.

The high pulse repetition rates can be achieved because the used data acquisition board has a swing buffer with two dual-ported memories. This enables the acquisition to continue in one side of the swing buffer while the other is being read out. The data acquisition board of the old system only has one buffer and therefore lacks this advantage. If this single buffer has been filled, it must first be transferred to memory before the acquisition can continue.

The new system has two disadvantages that are also present in the current data acquisition system. The first disadvantage is that the amount of data that has to be transferred to the host memory is a factor 8/3 times the memory occupied by the actual data of interest. This is due to the fact (dictated by the hardware of the data acquisition board) that always two channels are recorded into the same side of the swing buffer and that the 12-bit data from the two channels is combined into a 32-bit word. No high-speed data acquisition board containing a swing buffer has been found that did not have this disadvantage. For this reason, this drawback cannot be eliminated at present.

The second disadvantage is that the currently used time stamp board can only deal with PRFs up to 2 kHz. However, this is not a severe problem because synchronization with external measurement systems is not (yet) required and therefore software time stamping can be used. In addition, a PCI board named BC635/637PCI that can handle PRFs up to 4 MHz is available at the moment. Due to the modularity, only the time stamp board driver has to be rewritten in case this new board is integrated into the system.

The SHIRA system will be used as a coherent radar for future applications, which requires that two or more channels are sampled simultaneously. Hence, a big advantage of the new system is the fact that the employed data acquisition board simultaneously samples its two channels.

7 SHIRA signal processing

7.1 Introduction and description of project goals

SHIRA is an instrument developed to satisfy the need for measurements of directional wave spectra, water currents and water depths. In order to obtain this oceanographic information, a number of signal processing operations has to be performed on the digitized data. The second part of the project that starts with this chapter deals with the spectral analysis of the radar data, which is one of the main signal processing operations. At present, the individual images out of a time-series are first scan converted, i.e. they are resampled from the nonuniform polar grid onto a Cartesian grid, and next the two-dimensional FFT is computed (see Section 7.2). However, TNO-FEL observed that the spectra obtained this way seem to be distorted, especially for higher frequencies. They suspect that this distortion arises from the Nearest Neighborhood interpolation of the available polar samples onto a Cartesian grid (see sections 7.2.1 and 7.2.2). The main goals of the second part of project relate to the examination of this suspicion, making an inventory of alternative methods for computing the spectra, and finally selecting one of the alternatives for further exploration. During this exploration, it will be tested whether the alternative method gives rise to less distortion. In addition, the spectral resolution of the current will be compared to that of the new method. Because using the polar-grid samples directly intuitively seems to exploit the given information better, the alternative named *Polar Fourier Transform (PFT)*, which is described in Chapter 8, is chosen to be investigated in detail. With the PFT, in fact the scan conversion is skipped and a Fourier transform (FT) that takes polar input data is used. That way use can be made of the specific properties of the data, such as the fact that the data is equidistant in the range direction.

Since only about three months were available for the second part of the project, the research had to be restricted to some main themes. The most important topic concerns the comparison of the spectra computed with the (Discrete) PFT to the spectra computed with the currently used method by means of quality measures because the DPFT will be used for the SHIRA application only if it provides better performance (primarily regarding the quality) than the current method. In order to do this, the (D)PFT is computed indirectly via Hankel transforms because methods for the fast evaluation of Hankel transforms are described in the literature. The way the (D)PFT can be computed via Hankel transforms is the second main theme and it is described in Section 8.3. Within the scope of this report, the computational complexity of the DPFT is of less importance. However, since it will be important if the DPFT is to be used in practice, an overview of algorithms for the (fast) numerical evaluation of Hankel transforms found in the literature is also given in Chapter 8. Because criteria are required for the fair comparison of the DPFT to the current method, quality measures are defined in Chapter 9. In addition, the way they are computed is explained. Then, in Chapter 10 the comparison of the DPFT, computed with the method presented in chapter 8, to the current method is discussed and the results are given. In order to make a fair comparison, the current method is altered a little bit and the resulting method is called the conventional method (see Section 10.1). Since only the two-dimensional spatial data has to be scan converted, the two-dimensional problem is considered. So, the research described here is restricted to one image out of a time series. Finally, in Chapter 11 conclusions and recommendations are given.

Since the primary concern is to investigate the quality of the PFT as compared to the conventional method, and not yet the efficient implementation of the employed algorithms, all simulations are done in Matlab. When it turns out that the investigated method is useful for future use, it can be implemented efficiently later on. Simulated data is used for the simulations because the input field can then be chosen in such a way that the output is known. Moreover, artifacts that are present in real radar data (such as shadowing), and which are not considered in this project, can be avoided. In order to simulate the current and new spectral analysis methods, a number of Matlab script files, the so-called m-files, are written. The most important files can be found in the appendices and are explained in the text.

This chapter provides an overview of the currently used methods and points out some possible causes for the distortions in the spectra. First, in order to make clear the context of the spectral analysis with respect to the other signal processing operations, an overview of the most important operations together with their problems (when relevant) is given in Section 7.2.
Next, the parameters that are used throughout the second part of the report are defined and explained in Section 7.3. Appendix C lists the most important parameters together with the names that are used in the m-files. Section 7.3 also describes the structure and generation of the polar input (simulation) data since all simulations have this part in common. After that, the simulation of the currently used method is described in Section 7.4 followed by a brief discussion on its complexity in Section 7.5. Finally, the chapter ends by briefly discussing possible improvements and alternatives in the last section.

7.2 Currently used data analysis methods

7.2.1 Overview

The first part of the currently used data analysis technique, which is also the topic of the second part of the project, concerns the computation of the (directional) wave spectra from the available raw data. After that, the spectra of the raw data have to be filtered in order to obtain more reliable spectra. Moreover, other quantities, such as the water current vector and the water depth, can be derived from the spectra. Usually, a time series of two-dimensional images of the sea surface is recorded, yielding a three-dimensional database with two *polar* spatial dimensions and one time dimension. In order to be able to analyze the data spectrally by means of the standard Cartesian FFT, the nonuniform polar input data of each image in a time series of images has to be resampled onto a uniform Cartesian grid. The procedure of resampling the polar data onto a Cartesian lattice is called *scan conversion* (left side of Figure 7-1).

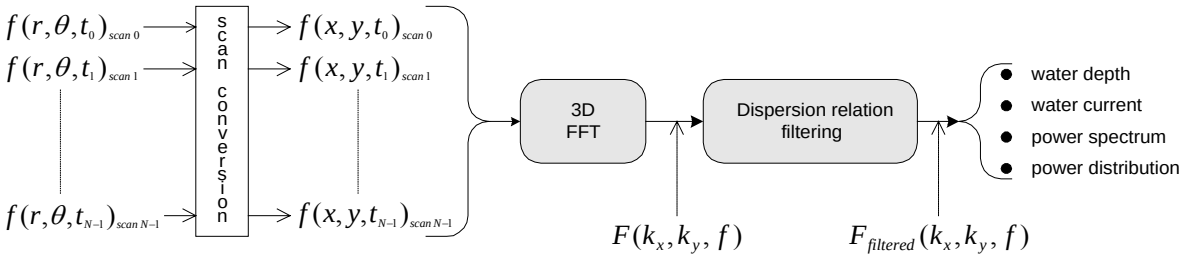

Figure 7-1 Overview of SHIRA signal processing

The result of the scan conversion step is again a three-dimensional database, in which two-dimensional *Cartesian* spatial and one-dimensional temporal sea state information is stored. Like all other signal processing operations, the scan conversion is currently done in Matlab. Section 7.2.2 describes the scan conversion process in more detail. After the scan conversion, the Cartesian database is transformed to the wavenumber-frequency domain by computing the three-dimensional FFT. The resulting spectrum contains two wave number dimensions (k_x, k_y) and one frequency dimension (f). Each spectrum of an individual image of the time series has a $180°$ ambiguity due to the fact that the Fourier transform of real data is conjugate-symmetric (see equation (7-3)). Integration over the positive frequencies gives a full two-dimensional wave spectrum without that ambiguity [Seemann and Ziemer, 1995]. The imaging by the radar system is nonlinear due to shadowing and also due to the modulation mechanism of the radar cross-section of the ocean surface. The nonlinearity manifests itself with harmonics at the image spectra besides the fundamental mode. Moreover, the images are contaminated with noise and other artifacts. Because it is known that the spectral energy of the sea state satisfies a certain relation for surface gravity waves, called dispersion relation, this relation can be used as a filter in the determination of more reliable wave number spectra. Besides that, by fitting the dispersion relation to the measured spectrum, it is possible to estimate the water depth and the water current vector [Senet, Seemann and Ziemer, 1997]. After a reliable three-dimensional spectrum

has been obtained by filtering the result from the 3D FFT with the dispersion relation, the power spectrum, i.e. the power as a function of frequency, the power distribution, i.e. the power as a function of the wave number, and other similar spectra are computed (see Figure 7-1 and Section 7.2.5). The computation of the spectra of the raw data by means of the current method is simulated by the m-file `CurrMeth` which is listed in Appendix F and explained in Section 7.4. The next sections explain the separate processing stages of the current method in more detail.

7.2.2 The scan conversion and its problems

The scan conversion is necessary to make the data suitable for the 3D FFT (in fact for the 2D FFT of each individual image in a time series). The polar data of a circular section that encompasses the user-defined rectangle must be interpolated onto that rectangle (for more details, see Section 7.4). This two-dimensional interpolation is one of the most time-consuming operations. It is expected that the interpolation step introduces distortions in the spectra that are computed from the interpolated data. Only ideal interpolation would result in spectra with no (artificial) distortion due to the scan conversion step. Section 7.4, which deals with the simulation of the currently used method, shows two spectra that are computed with different interpolation methods. Figure 7-2 shows the interpolation from the nonuniform polar grid onto the uniform Cartesian grid. The points indicated by small circles are the known points (data samples) of the sweeps that are nonuniformly spaced in the azimuth direction. With SHIRA, the maximum variation in the angular velocity (A_d), and consequently in the angular spacing, is 25 %. In order to avoid aliasing, the sampling parameters of the polar grid have been chosen in such a way that the Nyquist criterion is satisfied in both the range and azimuth directions. The points indicated by stars are the points of the selected rectangular grid on which the two-dimensional FFT will be performed.

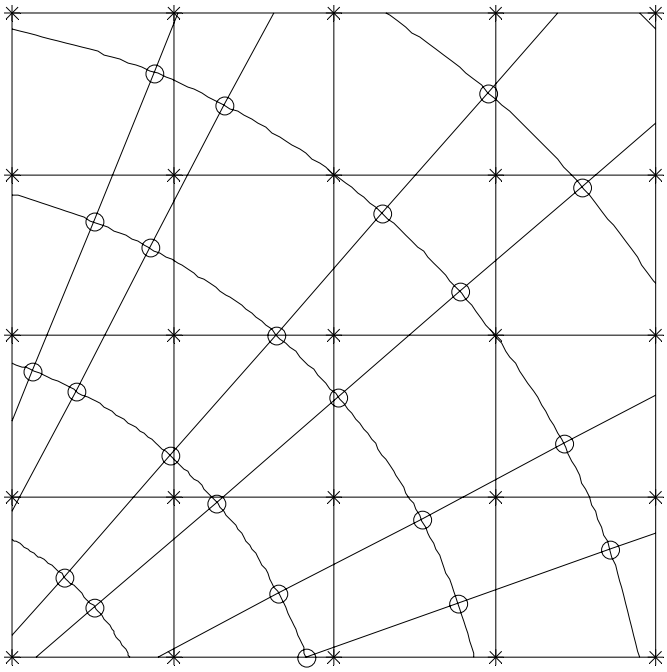

Figure 7-2 Interpolation from nonuniform polar grid to uniform Cartesian grid. Variation in angular velocity is 25 % at maximum. (o) Known points; (*) required points for FFT

Because the sample angles are different for each image in a time series, look-up tables cannot be used for the transformation of the coordinates. A number of interpolation methods can be used like Nearest Neighborhood (NN), bi-linear, cubic spline [Johnson, 1982], polynomial estimation methods [Zakhor and Alvstad, 1992], etc. The problem is that only the simplest methods, i.e. NN and bi-linear can be computed relatively fast. Always, *accuracy has to be traded off against simplicity (directly related to*

the computation time). TNO-FEL currently uses the NN method because it is the fastest method. With NN (which will be explained further on), a coordinate for which the value has to be determined is assigned the value of the point in the original grid that is closest to that coordinate; it is a zero-order interpolation (strictly speaking, it is not an interpolation). However, NN has a number of serious drawbacks. Obviously, it is a very coarse interpolation method. Energy can be lost or gained and is not assigned in a weighted way to the points of the uniform rectangular grid. This introduces *distortions in the spectra* that are computed after the interpolation. Obviously, most distortion is introduced at high spatial frequencies (i.e. large wave numbers). The high spatial frequencies correspond to the small wavelengths, at which only a few sample points per wavelength are available. Hence, the higher the spatial frequencies, the larger the errors made by the NN interpolation. It is possible to eliminate the distortion totally because methods exist that ideally reconstruct two-dimensional signals from nonuniform samples in polar coordinates if the function is band limited (which is the case for the SHIRA application) as described in [Marvasti, 1990] and [Marks II, 1993]. However, again this involves a complexity that is unacceptably high.

As will be explained in Section 8.3, the PFT still requires an interpolation step. However, this is a one-dimensional interpolation instead of a two-dimensional interpolation. One of the main goals of examining the alternative method, the PFT, is to determine whether the drawbacks of the NN interpolation are less severe for the PFT than for the current method (instead of using a better interpolation method with the current method). This comparison will primarily be based on the amount of distortion in the spectra and secondarily on the spectral resolution.

The interpolation can be performed in two ways that correspond to the polar and Cartesian coordinate systems respectively. In the first method, the *coordinates* of the uniform rectangular grid are expressed in polar *coordinates* and then *the interpolation is performed in the polar domain*. The second method does it the other way around: it expresses the nonuniform (input) coordinates in Cartesian coordinates and then *the interpolation is performed in the Cartesian domain*. Because interpolation in the polar domain is faster than interpolation in the Cartesian domain (due to the fact that the data vectors are regularly spaced in the polar domain, but not in the Cartesian domain), TNO-FEL currently uses the first method. However, a possible cause for the distortions shows up here since the interpolation results of both methods will differ due to the fact that the distances as measured in polar coordinates are different from the distances measured in the Cartesian domain, which are the "real" Euclidian distances. Therefore, the differences between the different interpolation domains will be examined in more detail. Also some comparison results are reported in Chapter 10. The NN interpolation in the polar domain does not yield the intuitively desired result because a coordinate of the rectangular grid for which the value has to be determined is assigned the value of the point in the polar grid that is nearest to that coordinate as measured in polar distances instead of the "real" Euclidian distances. In order to illustrate this more clearly, an (exaggerated) example is given in Figure 7-3. The points A, B, C and D are the known points of the nonuniform polar lattice and P is a point of the uniform rectangular lattice required for the FFT for which the value has to be determined. The vertical line L is located halfway between A and B (see further). In case the Cartesian coordinates of P are transformed to polar coordinates and the NN interpolation is performed in the polar domain, the value V_P of the point P is determined in the following way (see the flow diagram in Figure 7-4 and the decision region in Figure 7-5): firstly, it is determined whether the angle θ_1, which is the absolute value of the angle of the line OP with the line trough the points O and B (or A), is smaller than the angle θ_2, which is the absolute value of the angle of the line OP with the line trough the points O and C (or D). If this is true, the value of P will either be the value of A (V_A) or B (V_B). As can be seen in the figure, this is the case for the discussed example. If $\theta_1 > \theta_2$, the value of P will either be the value of C (V_C) or D (V_D). Next, Figure 7-4 shows that it is determined whether the absolute value of the difference between r_P, which is the radius of the circle through P (see Figure 7-3), and r_0, the radius of the circle through A and D, is smaller than the absolute value of difference between r_P and r_e (the radius of the circle through B and C). If this is true, P is assigned the value of A, i.e. $V_P = V_A$, else the value of B. In the example r_P is chosen to be little greater than the mean of r_0 and r_e: as shown in Figure 7-3 the circle through P crosses the positive x-axis to the right of the intersection point of the x-axis and the line L,

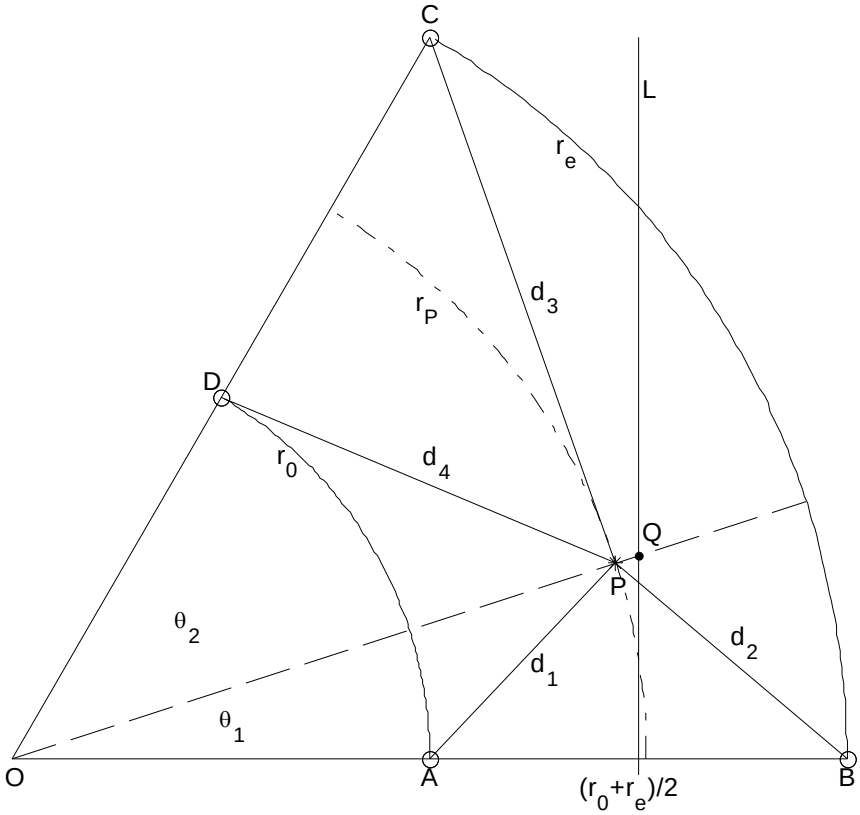

Figure 7-3 Close-up of NN interpolation. (o) Known points; (*) required point(s) for FFT

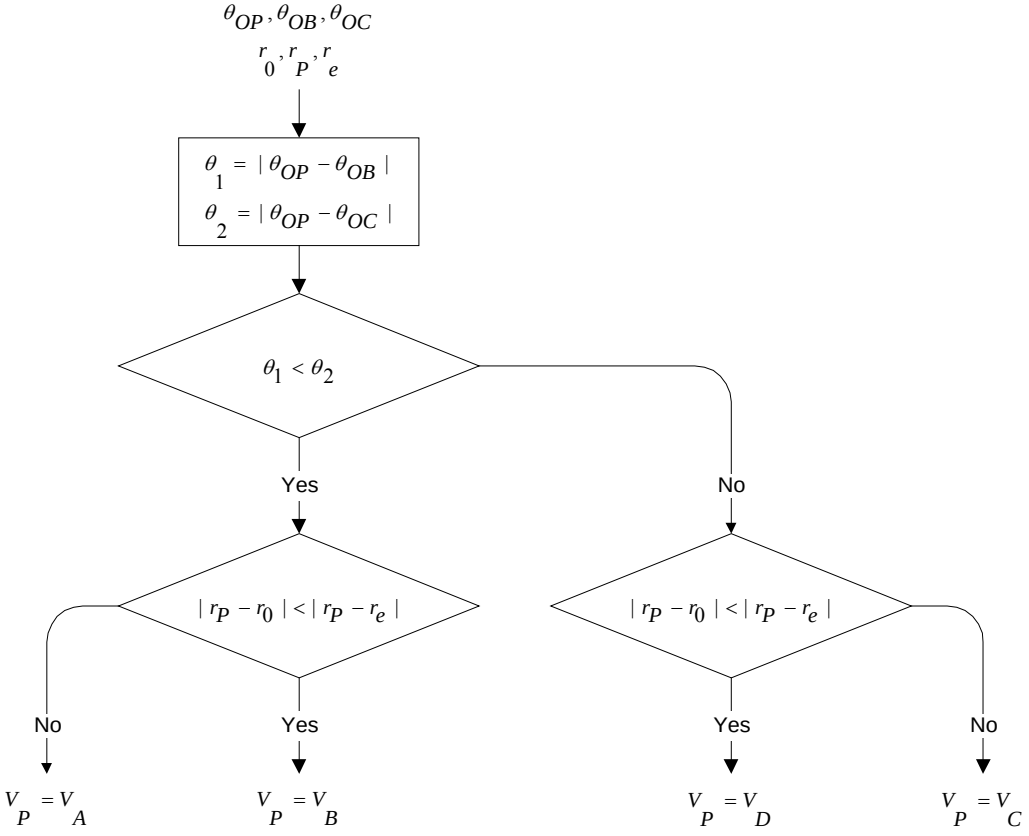

Figure 7-4 Flow diagram of the determination of closest point in the polar domain

which is the line perpendicular to the x-axis halfway between A and B. So, the interpolation algorithm will assign the value of B to P, i.e. $V_P = V_B$. It is also possible to first compare the differences in the range direction and then the differences in the azimuth direction. Obviously, for a better *spatial energy distribution*, the Euclidian distance measurement must be used to determine the closest neighbor. In this case, the nonuniform polar input coordinates are converted to rectangular coordinates and then the NN interpolation is performed in the Cartesian domain. Since d_1 is smaller than d_2 for all points on the line OQ that are left of the vertical line L and since θ_1 is smaller than θ_2, all these points are closer to A than to B. Hence, the NN interpolation algorithm will now demarcate A as the nearest neighbor of P and therefore P is assigned the value of A instead of the value of B. So, it can be concluded that interpolation in the polar domain yields results that are different from the results obtained by interpolation in the Cartesian domain. For all $\theta_1 < \theta_2$, the NN interpolation results will differ for all points on the line interval PQ in Figure 7-3. The total area for which the results of the NN interpolation in the two coordinate systems are different is shown in Figure 7-5. It is the shaded area enclosed by the line halfway between A and B that is orthogonal to AB, the line halfway between C and D that is orthogonal to CD and radius r_m, which is the mean of r_0 and r_P.

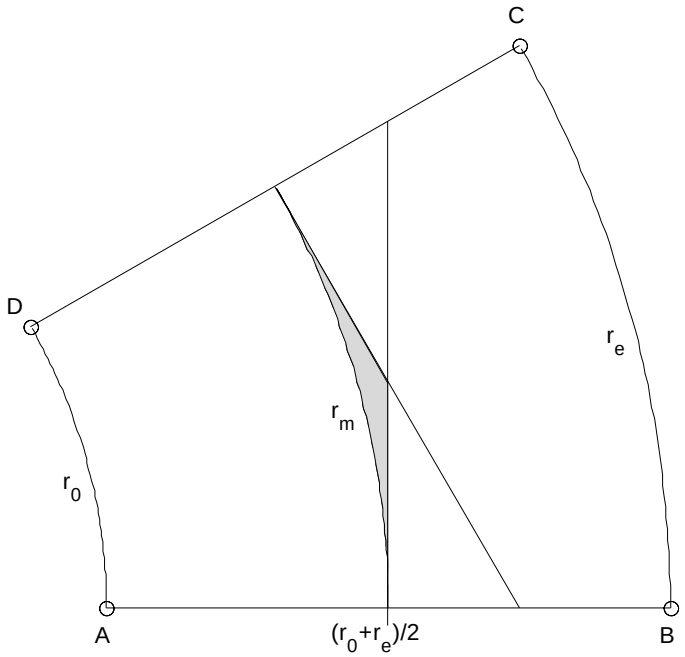

Figure 7-5 Area for which interpolation in polar domain is different from interpolation in Cartesian domain

The figure shows that, even for this exaggerated example, the area for which NN interpolation in the polar domain differs from NN interpolation in the Cartesian domain is rather small. Therefore, it is expected that the interpolation results, and hence the spectra computed from these results, do not differ very much. However, since this is a possible cause of distortion, some simulations are performed in order to check whether this expectation is true. Section 7.4 describes the file that is used for this purpose. The simulations for nominal SHIRA parameters (see Appendix A and Section 7.3) show that the expectation is true because (virtually) no difference is observed in the spectra. Hence, it can be concluded that the fact that the interpolation is currently performed in the polar domain does not cause significant distortions in the spectra.

7.2.3 Three-dimensional FFT

After the scan conversion, the data of each image of a time series is available on a uniform rectangular grid. The size and position of this rectangular area that will be extracted out of each two-dimensional image of a time series is selected by the user. The rectangular area has to be fully contained within the

circular section. Rotations of the rectangle are allowed, however, in this case the computed spectra also have to be rotated by the same angle as the rectangular data window (see Section 10.3). The size of the rectangle is determined by the number of uniformly spaced grid points in the x-direction and y-direction N_x and N_y, and the grid sizes r_x and r_y (sampling distances) in both directions (see Appendix C). The position is given by the coordinates of the center (of mass) of the rectangle: P_{cx} is the x-coordinate of the center and P_{cy} the y-coordinate.

If $f(\vec{x},t)$ represents a continuous signal that is a function of the spatial position $\vec{x}$ and time t, the three-dimensional continuous Fourier-transform can be used to obtain the wavenumber-frequency spectrum [Dudgeon and Mersereau, 1984]:

$$F(\vec{k},\omega) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\vec{x},t)\exp[-j(\vec{k}\cdot\vec{x}-\omega t)]d\vec{x}\,dt\,. \tag{7-1}$$

The variable ω represents the temporal frequency. Analogously, the wave number vector $\vec{k} = (k_x, k_y)$ represents the spatial frequency. The inproduct $\vec{k}\cdot\vec{x}$ is by definition $\vec{k}\cdot\vec{x} = k_x x + k_y y$. Because the data is available for discrete values only and the Nyquist criterion is satisfied (i.e. it is assumed that no aliasing is present in the radar data), the discrete Fourier transform is used to obtain an estimate of the spectrum of $f(\vec{x},t)$:

$$F(i_{k_x}, i_{k_y}, i_\omega) = \sum_{n_x=0}^{N_x-1}\sum_{n_y=0}^{N_y-1}\sum_{n_t=0}^{N_t-1} f(n_x,n_y,n_t)\exp[-j2\pi(i_{k_x}\,n_x/N_x + i_{k_y}n_y/N_y - i_\omega n_t/N_t)\,, \tag{7-2}$$

where

$i_{k_x} = 0, 1, \cdots, N_x - 1$, $i_{k_y} = 0, 1, \cdots, N_y - 1$, $i_\omega = 0, 1, \cdots, N_t - 1$ and $F(i_{k_x}, i_{k_y}, i_\omega)$ and $f(n_x, n_y, n_t)$ are the discretized versions of $F(\vec{k},\omega)$ and $f(\vec{x},t)$ respectively.

Due to the fact that an individual image $f(n_x, n_y)$ out of a time series is real-valued, its spectrum has the property:

$$F(i_{k_x}, i_{k_y}) = F^*(-i_{k_x}, -i_{k_y})\,. \tag{7-3}$$

Since the spectrum is conjugate-symmetric, its amplitude spectrum has a $180°$ ambiguity. This means that, if only one individual image is considered, it is not known whether waves with a wave number $\vec{k}$ propagate in the $\vec{k}$ or the $-\vec{k}$ direction. This ambiguity can be resolved by integrating over the positive frequencies as pointed out in [Seemann and Ziemer, 1995] and [Senet, Seemann and Ziemer, 1997]. This way, the dynamic information is utilized to obtain a full two-dimensional wave number spectrum without the $180°$ ambiguity. Furthermore, the symmetry relation (7-3) implies that only one half of the DFT coefficients of an individual image need to be computed. The remaining half is known immediately by symmetry.

As has been pointed out in the first part of the report, a time stamp is recorded for each sweep in a two-dimensional image. For the three-dimensional FFT however, it is necessary that only one time stamp is assigned to each image. Because the radar antenna rotates at a limited angular velocity, *the image that is recorded during one revolution of the radar is not an instantaneous ("frozen") image of the sea surface* (during the time the antenna rotates, the sea surfaces changes). Because the dynamics of the sea surface (determined by both the wave number and the frequency, see next section) are slow in comparison to the angular velocity of the radar antenna, the two-dimensional images may be considered to be instantaneous. Therefore, for each image the time stamp of the same sweep has to be used as the third-dimension input for the FFT (for example the time stamps of all first sweeps in the user-defined recording section can be used). These time stamps are denoted in Figure 7-1 as $t_0, t_1, \cdots, t_{N-1}$. The Matlab routine FFTN is used to transform the three-dimensional database that is obtained this way to the wavenumber-frequency domain.

7.2.4 The dispersion relation

Theoretically, for each wave number there is only one frequency for which energy can be present in the ocean wave field. The wavenumber-frequency "combinations" for which energy can be present in

the spectra are given by the dispersion relation, which describes the physical interaction between the inertial and gravitational forces. As has been pointed out in Section 7.2.1, it is used for two purposes. Firstly, it is used as a filter in order to obtain a reliable three-dimensional wavenumber-freqeuncy spectrum. The imaging by the radar system is nonlinear due to shadowing and also due to the modulation mechanism of the radar cross-section of the ocean surface. This nonlinearity manifests itself with harmonics at the image spectra besides the fundamental mode. Moreover, the images are contaminated with noise and other artifacts. Because it is known that the spectral energy of the sea state satisfies the dispersion relation, this relation can be used as a filter in the determination of more reliable wavenumber spectra. After a reliable three-dimensional spectrum has been obtained by filtering the result from the 3D FFT with the dispersion relation, the power spectrum, i.e. the power as a function of frequency, the power distribution, i.e. the power as a function of the wave number, and other similar spectra are computed (see next section).
Secondly, the dispersion relation is used to estimate the water current vector and the water depth by fitting it to the measured spectrum [Senet, Seemann and Ziemer, 1997].
The dispersion relation looks as follows (see [Kleijweg, 1991] and [Senet, Seemann and Ziemer, 1997]):

$$\omega = \sqrt{g|\vec{k}|\tanh(|\vec{k}|d)} + \vec{k}\cdot\vec{u}\,, \tag{7-4}$$

where ω is the temporal frequency, g the gravitational acceleration, $\vec{k}$ the wave number vector, d the water depth and $\vec{u}$ the water current vector with respect to the radar. All energy in the measured spectrum must be localized on a shell defined by the surface wave dispersion relation. This relation defines a two-dimensional surface in the three-dimensional wavenumber-frequency space. Figure 7-6 shows a plot of the dispersion relation for an infinite water depth ($\tanh(|\vec{k}|d) = 1$) and no water current.

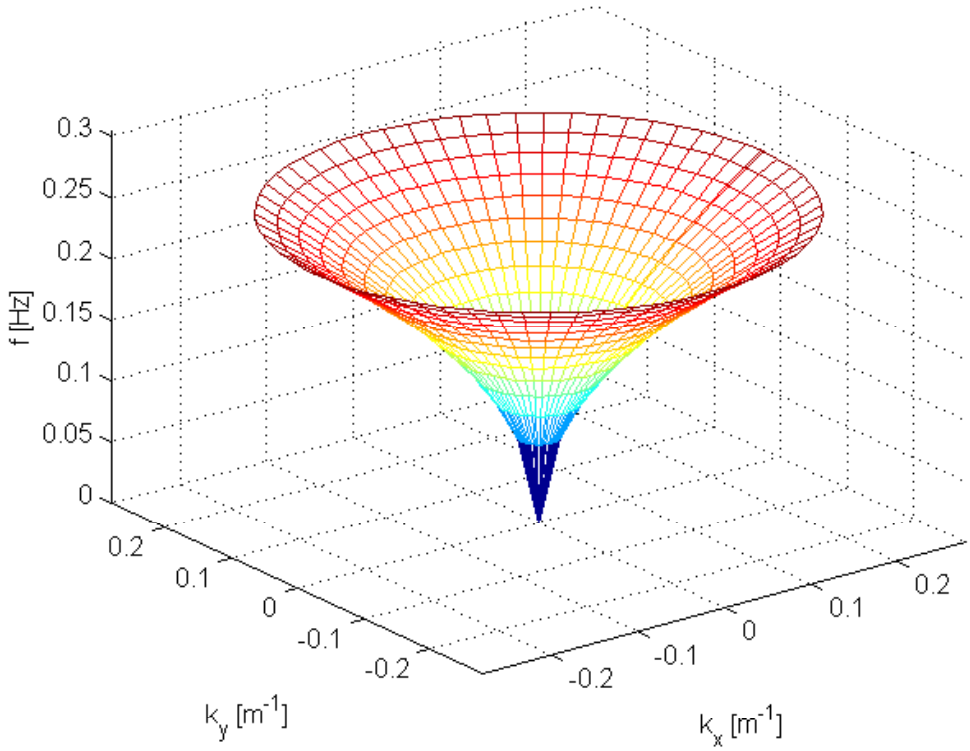

Figure 7-6 Dispersion relation for infinite water depth and zero water current

The derivative of the dispersion relation with respect to the modulus of the wave number, $v_g = d\omega/dk$, is the group velocity of the waves ($k = |\vec{k}|$). The phase velocity is given by $v_f = \omega/k$. As has been described above, theoretically there is only one frequency for each wave number for

which energy can be present in the ocean wave field. However, the three-dimensional database resulting from the 3D FFT contains discrete data, which do not exactly satisfy the dispersion relation. In order to avoid that all values of the spectra will be filtered out, the filter must have a certain "width", which can be varied by allowing small variations in $\vec{k}$, d and $\vec{u}$ for each frequency. Since the purpose of this section was to give an overview of the use of the dispersion relation and because it is not used in this project, it is not further discussed here. The interested reader can find more details in [Kleijweg and Greidanus, 1994], [Kleijweg, 1991], [Seemann and Ziemer, 1995] and [Senet, Seemann and Ziemer, 1997].

7.2.5 The final spectra

The final signal processing operations involve the calculation of different kinds of spectra, the water depth and the water current vector. This section lists the spectra that can be computed from the filtered three-dimensional spectra database, and in which the user is interested. The continuous frequency spectrum, i.e. the spectrum as a function of frequency, is given by:

$$FS(\omega) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(k_x, k_y, \omega)\, dk_x\, dk_y . \tag{7-5}$$

The continuous power spectrum, i.e. the power as a function of frequency, is given by:

$$PS(\omega) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \left|F(k_x, k_y, \omega)\right|^2 dk_x\, dk_y . \tag{7-6}$$

The continuous directional wave number spectrum, i.e. the spectrum as a function of the wave number vector, is given by:

$$WS(k_x, k_y) = 2 \cdot \int_{0}^{\infty} F(k_x, k_y, \omega)\, d\omega . \tag{7-7}$$

The continuous directional power distribution, i.e. the power as a function of the wave number vector, is given by:

$$PD(k_x, k_y) = 2 \cdot \int_{0}^{\infty} \left|F(k_x, k_y, \omega)\right|^2 d\omega . \tag{7-8}$$

The total power of all waves that propagate in a specific direction, determined by the angle of a slice of the Fourier transform in polar coordinates, is given by:

$$DPD(\phi) = 2 \cdot \int_{0}^{\infty}\int_{0}^{\infty} \rho \left|F(\rho, \phi, \omega)\right|^2 d\rho\, d\omega . \tag{7-9}$$

Here, the polar coordinates ρ and ϕ are used for the wave number vector. Note that ρ equals $k = |\vec{k}|$, the modulus of the wave number vector (i.e. they represent the same variable and always have the same meaning). In the remainder of this report, ρ will be used when the fact that the Fourier spectrum is available in polar coordinates is stressed, and k in all other cases.
The one-dimensional wave number spectrum, i.e. the spectrum as a function of the modulus of the wave number vector, is given by:

$$WS1(\rho) = 2 \cdot \int_{0}^{2\pi}\int_{0}^{\infty} \rho\, F(\rho, \phi, \omega)\, d\omega\, d\phi . \tag{7-10}$$

The total power as a function of the modulus of the wave number is given by:

$$PWS1(\rho) = 2 \cdot \int_{0}^{2\pi}\int_{0}^{\infty} \rho \left|F(\rho, \phi, \omega)\right|^2 d\omega\, d\phi . \tag{7-11}$$

Note that the integrations over ω in equations (7-7) through (7-11) run over the positive frequencies only in order to remove the $180°$ ambiguity. Therefore, the integrals are multiplied by a factor two. The discrete spectra and distributions are obtained by replacing the integrals in the previous equations with summations. In order to compute equations (7-9), (7-10) and (7-11), the data that is available on a Cartesian grid in the Fourier domain have to be resampled onto a polar grid. In fact this process is the reverse of the scan conversion process and again distortions are introduced in the polar Fourier spectra due to the interpolations.

7.3 Definition of parameters and generation of the input data field

This section defines and explains most of the parameters that are used in the remainder of the report. While doing this, the structure of the input data is discussed in detail and the generation of the simulation input data is explained.

As stated in the introduction to this report, the data acquisition system obtains the sample values on a nonuniform polar grid. While the radar antenna rotates, pulses are emitted and the corresponding pulse returns are sampled. The angle at which each pulse is emitted is recorded with each sweep. These angles and the distances from the radar at which samples are taken define the polar sampling grid (i.e. the data is collected in range-azimuth coordinates).

Because the angular velocity of the radar antenna is not constant, due to the wind for example, the sampling of the sweeps is non-equidistant in the azimuth (the sampling in the range direction however, is equidistant). The maximum deviation of the angular velocity of the radar antenna from its nominal value, and therefore the maximum deviation of the angles from the mean value, is denoted by A_d (in per cent). For every rotation of the antenna, an image on such a polar grid is obtained, thus forming a time series of images. The methods that are used for the spectral analysis can be compared by comparing only one image out of a time series, because the data is nonuniform in the two spatial dimensions only (it is assumed that the data is uniform in the time direction). Therefore, only individual images of a time series are considered and the m-file that simulates the input data field, called GenInpData (Appendix E), generates such a two-dimensional image in polar coordinates with nonuniformly distributed angles. This image is defined on a circular section indicated by the shaded area in Figure 7-8 and Figure 7-10. Since the generation of the input data is the starting-point of all simulations, GenInpData is called by the m-files that simulate the separate methods (see sections 7.4 and 8.5). The flow of actions that take place in GenInpData is depicted in Figure 7-7 and is exemplified in the remainder of this section.

GenInpData is called by the (main) simulation programs after another m-file, in which the input parameters for the spectral analysis are defined, has been executed. An example of such a file is given in Appendix D. The parameters that are defined in this file will be explained in detail because they are used in almost all m-files. Appendix C provides an overview of the definitions for quick reference.

The most important parameters are illustrated in Figure 7-8 and Figure 7-10. The parameter N_θ is the number of sweeps in the data window that is available (or selected) for analysis. N_a is the number of sweeps in one revolution of the radar antenna and it is equal to the rotation time of the radar antenna times the pulse repetition frequency (PRF): $N_a = T_R \cdot PRF$. Since the rotation time T_R is approximately constant, N_a is approximately constant. The mean or nominal spacing between the angles of the sweeps is $2\pi/N_a$. For the simulations, it is assumed that the angles in the data window are realizations of a random variable that has a uniform probability density function with mean $2\pi/N_a$ and a maximum deviation of A_d per cent from the mean (normally A_d is about 25 %). The probability density function (pdf) of the angles θ_{NU} is then given by:

$$f_{\Theta_{NU}}(\theta_{NU}) = \begin{cases} 0, & \theta_{NU} < (1 - A_d/100)2\pi/N_a \\ \dfrac{25N_a}{\pi A_d}, & (1 - A_d/100)2\pi/N_a < \theta_{NU} < (1 + A_d/100)2\pi/N_a \\ 0, & \theta_{NU} > (1 + A_d/100)2\pi/N_a . \end{cases} \tag{7-12}$$

GenInpData draws N_θ samples $\theta_{NU,i}$ with $i = 0, 1, ..., N_\theta - 1$ from this pdf in order to generate the nonuniformly spaced angles in the input data window. The start azimuth (angle) of the input data window if A_d is zero, i.e. if the angles were uniformly spaced, is denoted θ_0. More precisely, this is the nominal angle (no deviation) of the first sweep in the data window. The start azimuth of the *area* that is associated with the discrete sample points (the shaded area in Figure 7-8) is denoted $\theta_{\min}$ and is given by:

$$\theta_{\min} = \theta_0 - \frac{1}{2}\frac{2\pi}{N_a} = \theta_0 - \frac{\pi}{N_a}. \tag{7-13}$$

The available discrete sample points are indicated by circles in Figure 7-8.

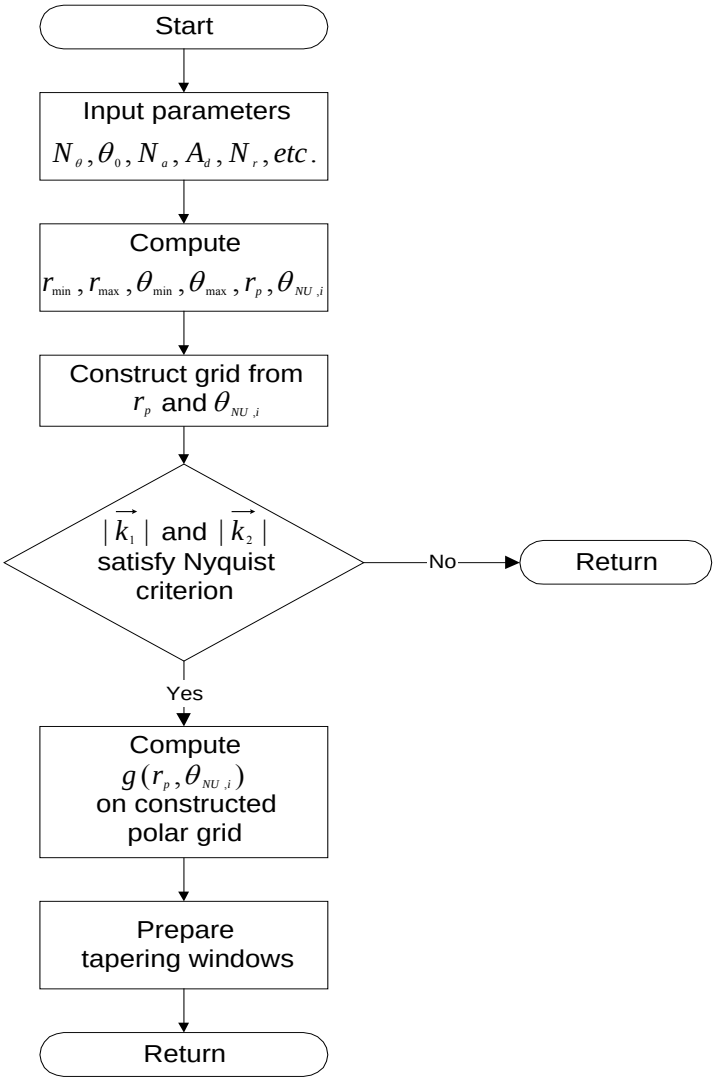

Figure 7-7 Flow diagram of `GenInpData` *(input field generation script file)*

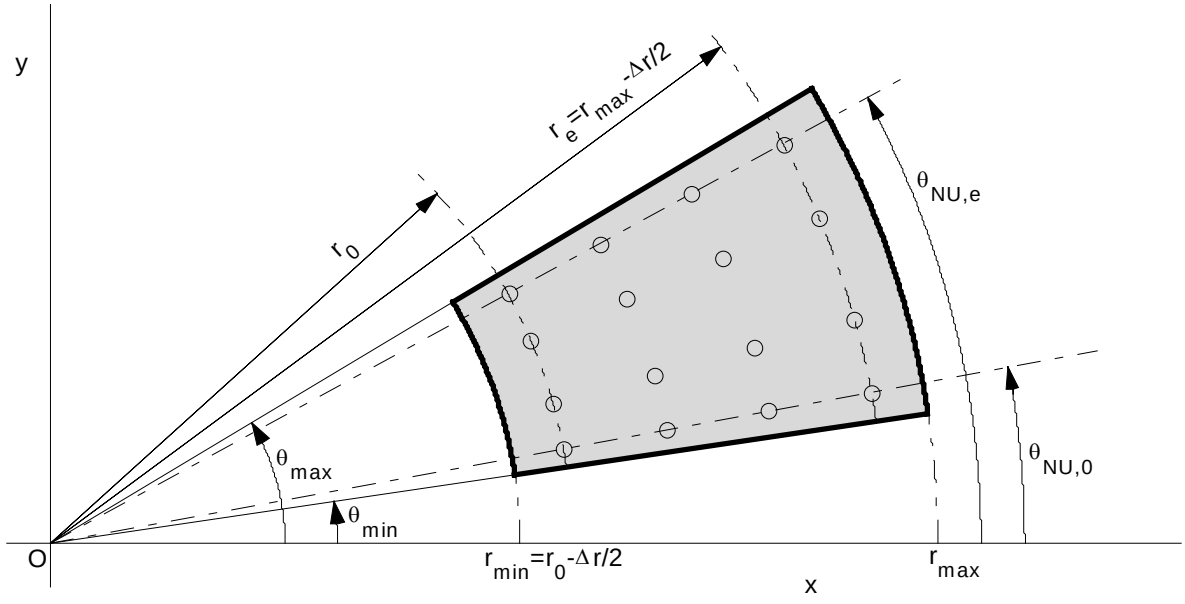

Figure 7-8 Shape and parameters of input data window for simulations

The angle of the first sweep in the data window $\theta_{NU,0}$ is the nominal angle of the first sweep (θ_0) plus its deviation from the nominal value ($\Delta\theta_{NU,0}$):

$$\theta_{NU,0} = \theta_0 + \Delta\theta_{NU,0} . \tag{7-14}$$

Likewise, the maximum azimuth of the shaded area in the figure that is associated with the discrete sample points is indicated by $\theta_{\max}$ and is given by:

$$\theta_{\max} = \theta_{\min} + N_\theta \frac{2\pi}{N_a} = \theta_0 + N_\theta \frac{2\pi}{N_a} - \frac{\pi}{N_a} . \tag{7-15}$$

The angle of the last sweep (that contains samples) in the data window $\theta_{NU,e}$ is the nominal angle of the last sweep θ_e plus its deviation from the nominal value ($\Delta\theta_{NU,e}$):

$$\theta_{NU,e} = \theta_{\max} - \frac{\pi}{N_a} + \Delta\theta_{NU,e} = \theta_0 + (N_\theta - 1)\frac{2\pi}{N_a} + \Delta\theta_{NU,e} . \tag{7-16}$$

The parameter N_r is the number of ranges in the data window, i.e. the number of samples per sweep. The Matlab program GenInpData computes the uniform ranges in the data window as:

$$r_p = r_0 + p\Delta r , \tag{7-17}$$

where $p = 0, 1, ..., N_r - 1$ and Δr is the sampling distance in the range direction. The start range of the input data window is denoted r_0. More precisely, this is the first range in the data window on which samples are located. The start range of the *area* that is associated with the discrete sample points is denoted $r_{\min}$ and is given by:

$$r_{\min} = r_0 - \tfrac{1}{2}\Delta r . \tag{7-18}$$

In the same way, the maximum or end range of the area that is associated with the discrete sample points in the data window is indicated by $r_{\max}$ and is given by:

$$r_{\max} = r_{\min} + N_r\Delta r = r_0 + N_r\Delta r - \tfrac{1}{2}\Delta r . \tag{7-19}$$

The last range in the data window on which samples are located is denoted r_e and is given by:

$$r_e = r_{\max} - \tfrac{1}{2}\Delta r = r_0 + (N_r - 1)\Delta r . \tag{7-20}$$

As can be seen in the flow diagram of Figure 7-7, the parameters $N_\theta, \theta_0, N_a, A_d, N_r, r_0$ and Δr are chosen by the user in the input parameters file (Appendix D) and almost all other parameters are derived from them. The figure also illustrates the computation of the formulae (7-12), (7-13), (7-15), (7-17), (7-18) and (7-19). In addition, it shows that from the nonuniformly spaced angles computed by equation (7-12) and uniformly spaced ranges computed by equation (7-17), a polar grid is constructed. The parameters that have just been explained resemble the parameters that are used in the first part of the report, which describes the data acquisition system, very much (see Appendix A). The only differences are that the latter ones apply to a recording window, i.e. the user-defined region in which data is acquired, while the former ones apply to an analysis window, which is not necessarily the same. The user can select a part of the recording window for analysis. This part is then called the analysis or data input window. So, the meaning of the parameters remains the same, but their values can differ. Two other parameters that are defined in the input parameters file are N_ρ and N_ϕ and they define the number of ranges in the polar Fourier plane (the absolute values of the wave numbers) and the number of angle bins into which the polar Fourier plane must be divided, respectively. These parameters are exemplified in more detail in Chapter 8, which deals with the Polar Fourier transform.

The circular section that forms the input data window has been defined now and the sea wave pattern can be simulated. Since propagating plane waves are a first approximation of the sea wave pattern and have *known* discrete frequency components, a snapshot of a plane wave is used as the input field. The input fields for most simulations will have either one or two sinusoidal component(s) and is defined by:

$$g(x, y) = \sin(k_{x1}x + k_{y1}y) + A_2 \sin(k_{x2}x + k_{y2}y) , \tag{7-21}$$

where $\vec{k}_1 = (k_{x1}, k_{y1})$ is the wave number (spatial frequency) of the first sine component with amplitude one and $\vec{k}_2 = (k_{x2}, k_{y2})$ the wave number of the second sine component with amplitude A_2 (if one component is desired, $A_2 = 0$). The amplitude spectrum of the Fourier transform contains a "sinc"-peak for each sinusoidal component. In order to compare the quality of the current method to the quality of the PFT, the amount of distortion in the spectra and some characteristics of the peak(s) are measured (see chapters 9 and 10). The spatial frequencies of the components must satisfy the Nyquist criterion because the "real-world" radar data also satisfy this criterion. This means that *the sampling distances in both the range and azimuth directions must be smaller than half the smallest wavelength in the input field*. The Nyquist criterion in the range direction is satisfied if the modulus of the wave number(s) $k = |\vec{k}|$ is chosen in such a way that $k < \pi/\Delta r$. The restriction on k that results from the Nyquist criterion in the azimuth direction can be derived as follows. The maximum (Euclidian) worst-case sampling distance in the azimuth direction is the distance between the points B and C in Figure 7-9.

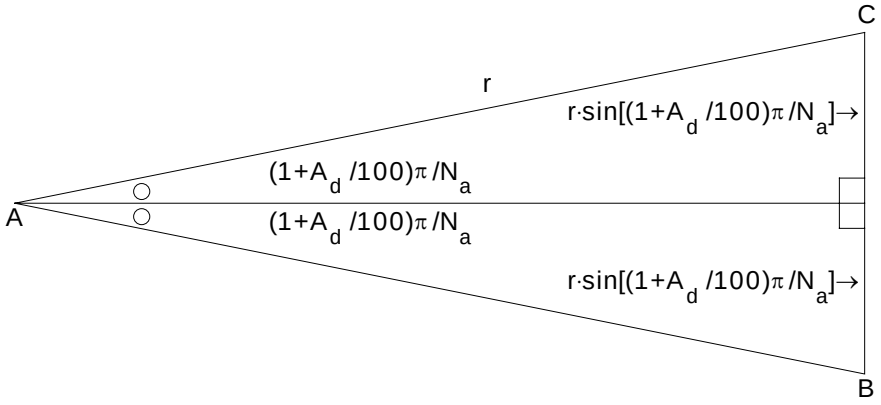

Figure 7-9 Determination of sampling distance in azimuth direction

The maximum angular spacing between sweeps is $(1 + A_d/100)2\pi/N_a$ (see formula (7-12)). So, half the length of the line BC is $r\sin[(1 + A_d/100)\pi/N_a]$ and the sampling distance in the azimuth direction is

$$d_S = 2r\sin[(1 + A_d/100)\pi/N_a]. \tag{7-22}$$

From the condition $d_S < \lambda_{\min}/2$ (which is equal to the condition $k < \pi/d_s$), where $\lambda_{\min}$ is the smallest wavelength in the input field, the following restriction for k is derived:

$$k < \frac{\pi}{2r_{\max}\sin[(1 + A_d/100)\pi/N_a]}. \tag{7-23}$$

Consequently, the wave numbers of the input field components must satisfy the relation:

$$k < \min\left\{\frac{\pi}{2r_{\max}\sin[(1 + A_d/100)\pi/N_a]}, \frac{\pi}{\Delta r}\right\}. \tag{7-24}$$

In order to obtain the representation of a plane wave field in polar coordinates, the substitutions $x = r\cos\theta$ and $y = r\sin\theta$ are made in the equation for the plane wave field.
The input field for discrete values of the ranges and the angles is then computed as:

$$g(r_p, \theta_{NU,i}) = \sin[k_{x1}r_p\cos(\theta_{NU,i}) + k_{y1}r_p\sin(\theta_{NU,i})] + A_2\sin[k_{x2}r_p\cos(\theta_{NU,i}) + k_{y2}r_p\sin(\theta_{NU,i})]. \tag{7-25}$$

Figure 7-8 and Figure 7-10 illustrate that only a finite area is selected for analysis. The shape of this analysis area will be referred to as "circular section". When analyzing such a restricted data window with the discrete Fourier transform, spectral leakage occurs. Therefore, tapering (windowing) has to be applied to the data window in order to reduce the spectral leakage (also see Section 8.5). Therefore, the final action of `GenInpData` is the generation of the tapering windows, chosen by the user in the input

parameters file (Appendix D), for both the range and azimuth directions. These windows will be used by the m-files that simulate the separate spectral analysis methods (see sections 7.4 and 8.5).

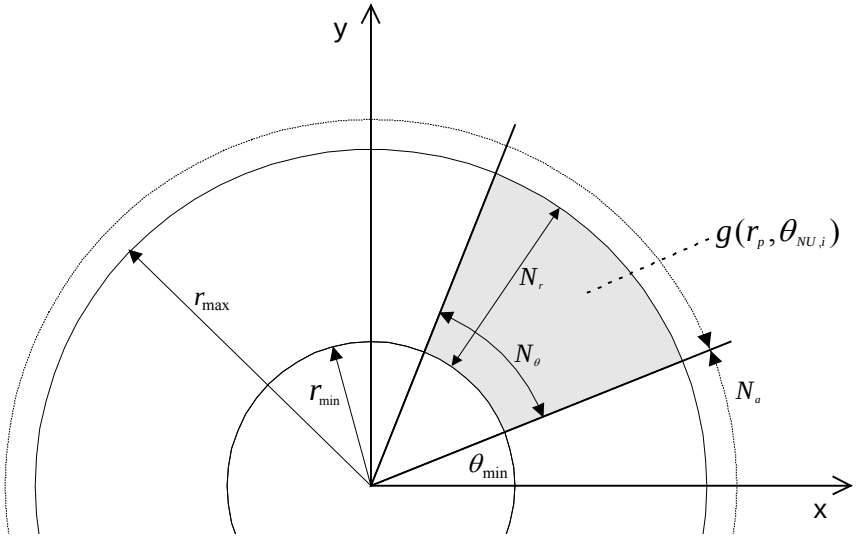

Figure 7-10 Circular section with input data

7.4 Simulation of currently used spectral analysis method

Now, after an overview has been given of the SHIRA signal processing stages, the simulation of the part that is relevant for this project, namely the currently used spectral analysis method of the raw radar data, is described. For this purpose, the file GenInpData that was discussed in the previous section is needed. The computation of the spectra of the raw data by means of the currently used method is simulated by the m-file CurrMeth that is listed in Appendix F. CurrMeth starts by calling an m-file in which the input parameters for the simulation are defined. This m-file looks somewhat different from the one in Appendix D. Instead of defining N_θ, θ_0, N_r and r_0, which are the parameters of a circular section, N_x, N_y, r_x, r_y, P_{cx} and P_{cy} are defined (see Section 7.2.3 and Appendix C). After that, the sizes of the rectangle in both directions are computed. The size in the x-direction is $s_x = N_x r_x$ and in the y-direction $s_y = N_y r_y$. Because the current method, like all other methods, starts with polar input data, a circular section that encompasses the rectangle is used as the recording window (see Figure 7-11) that provides the required data.

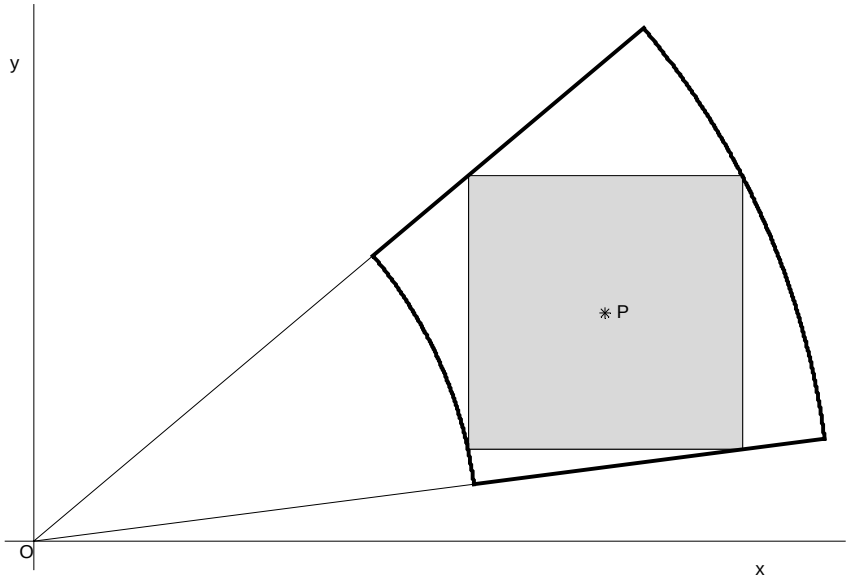

Figure 7-11 Rectangle for current method with encompassing circular section

The parameters N_θ, θ_0, N_r and r_0 are computed in such a way that the circular section encompasses the selected rectangle. As can be seen in Figure 7-11, a lot of data is wasted. In order to use the available data as efficiently as possible, the rectangle must be positioned in such a way that it coincides with the available input data circular section as much as possible. This is done when the currently used (conventional) method is compared to the PFT (see Section 10.2 and Figure 10-1).
After the parameters have been defined or computed, CurrMeth calls GenInpData, which generates an input field on the circular section that encompasses the rectangle. Next, the uniform rectangular coordinates are expressed in polar coordinates and the data sample points in the circular section are interpolated onto the sample points of the rectangle (thus, the interpolation is performed in the polar domain). The interpolation method to be used can be chosen by the user in the input parameters file (for an example, see Appendix D).
Finally, Appendix F shows that after the data is windowed with the windows defined in the input parameters file, the two-dimensional FFT is computed.
Figure 7-12 and Figure 7-13 show examples of spectra computed this way with parameters $N_x = 64$, $N_y = 64$, $r_x = 3.75\,m$, $r_y = 3.75\,m$, $P_{cx} = 400\,m$, $P_{cy} = 340\,m$, $N_a = 1024$, $A_a = 25$, $A_2 = 0$ and $k_{x1} = k_{y1} = 0.4073\,m^{-1}$. The first figure shows the spectrum when NN interpolation is used for the two-dimensional interpolation step preceding the FFT and the second when bi-linear interpolation is used. The location of the peaks is computed correctly for both interpolation methods. However, the figures clearly illustrate that the spectrum computed with the Nearest Neighborhood interpolation method has more distortion than the spectrum computed with the bi-linear method, as has been explained in Section 7.2.2.

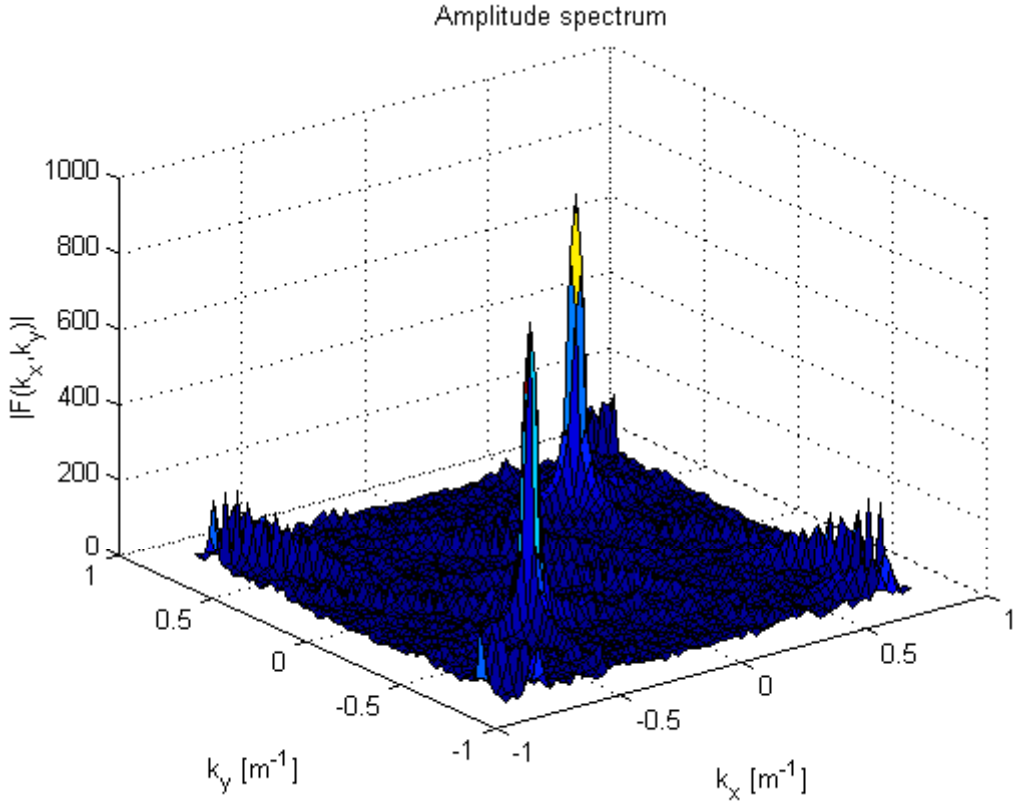

Figure 7-12 Example of 2D spectrum computed with current method (NN interpolation)

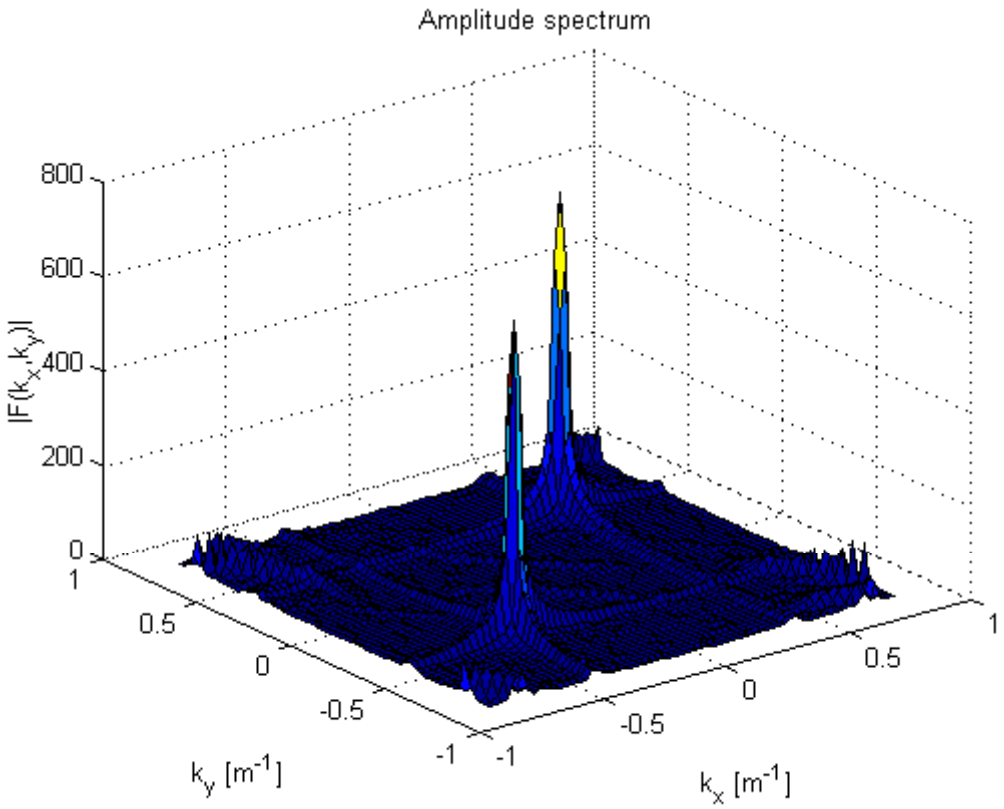

Figure 7-13 Example of 2D spectrum computed with current method (bi-linear interpolation)

7.5 Complexity of conventional method

The mapping (interpolation) of the polar data to Cartesian data is the most computation-intensive operation. On page 295 of [Briggs and Henson, 1995] it is claimed that for most interpolation schemes, $O(1)$ operations are required for each of the $N_x N_y$ points to be interpolated, hence the cost of the interpolation is $O(N_x N_y)$. It should be noted, however, that some of the extremely accurate interpolations based on the Shannon Sampling Theorem incur costs approaching $O(N^3)$ (in case $N_x = N_y$). The complexity of the scan conversion is

$$O\{scan\ conversion\} = C_1 \cdot O(N_x N_y), \tag{7-26}$$

where the constant C_1 depends on the interpolation method. The cost of computing the two-dimensional FFT is the cost of N_x one-dimensional FFTs of length N_y plus the cost of N_y one-dimensional FFTs of length N_x:

$$O\{2D\ FFT\} = C_2[N_x \cdot O\{N_y \log_2(N_y)\} + N_y \cdot O\{N_x \log_2(N_x)\}] = C_2 \cdot O\{N_x N_y \log_2(N_x N_y)\}, \tag{7-27}$$

where the constant C_2 depends on the specific FFT algorithm used. So, the total complexity of the currently used spectral analysis method is:

$$O\{current\ spectral\ analysis\ method\} = C_1 \cdot O\{N_x N_y\} + C_2 \cdot O\{N_x N_y \log_2(N_x N_y)\}. \tag{7-28}$$

For most interpolation schemes and values of N_x and N_y, the constant C_1 is much larger than $C_2 \log_2(N_x N_y)$ and therefore it is the most computation-intensive operation. As stated in Section 7.1, within the scope of this report, the computational complexity is of less importance. However, an attempt will be made to find a way that is at least as fast as the currently used method.

7.6 Possible improvements and alternatives

In order to obtain better spectra, the following options might be considered. Since the data is available on a polar grid, using the polar-grid samples directly intuitively seems to exploit the given information better. This way it should be possible to take into account the fact that the data is equidistant in the range direction. In addition, as will be pointed out, interpolation is only necessary in the azimuth direction so as to obtain the data on a uniformly sampled polar grid. Moreover, use can be made of the fact that if the angular velocity is not constant due to the wind for example, the angles are sampled with a high density at one side of the antenna axis and with a low density at the other. The interpolation at one side can then be coarser than at the other side. All these improvements or alternatives can be used when a Fourier transform that takes polar input data is used. For this reason, the PFT has been chosen to be further investigated.

Another obvious improvement is the use of better interpolation methods like linear, Lagrange, cubic spline, polynomial [Zakhor and Alvstad, 1992] interpolation etc. instead of Nearest Neighborhood interpolation. This implies, however, that *computation time* must be compromised in favor of the quality of the interpolation (and therefore of the spectra). Also, interpolation by means of the FFT [Fraser, 1989] may be used. An alternative for the computation of the spectra is to treat the polar grid as an "irregular Cartesian" grid and to employ methods for computing FFTs on irregular (nonuniform) grids. Such methods are fairly recent additions to the FFT family. Information on fast algorithms for the nonuniform Fourier transform (NUFFT) can be found for example in [Nguyen and Liu, 1999], [Ware, 1998] and [Bland, Laakso and Tarczynski]. An exhaustive treatment can be found in [Dutt and Rokhlin, 1993]. Finally, an improvement regarding the more efficient usage of the data is to let the analysis rectangle coincide with the circular section, which contains the input data, as much as possible (Section 7.4).

8 The discrete polar Fourier transform (DPFT)

8.1 Introduction

Since the data is available on a polar grid, it intuitively seems better to directly use the polar input data instead of first converting them to Cartesian data, because this way errors in the interpolation step are avoided as much as possible. Therefore, as has been stated in the previous chapter, the PFT is investigated in more detail. Moreover, in order to compute the spectra defined by equations (7-9), (7-10) and (7-11), the data must be available on a polar grid in the Fourier domain. With the current method, the data is available on a Cartesian grid and must therefore be resampled onto a polar grid. Thus, again distortions are introduced in the polar Fourier spectra due to the interpolations, and these distortions propagate to the spectra computed by the mentioned equations. For this reason, it is desired that the Fourier data is directly available on a polar grid.
Therefore, the scan conversion step is eliminated and the two-dimensional Cartesian Fourier transformation of each image in a time series is replaced by the polar Fourier transform (PFT). In Section 8.2, this PFT is defined as the Fourier transform (FT) that takes polar input data and produces polar output data. In that section, first a formula for the continuous PFT is derived from the two-dimensional Cartesian FT. Then, this formula is discretized in order to obtain a formula that can be used to compute the desired spectra from discrete (digitized) polar input data. This formula can be computed directly, but it can also be computed indirectly via Hankel transforms. This method seems promising because methods for the fast evaluation of Hankel transforms are reported in the literature. Therefore, the computation of the (D)PFT via Hankel transforms is described in Section 8.3. Also, some optimizations, which can be deduced directly from the equations and that will be used for the simulations, are described. As has been stated in Section 7.1, the computational complexity of the DPFT is not of primary concern within the scope of this report. Therefore, and also due to the limited amount of time, only an overview of algorithms for the (fast) numerical evaluation of Hankel transforms found in the literature is given in Section 8.4. The separate classes of algorithms and their applicability to the SHIRA application are discussed. In addition, references are given for the interested reader. Finally, Section 8.5 explains the application of the method described in Section 8.3 to SHIRA, and its simulation by means of the m-file PolFFT.

8.2 The polar Fourier transform

For the derivation of the formulae, it is assumed that the input function is known over the entire spatial plane. Later on, when the transform is applied to finite input data, the consequences of a finite data set will be discussed.
The continuous two-dimensional Cartesian FT of each image in a time series is given by:

$$F(k_x,k_y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\exp[-j(k_x x+k_y y)]\,dx\,dy\,. \tag{8-1}$$

By making the substitutions

$$x=r\cos\theta,\ y=r\sin\theta,\ k_x=\rho\sin\phi \text{ and } k_y=\rho\cos\phi, \tag{8-2}$$

the formula for the continuous PFT can be written as:

$$F(\rho,\phi)=\int_{0}^{\infty}\int_{0}^{2\pi} r\,f(r,\theta)\exp[-jr\rho\sin(\theta+\phi)]\,dr\,d\theta\,. \tag{8-3}$$

Because the input field is chosen in such a way that *the Nyquist criterion is satisfied in both the azimuth and range directions* (see Section 7.3), no aliasing is present and it is reasonable that the integrals may be replaced by summations, resulting in the following discrete version of equation (8-3), the discrete Polar Fourier transform (DPFT):

$$F(\rho_l,\phi_m)=\sum_{p=0}^{N_r-1}\sum_{i=0}^{N_a-1} r_p\,f(r_p,\theta_i)\exp[-jr_p\rho_l\sin(\theta_i+\phi_m)]\,. \tag{8-4}$$

In this equation, the discrete variables are defined as follows:

- $r_p = r_0 + p\Delta r$ with $p = 0, 1, ..., N_r - 1$ are the ranges at which the data is available
- $\theta_i = i(2\pi/N_a)$ with $i = 0, 1, ..., N_a - 1$ are the angles of the sweeps at which the data is available
- $\rho_l = l\,\Delta\rho$ with $l = 0, 1, ..., N_\rho - 1$ are the absolute values of the wave numbers for which the spectrum values are computed
- $\phi_m = m\,\Delta\phi$ with $m = 0, 1, ..., N_\phi - 1$ are the angles in the (polar) Fourier domain for which the spectrum values are computed.

Equation (8-4) is an estimation of the Fourier spectrum of data available on a polar grid and also holds for nonuniformly spaced ranges and angles (given that they are spaced in such a way that the Nyquist criterion is still satisfied). Care must be taken while replacing the integrals with summations. Since the situation is not completely analogous to the Cartesian FT because of the different coordinate systems, the validity of the replacements should be proven explicitly for this case (this is not done in the current project). [Briggs and Henson, 1995] deals with the DFT approximation of the Fourier transform and the errors of the approximation. Similar techniques will have to be used to prove that equation (8-3) may be approximated by equation (8-4) in case the Nyquist criterion is satisfied in both the range and azimuth directions. However, the results of simulations show that it is highly probable that the replacements are valid, as can be seen in Figure 8-2, which gives an example of the DPFT of a 2D image out of a time series. The parameters for the input circular section of this DPFT computation are chosen in such a way that approximately the same area is analyzed as in the example of Section 7.4. The location of the peak and its form are very similar to that in Figure 7-12. The m-file PolFftD that simulates the DPFT by directly computing it is listed in Appendix G, while a flow chart is depicted in Figure 8-1. PolFftD starts by calling an m-file (see Appendix D) in which the input parameters for the simulation are defined (box 2 in Figure 8-1). After that, it calls GenInpData that generates an input field on the circular section defined by the input parameters (box 3). For the simulations, formula (8-4) is used with uniformly spaced angles; the reason for this will become clear in the next section and in Section 8.5. Therefore, after calling GenInpData, PolFftD first interpolates the nonuniform data in the azimuth direction onto uniformly spaced sweeps in order to make the data available on a uniform polar grid (boxes 4 and 5). Because of the finite data window lengths in both the range and azimuth directions spectral leakage arises. Tapering windows are used to reduce this spectral leakage (box 6). Section 8.5 discusses this subject in more detail. Finally equation (8-4) is computed directly (box 7) and PolFftD returns.

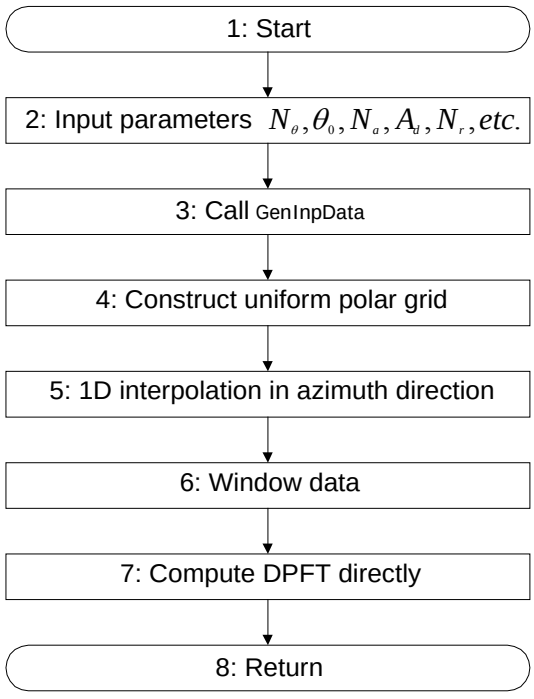

Figure 8-1 Flow diagram of PolFftD (direct computation of DPFT)

An example of a spectrum computed this way is shown in Figure 8-2. The y-component of the wave number (k_y) is at the positive "x-axis" and the x-component of the wave number (k_x) is at the positive "y-axis" because $k_x = \rho \sin\phi$ and $k_y = \rho \cos\phi$ (equation (8-2)). However, by mirroring k_x and k_y with respect to the line $k_x = k_y$, a "normal" coordinate system can be obtained.
The direct computation of the equation for all images in a time series takes a lot of time. Although the efficient computation of the spectra is not a primary issue, it is desired to try to compute them as fast as possible (at least for the simulations). Therefore, a method for indirectly computing the (D)PFT via Hankel transforms is investigated in Section 8.3. It has also been tried to compute equation (8-4) in a way similar to FFT algorithms. However, in agreement with other authors that claim that no FFT for the PFT is available [Briggs and Henson, 1995], it must be concluded that *no FFT like algorithm to compute the PFT currently exists or is likely to be invented in the future*. The properties of the kernel in the Fourier transform that are necessary for the development of the FFT are not found in the kernel of the PFT due to the sine term in the imaginary exponent of the kernel.

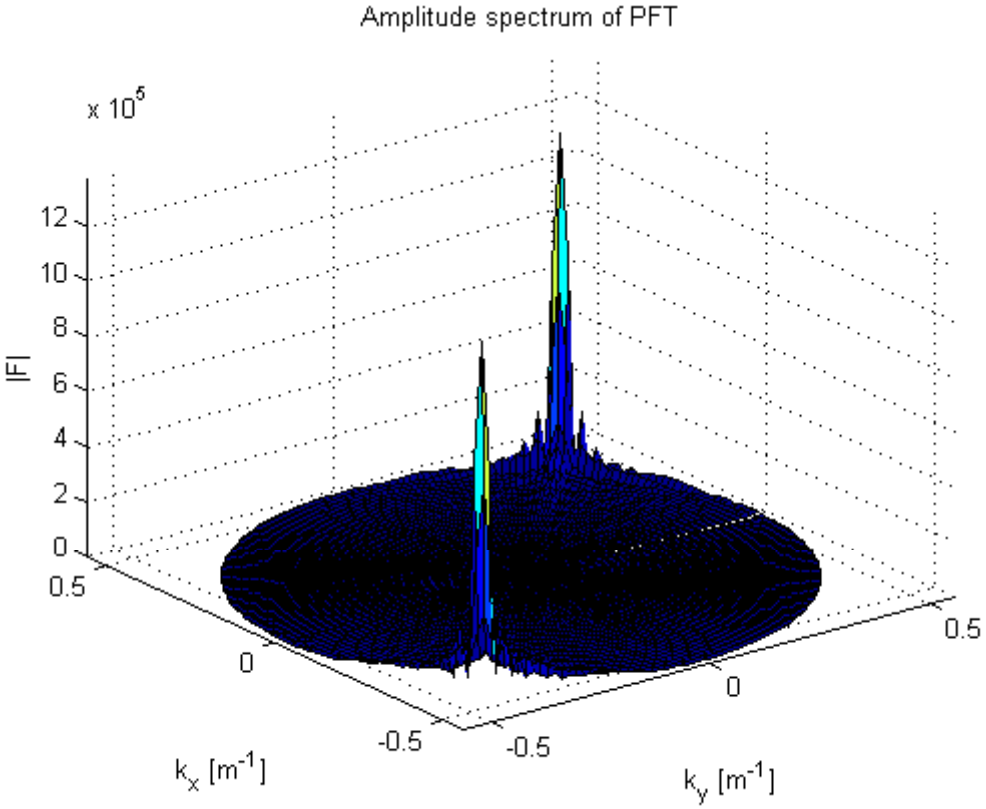

Figure 8-2 Example of a spectrum computed with the DPFT

8.3 Computing the (D)PFT via Hankel transforms

The approach followed in this section is the same as in the previous: first the derivation is given for continuous signals ([Stark, 1979], [Higgings and Munson, 1988]) and next the discrete version is derived in an analogous way. The starting point is the continuous PFT given in equation (8-3):

$$F(\rho,\phi) = \int_0^{\infty}\int_0^{2\pi} r\, f(r,\theta) \exp[-jr\rho\sin(\theta+\phi)]\, dr\, d\theta\,. \tag{8-5}$$

Since $f(r,\theta)$ is periodic in θ with period 2π, i.e.,

$$f(r,\theta) = f(r,\theta+2\pi s) \qquad s = \pm 1, \pm 2, \ldots \tag{8-6}$$

it may be written as a Fourier series in θ:

$$f(r,\theta) = \sum_{n=-\infty}^{\infty} c_n(r) \exp(jn\theta)\,, \tag{8-7}$$

where the coefficients

$$c_n(r) = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(r,\theta)\exp(-jn\theta)\,d\theta \tag{8-8}$$

are referred to as *circular harmonics* of $f(r,\theta)$.

By substituting equation (8-7) in (8-5) and changing the order of integration and summation, the following expression is obtained:

$$\begin{aligned} F(\rho,\phi) &= \sum_{n=-\infty}^{\infty}\int_0^{\infty} dr\; r c_n(r)\int_0^{2\pi}\exp[\,j(n\theta - r\rho\sin(\theta+\phi))]d\theta \quad \rightarrow \{\theta' = \theta+\phi\} \rightarrow \\ &= \sum_{n=-\infty}^{\infty}\exp(-jn\phi)\int_0^{\infty} dr\; r c_n(r)\int_{\phi}^{2\pi+\phi}\exp[\,j(n\theta - r\rho\sin(\theta))]d\theta. \end{aligned} \tag{8-9}$$

Using the identity for the n-th order Bessel function of the first kind (page 20 of [Watson, 1966])

$$J_n(x) = \frac{1}{2\pi}\int_0^{2\pi}\exp[\,j(n\theta - x\sin(\theta))]d\theta = \frac{1}{2\pi}\int_{\phi}^{2\pi+\phi}\exp[\,j(n\theta - x\sin(\theta))]d\theta, \tag{8-10}$$

it follows that $F(\rho,\phi)$ can be written as

$$F(\rho,\phi) = 2\pi\sum_{n=-\infty}^{\infty}\exp(-jn\phi)\int_0^{\infty} r\,c_n(r)J_n(\rho r)dr. \tag{8-11}$$

The n-th order Hankel transform of a function $g(r)$ is by definition *[Sneddon, 1955]*:

$$\overline{g_n(\rho)} = \mathrm{H}_n\{g(r)\} = \int_0^{\infty} r\,g(r)J_n(\rho r)dr. \tag{8-12}$$

Hence

$$F(\rho,\phi) = 2\pi\sum_{n=-\infty}^{\infty}\overline{c_{nn}(\rho)}\exp(-jn\phi), \tag{8-13}$$

where

$$\overline{c_{nn}(\rho)} = \mathrm{H}_n\{c_n(r)\} = \int_0^{\infty} r c_n(r)J_n(\rho r)dr. \tag{8-14}$$

Equation (8-13) is the polar Fourier series expansion of the Fourier transform of $f(r,\theta)$ with coefficients $\overline{c_{nn}(\rho)}$ (circular harmonics), which are the n-th order Hankel transforms of the circular harmonics of $f(r,\theta)$.

In order to discretize these equations, equation (8-4) is taken as a starting point (for the definition of r_p, θ_i, ρ_l and ϕ_m, see Section 8.2):

$$F(\rho_l,\phi_m) = \sum_{p=0}^{N_r-1}\sum_{i=0}^{N_a-1} r_p\, f(r_p,\theta_i)\exp[-jr_p\rho_l\sin(\theta_i+\phi_m)]. \tag{8-15}$$

If the angles θ_i are ***uniformly spaced***, i.e. $\theta_i = i(2\pi/N_a)$ with $i = 0, 1, ..., N_a - 1$,
then $f(r_p,\theta_i) = f[p,i]$ is periodic in i with period N_a, i.e.,

$$f(r_p,\theta_i) = f[p,i] = f[p,i+sN_a] = f(r_p,\theta_i+2\pi s) \qquad s = \pm1, \pm2, ... \tag{8-16}$$

In practice however, the angles are not uniformly spaced as has been stated earlier. So, in order to obtain the data on uniformly spaced angles (and thus on a uniform polar grid), a one-dimensional interpolation in the azimuth direction is required. Then, if $f(r_p,\theta_i)$ is known on a uniform polar grid, it may be written as the Inverse Discrete Fourier Transform (IDFT) of the circular harmonic coefficients $c_n(r_p)$:

$$f(r_p,\theta_i) = \frac{1}{N_a}\sum_{n=0}^{N_a-1} c_n(r_p)\exp(jn\theta_i), \tag{8-17}$$

where the sequence

$$c_n(r_p) = \sum_{i=0}^{N_a-1} f(r_p,\theta_i)\exp(-jn\theta_i) \tag{8-18}$$

is the DFT of $f(r_p,\theta_i)$. The coefficients $c_n(r_p)$ are the DFT coefficients of the function $f(r_p,\theta_i)$ for constant radii r_p (which can be computed fast). By using equation (8-17) in (8-15) and changing the order of the summations, the following formula is obtained:

$$\begin{aligned} F(\rho_l,\phi_m) &= \frac{1}{N_a}\sum_{n=0}^{N_a-1}\sum_{p=0}^{N_r-1} r_p c_n(r_p)\sum_{i=0}^{N_a-1}\exp[j(n\theta_i - r_p\rho_l\sin(\theta_i+\phi_m))] \quad \rightarrow \{\theta_i' = \theta_i+\phi_m\} \rightarrow \\ &= \sum_{n=0}^{N_a-1}\exp(-jn\phi_m)\sum_{p=0}^{N_r-1} r_p c_n(r_p)\frac{1}{N_a}\sum_{i=0}^{N_a-1}\exp[j(n\theta_i - r_p\rho_l\sin(\theta_i)]. \end{aligned} \tag{8-19}$$

Now, if $J_n^d(x)$ (called here the n-th order discrete Bessel function of the first kind) is defined as:

$$J_n^d(x) = \frac{1}{N_a}\sum_{i=0}^{N_a-1}\exp[j(n\theta_i - x\sin(\theta_i)], \tag{8-20}$$

it follows that $F(\rho_l,\phi_m)$ can be written as

$$F(\rho_l,\phi_m) = \sum_{n=0}^{N_a-1}\exp(-jn\phi_m)\sum_{p=0}^{N_r-1} r_p c_n(r_p) J_n^d(r_p\rho_l). \tag{8-21}$$

If the n-th order Discrete Hankel transform (DHT) of a function $g(r_p)$ is defined (here) as:

$$\overline{g_n^d(\rho_l)} = \mathrm{H}_n^d\{g(r_p)\} = \sum_{p=0}^{N_r-1} r_p f(r_p) J_n^d(r_p\rho_l) \tag{8-22}$$

the following result is obtained:

$$F(\rho_l,\phi_m) = \sum_{n=0}^{N_a-1}\overline{c_{nn}^d(\rho_l)}\exp(-jn\phi_m), \tag{8-23}$$

where

$$\overline{c_{nn}^d(\rho_l)} = \mathrm{H}_n^d\{c_n(r_p)\} = \sum_{p=0}^{N_r-1} r_p c_n(r_p) J_n^d(r_p\rho_l). \tag{8-24}$$

Equation (8-23) is the DFT representation of the DPFT of $f(r_p,\theta_i)$ with coefficients $\overline{c_{nn}^d(\rho_l)}$ (circular harmonics), which are the n-th order discrete Hankel transforms of the circular harmonics of $f(r_p,\theta_i)$ (by definition). It can be computed by means of an FFT algorithm if $N_\phi = N_a$. The sequence of computations needed to arrive at equation (8-23) is illustrated in Figure 8-3.

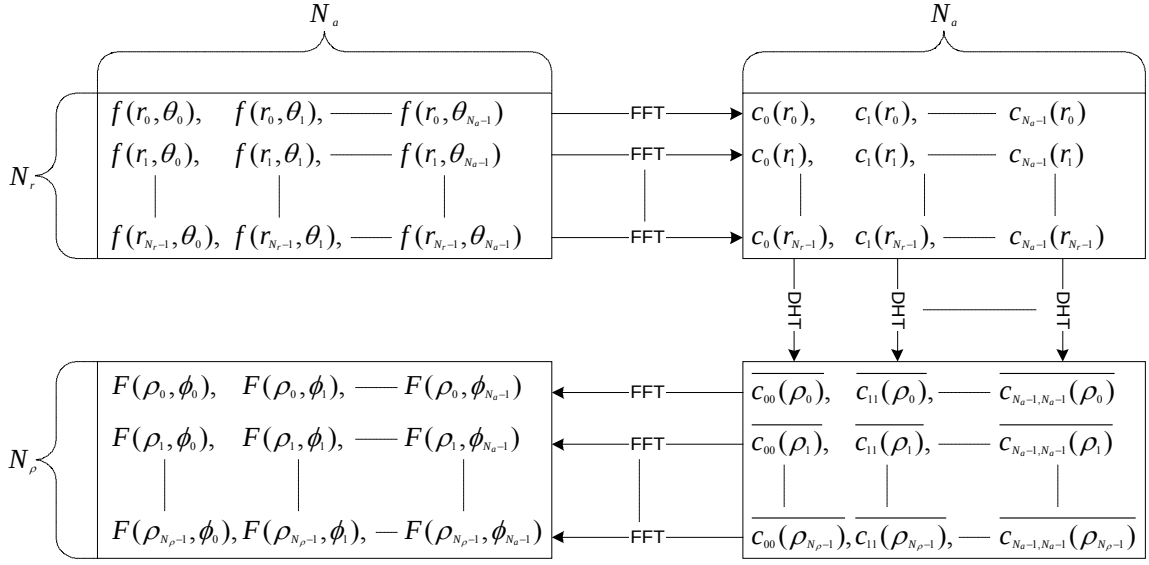

Figure 8-3 Computation of the DPFT of uniform polar data via the Hankel transforms

First the FFTs of the data on each radius are computed (equation (8-18)). Next the discrete Hankel transforms (equation (8-24)) are computed. As can be seen in the figure, each element of a column of

the bottom right box is computed by a DHT of the corresponding column in the top right box. Finally the estimated Fourier spectrum in polar coordinates, the DPFT, is obtained by computing the FFTs of the rows of the bottom right box (equation (8-23)).
Simulations (see Section 8.5) show that even the computation of the DPFT via Hankel transforms by directly computing the above derived equations, is already faster than computing equation (8-4) directly. However the direct evaluation of equation (8-20) still incurs a heavy computational load. In order to compute the combination of equations (8-20) and (8-24) rather fast, fast (discrete) Hankel transforms are required. Therefore, a literature search towards fast Hankel transforms is performed and an overview of the available classes of algorithms is given in Section 8.4.
Since no time is available to implement fast Hankel transform algorithms, it is tried to optimize the above equations as much as possible in order to be able to perform the simulations for the comparison of the DPFT to the current method (see Chapter 10) in an acceptable amount of time. It must be noted that in spite of the fact that the size of the analysis window in the azimuth direction is N_θ, the summation over n in equation (8-21) runs over N_a values. It is impossible to make the summation over n run over N_θ values instead of N_a. This can be explained as follows. First it is noted that the summation over i in equation (8-15) runs over N_θ values instead of N_a because the data is padded with zeros on the circular section outside the shaded area in Figure 7-10. Thus, if it is assumed that the summation over i starts with $i = 0$, equation (8-15) can be written as

$$F(\rho_l, \phi_m) = \sum_{p=0}^{N_r-1} \sum_{i=0}^{N_\theta-1} r_p \, f(r_p, \theta_i) \exp[-j r_p \rho_l \sin(\theta_i + \phi_m)]. \tag{8-25}$$

Now, for the N_θ values of i where $f(r_p, \theta_i)$ is given, $f(r_p, \theta_i)$ may be written as:

$$f(r_p, \theta_i) = \frac{1}{N_\theta} \sum_{n=0}^{N_\theta-1} c_n(r_p) \exp(jni2\pi / N_\theta), \tag{8-26}$$

where

$$c_n(r_p) = \sum_{i=0}^{N_\theta-1} f(r_p, \theta_i) \exp(-jni2\pi / N_\theta). \tag{8-27}$$

By substituting equation (8-27) in (8-25) and changing the order of the summations, the following formula is obtained:

$$\begin{aligned} F(\rho_l, \phi_m) &= \frac{1}{N_\theta} \sum_{n=0}^{N_\theta-1} \sum_{p=0}^{N_r-1} r_p c_n(r_p) \sum_{i=0}^{N_\theta-1} \exp[j(ni2\pi / N_\theta - r_p \rho_l \sin(\theta_i + \phi_m))] \quad \rightarrow \{\theta_i' = \theta_i + \phi_m\} \rightarrow \\ &\neq \frac{1}{N_\theta} \sum_{n=0}^{N_\theta-1} \exp(-jn\phi_m) \sum_{p=0}^{N_r-1} r_p c_n(r_p) \sum_{i=0}^{N_\theta-1} \exp[j(ni2\pi / N_\theta - r_p \rho_l \sin(\theta_i)]. \end{aligned} \tag{8-28}$$

So, because of the fact that the summation

$$\sum_{i=0}^{N_\theta-1} \exp[j(ni2\pi / N_\theta - r_p \rho_l \sin(\theta_i + \phi_m))] = \sum_{i=0}^{N_\theta-1} \exp[j(ni2\pi / N_\theta - r_p \rho_l \sin(2\pi i / N_a + \phi_m))]$$

is not periodic in i with period N_θ, $F(\rho_l, \phi_m)$ cannot be written as a Fourier series in ϕ_m and it must be concluded that it is impossible to let all summations in the above equations run over N_θ values instead of N_a values. However, some of the above equations defining the computation of the DPFT via Hankel transforms can be optimized by making use of symmetries. First of all, the discrete Bessel coefficients can be computed more efficiently. It can be shown that the imaginary part of equation (8-20) is zero:

$$\begin{aligned}
\mathrm{Im}\{J_n^d(x)\} &= \mathrm{Im}\left\{\sum_{i=0}^{N_a-1}\exp[\,j(n\theta_i - x\sin(\theta_i)]\right\} = \sum_{i=0}^{N_a-1}\sin[n\theta_i - x\sin(\theta_i)] \\
&= \sum_{i=0}^{N_a/2-1}\sin[n\theta_i - x\sin(\theta_i)] + \sum_{i=N_a/2}^{N_a-1}\sin[n\theta_i - x\sin(\theta_i)] \\
&= \{p(i) := \sin[n\theta_i - x\sin(\theta_i)]\} \\
&= \sum_{i=0}^{N_a/2-1} p(i) + \sum_{i=N_a/2}^{N_a-1} p(i) = \{i' = N_a - i\} = \sum_{i=0}^{N_a/2-1} p(i) + \sum_{i=N_a/2}^{1} p(-i) \\
&= \sum_{i=0}^{N_a/2-1} p(i) - \sum_{i=1}^{N_a/2} p(i) = p(0) - p(N_a/2) = 0.
\end{aligned} \tag{8-29}$$

Hence, equation (8-20) may be written as

$$J_n^d(x) = \sum_{i=0}^{N_a-1}\cos[n\theta_i - x\sin(\theta_i)]. \tag{8-30}$$

So, a factor two has been gained for the evaluation of equation (8-20). A further optimization can be obtained by bisecting the summation over i. In order to do this, a similar approach as in equation (8-29) is taken:

$$\begin{aligned}
J_n^d(x) &= \sum_{i=0}^{N_a-1}\cos[n\theta_i - x\sin(\theta_i)] \\
&= \sum_{i=0}^{N_a/2-1}\cos[n\theta_i - x\sin(\theta_i)] + \sum_{i=N_a/2}^{N_a-1}\cos[n\theta_i - x\sin(\theta_i)] \\
&= \{q(i) := \cos[n\theta_i - x\sin(\theta_i)]\} \\
&= \sum_{i=0}^{N_a/2-1} q(i) + \sum_{i=N_a/2}^{N_a-1} q(i) = \{i' = N_a - i\} = \sum_{i=0}^{N_a/2-1} q(i) + \sum_{i=N_a/2}^{1} q(-i) \\
&= \sum_{i=0}^{N_a/2-1} q(i) + \sum_{i=1}^{N_a/2} q(-i) = \{q(i) = q(-i)\} = 2\cdot\sum_{i=0}^{N_a/2-1} q(i) + q(N_a/2) - q(0) \\
&= 2\cdot\sum_{i=0}^{N_a/2-1} q(i) + q(N_a/2) - q(0) = 2\cdot\sum_{i=0}^{N_a/2-1} q(i) + (-1)^n - 1.
\end{aligned} \tag{8-31}$$

Thus, equation (8-20) may be written as

$$J_n^d(x) = 2\cdot\sum_{i=0}^{N_a/2-1}\cos[n\theta_i - x\sin(\theta_i)] + (-1)^n - 1. \tag{8-32}$$

The summation in this equation runs over $N_a/2$ values instead of N_a values, while the expression to be summed up is the same. Only a constant depending on n has to be added. This means that again approximately a factor two has been gained for the evaluation of equation (8-20). The same procedure can be carried out once again in order to bisect the summation interval another time. However, since in this case the expression to be summed up turns out to be more complicated, the computation time increases again.

Another advantage can be obtained by making use of the symmetry in the following equation (equation (8-24)):

$$\overline{c_{nn}^d(\rho_l)} = \mathrm{H}_n^d\{c_n(r_p)\} = \sum_{p=0}^{N_r-1} r_p c_n(r_p) J_n^d(r_p\rho_l). \tag{8-33}$$

The direct evaluation of equation (8-23) requires that the coefficients $\overline{c_{nn}^d(\rho_l)}$ are computed up to order N_a and consequently the coefficients $J_n^d(r_p\rho_l)$ have to be computed up to order N_a. However, by making use of the symmetry in the FFT coefficients $c_n(r_p)$ and in the discrete Bessel coefficients $J_n^d(x)$, only one half of the coefficients $\overline{c_{nn}^d(\rho_l)}$ has to be computed by means of equation (8-33) and the other half can easily be deduced by a simple relation that can be derived as follows. From equation (8-30) it can be seen that

$$J_{N_a-n}^d(x) = J_{-n}^d(x). \tag{8-34}$$

The same relation holds for the FFT coefficients:

$$c_{N_a-n}(r_p) = c_{-n}(r_p). \tag{8-35}$$

In addition, because $f(r_p,\theta_i)$ is real, the FFT coefficients are conjugate-symmetric:

$$c_{-n}(r_p) = c_n^*(r_p)\,. \tag{8-36}$$

For the discrete Bessel coefficients the following relation holds (this relation also holds for continuous Bessel functions as can be found on page 16 of [Watson, 1966]):

$$J_{-n}^d(x) = (-1)^n J_n^d(x)\,. \tag{8-37}$$

Now equation (8-33) for order $N_a - n$ can be computed if it is known for order n in the following way:

$$\begin{aligned}\overline{c^d_{N_a-n,\,N_a-n}(\rho_l)} &= \overline{c^d_{-n,-n}(\rho_l)} = \sum_{p=0}^{N_r-1} r_p c_{-n}(r_p) J_{-n}^d(r_p\rho_l) \\ &= \sum_{p=0}^{N_r-1} r_p c_n^*(r_p)(-1)^n J_n^d(r_p\rho_l) = \left\{J_n^d(x) = \left\{J_n^d(x)\right\}^*\right\} \\ &= (-1)^n \sum_{p=0}^{N_r-1} r_p c_n^*(r_p)\left\{J_n^d(r_p\rho_l)\right\}^* = (-1)^n \left\{\overline{c^d_{nn}(\rho_l)}\right\}^*.\end{aligned} \tag{8-38}$$

This means that only one half of the coefficients $\overline{c^d_{nn}(\rho_l)}$ has to be computed by means of equation (8-33) and the other half is almost immediately known by equation (8-38). In addition, this implies that only one half of the discrete Bessel coefficients, the computation of which is most time-consuming, has to be computed.
Further optimizations can be obtained by making use of recursion relations, asymptotic expansions, etc (see [Abramowitz and Stegun, 1965]).

8.4 Fast Hankel transforms

8.4.1 Introduction

As has been stated earlier, a literature search has been performed on the (fast) numerical evaluation of the Hankel transform because if it is to be used in practice for the computation of the DPFT, it is necessary that it can be computed relatively fast. All methods found in the literature describe the numerical evaluation of the continuous Hankel transform of equation (8-12). In fact, the problem of computing the spectra of areas with a shape as depicted in Figure 7-8 can be seen as computing the continuous Fourier transform over that area, given that only discrete values are available. This way, it can be seen that formula (8-3) may be approximated by formula (8-4) because of the fact that the Nyquist criterion is satisfied in both the range and azimuth directions. Therefore, the (fast) numerical evaluation of the continuous Hankel transform can be used for SHIRA. Because no time is available to implement or study in detail algorithms found in the literature, an overview is given of methods reported in the literature and their applicability to the SHIRA application is discussed. The criteria for the determination of the applicability of the separate methods will be their computational complexity and whether they have equally spaced input and output sample points. If available, the information on the computational complexity is also described. References where more information can be found are mentioned. The only purpose of this section is to investigate whether methods for the fast computation of the Hankel transform that are suitable for SHIRA are available. It is not the intention to describe the operation of such algorithms. With SHIRA, the sample points are equispaced. This can be an advantage since many algorithms for the evaluation of the Hankel transform rely on the use of the DFT and naturally require equispaced samples. For SHIRA, also the Hankel transformed values must be available on an equispaced grid.
The various algorithms for the (fast) numerical evaluation of the Hankel transform that have been published in the literature can be subdivided into general categories. The next sections describe these categories.

8.4.2 Numerical quadrature

The first category consists of methods based on *numerical quadrature*. The topic of quadrature is concerned with the problem of obtaining an approximation for the value of a definite integral, in this

case the (truncated Hankel transform). A linear combination of values of the integrand at the so-called quadrature points (the points of the integrand that are used in the quadrature rule, i.e. the linear combination) is taken. In general, these methods are inefficient (at least $O(N^2)$ complexity). Moreover, some of them need to know the function to be transformed analytically [Suter, 1991]. Therefore, it is concluded that this class of methods is not suitable for SHIRA and they will not be considered here.

8.4.3 Exponential change of variables

The methods in the second category use an *exponential change of variables* to rewrite the Hankel transform as a convolution or correlation integral, which can then be evaluated by FFTs ([Siegman, 1977] and [Talman, 1978]). These methods can approach $O(N \log_2 N)$ complexity. However, they have the disadvantage of requiring sampling over an exponential grid, thereby leading to important errors in the Hankel transforms of functions with an oscillating tail. If this kind of method were to be used for the SHIRA application, the available equispaced samples in the range direction (see equation (8-24)) would first have to be interpolated onto the required exponential grid thus introducing another error source. For these reasons, these methods are not very suitable for SHIRA.

8.4.4 Asymptotic expansion

The third category is based on the *asymptotic expansion* of the Bessel series in terms of sines and cosines ([Candel, 1981], [Piessens, 1982] and [Sharafeddin et al., 1992]). The integrals obtained this way correspond to sine and cosine transforms and can thus be numerically evaluated efficiently by means of FFTs. The complexity of this kind of methods is $O(N \log_2 N)$. However, due to the fact that the asymptotic expansions are only valid in a restricted range of arguments or orders of the Bessel function, the results are also accurate within a restricted range of arguments or orders. More research is required to investigate whether these methods are suitable for the SHIRA application. Since SHIRA has certain well-defined restrictions on the range of orders and arguments of the (discrete) Bessel function in equation (8-24), it is expected that these methods can offer a solution.

8.4.5 Projection methods

The fourth category consists of the *projection methods*. Two distinct classes of projection methods are possible: those that implement the projection-slice theorem and those that implement the reverse process, i.e. the back projection theorem. The projection-slice theorem states that the two-dimensional Fourier transform of a two-dimensional function can be calculated by taking one-dimensional Fourier transforms of the projections formed by integrating the function along parallel rays. Coupled with the relationship of the Hankel transform to the Polar Fourier transform, which is given in the following equation, this gives a method for calculating the Hankel transform:

$$PFT\{g(r)\exp(jn\theta)\} = F(\rho,\phi) = e^{-jn\phi} H_n\{g(r)\} = e^{-jn\phi} \int_0^\infty r g(r) J_n(\rho r)\, dr \,. \qquad (8\text{-}39)$$

Oppenheim et al. ([Oppenheim et al., 1978] and [Oppenheim et al., 1980]) give a derivation of both the projection-slice and back projection methods. In addition to a Hankel transform algorithm that makes use of the back projection theorem, Higgins and Munson [Higgings and Munson, 1987] suggest useful symmetry arguments to reduce the computational load. The projection methods generally require the efficient implementation of Chebychev and Abel transforms. The overall computational complexity of projection-based methods unfortunately is $O(N^2)$. Consequently, these methods are less appropriate for SHIRA.

8.4.6 Miscellaneous methods

The last category consists of a set of algorithms that cannot be filed directly into one of the above categories. Therefore, the methods in this category are called *miscellaneous* methods. Cree and Bones briefly list some of these methods in [Cree and Bones, 1993]. They claim that in general these methods suffer from poor computational efficiency. Consequently, the methods are of limited

applicability to SHIRA. However, an article describing a new fast method for computing the Hankel transform of equally spaced samples, has appeared recently [Knockaert, 2000]. Knockaert describes a decomposition of the Hankel transform into a sine, a cosine and inversion transform, which can be implemented in $O(N \log_2 N)$ complexity by means of fast sine and cosine transforms together with appropriately chosen interpolation schemes. So, this method computes the Hankel transform in several times the FFT complexity and therefore it seems to be appropriate for SHIRA. This method is different from the methods in the third category in that it does not make use of asymptotic expansions.

From the overview above, it can be concluded that either a method that makes use of asymptotic expansions or the method described by Knockaert is suitable to the SHIRA application. In order to choose the method that is best suited to SHIRA, a thorough study of both methods is required. If the comparison of the DPFT to the current method reveals that the DPFT yields better performance regarding the quality of the spectra, and TNO-FEL in that case decides to use the DPFT instead of the current method, this study can be carried out after that decision has been taken. It must be noted that all of the above described methods try to compute a discrete implementation of the continuous Hankel transform and thereby make use of some interpolation scheme somewhere in the processing stage. Because it has been shown in Section 8.3 that the continuous Hankel transform integral in equation (8-14) may be approximated by the simple summation in equation (8-24), it can be expected that the described methods can be sped up because probably the simplest interpolation scheme, namely the zero-order hold interpolation (replacing the integrals by summations), may be used somewhere in the processing stage.

8.5 Simulation of the DPFT via Hankel transforms and application to SHIRA spectral analysis

This section describes the use and simulation of the DPFT computed via Hankel transforms for SHIRA. The m-file that simulates this method is called PolFFT and can be found in Appendix H. A flow diagram of this m-file is depicted in Figure 8-4. PolFFT starts by calling an m-file (see Appendix D) in which the input parameters for the simulation are defined (box 2 in Figure 8-4). After that, it calls GenInpData that generates an input field on the circular section defined by the input parameters (box 3). As has been explained in Section 8.3 (also see equation (8-16)), the data must be available on a uniform polar grid (defined in box 4 in the flow diagram). This implies that the data has to be interpolated from a nonuniform to a uniform polar grid. Since the data is nonuniform in the azimuth direction only, a one-dimensional interpolation in the azimuth direction is sufficient to obtain the data on the desired uniform polar lattice (box 5). In contrast with the conventional method, which requires a two-dimensional interpolation, this is a one-dimensional interpolation. Therefore, it is expected that less interpolation errors occur, and hence less distortions in the spectra. Moreover, it is faster since computing *N* one-dimensional interpolations of *N* points is cheaper than computing one two-dimensional interpolation of *N*-by-*N* points in case of the scan conversion step of the current method. Again, a number of interpolation methods can be used (see Section 7.2.2). In order to obtain a fair comparison, the same interpolation as in the current method must be used (see Chapter 9). As with the current method (see Section 7.4), the interpolation method to be used can be chosen by the user in the input parameters file (Appendix D). A little problem arises when N_a is not a power of two. N_θ sweeps are available in the data window, whereas N_a sweeps fit in the entire revolution. Usually, N_θ, N_r and N_ρ are 128, and N_a has a nominal value of $N_a = T_R \cdot PRF = 1800$ (see Appendix A). However, in order to compute formulae (8-18) and (8-23) by means of FFTs, N_a needs to be a power of two. This problem can be solved by computing the first power of two that is higher than $T_R \cdot PRF$ and assigning this number to N_a:

$$N_a = 2^{\lceil \log_2 (T_R \cdot PRF) \rceil}. \tag{8-40}$$

The angles of the uniform polar grid onto which the data has to be interpolated then have to be defined by $\theta_i = i(2\pi / N_a)$ with $i = 0, 1, ..., N_a - 1$, where N_a is given by equation (8-40).
Since the new spacing between the uniform angles is smaller than the original one, the azimuth span of

the circular section that is analyzed becomes somewhat smaller. For the simulations, it is assumed that N_a is a power of two since this is not a structural problem.

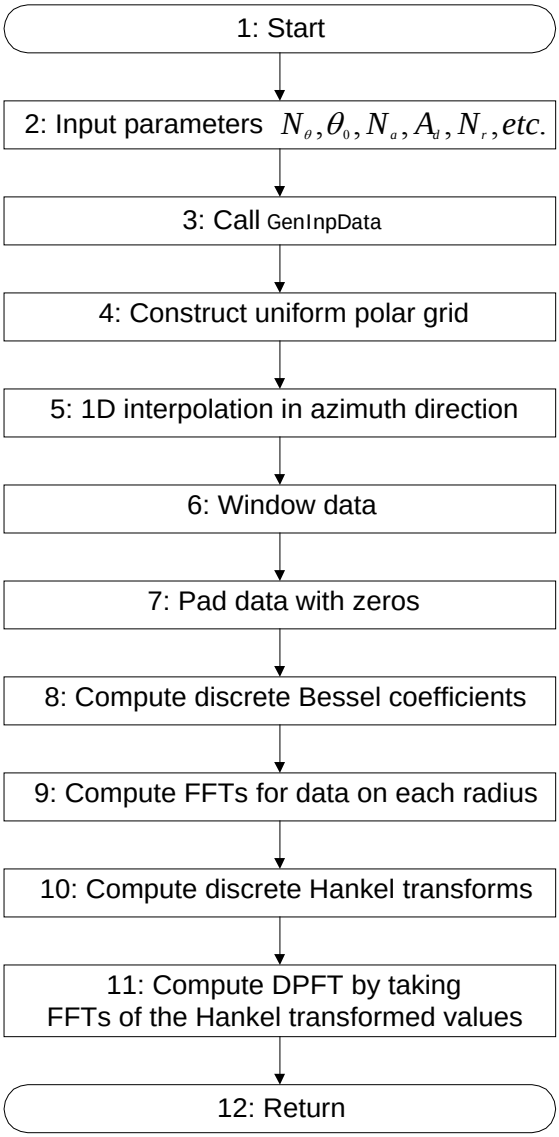

Figure 8-4 Flow diagram of PolFFT (computation of DPFT via Hankel transforms)

Because of the finite data window lengths, it may be desired to *taper* or *window* the data before transformation in order to reduce spectral leakage (box 6). Spectral leakage is the phenomenon that the energy from a single frequency is spread to many frequency locations due to the discontinuities at the edges of the data window. Several windows, such as Bartlett, Blackman, Chebychev, Hamming, Hanning, Kaiser, etc. can be used to remove the discontinuities. The type of window to be used for the simulations also can be specified in the input parameters file.

In order to compute equation (8-18), the function values to be (polar) Fourier transformed must be available over the entire revolution in the spatial domain, which is not the case in practice. With SHIRA, normally only a small area of the form depicted in Figure 7-10 is selected for analysis. The selected area lies between two constant angles and two constant ranges. N_θ sweeps are available in the data window, whereas N_a sweeps fit in the entire revolution, i.e. the spacing of the angles of the uniform polar grid is $2\pi / N_a$. Therefore, the data is padded with zeros on the circular section outside the shaded area in Figure 7-10. This is shown in box 7 of the flow diagram. In fact, padding $f(r_p, \theta_i)$ with zeros for constant radii r_p and then computing equation (8-18) by means of an FFT of length N_a

means that the spectrum of $f(r_p,\theta_i)$ for constant radii r_p is upsampled to N_a samples in the frequency domain instead of N_θ samples, because equation (8-18) can also be written as

$$c_n(r_p) = \sum_{i=0}^{N_\theta - 1} g(r_p,\theta_i)\exp(-jni2\pi / N_a), \tag{8-41}$$

where $g(r_p,\theta_i)$ denotes the unpadded version of $f(r_p,\theta_i)$ (also see Figure 7-10).
As can be seen in Figure 8-4 and Appendix H, after the data has been padded with zeros, the discrete Bessel coefficients $J_n^d(x)$ are computed by means of equation (8-32) for orders zero to $N_a/2$. Equation (8-32) is computed separately from equation (8-24), i.e. no fast discrete Hankel transform algorithm that tries to compute the combination of equations (8-24) and (8-32) as fast as possible without pre-computing the discrete Bessel coefficients, is used. Next, PolFFT computes the circular harmonics $c_n(r_p)$ of $f(r_p,\theta_i)$ by taking N_r one-dimensional FFTs of the data on each (constant) radius (also see Figure 8-3 and box 9 in the above flow diagram). The next piece of Matlab code from PolFFT shows that the circular harmonic coefficients $\overline{c_{nn}^d(\rho_l)}$, which are the n-th order discrete Hankel transforms of the coefficients $c_n(r_p)$, are computed by means of (8-24) for orders zero to $N_a/2$ (box 10).

```
% Compute the coefficients Cnn(rho[l]) by means of Hankel transforms
t_start=clock;                    % Measure computation time of discrete Hankel transforms
Cnn_rho_l=zeros(Na,Nrho);
for n=1:Na/2+1
  for l=1:Nrho
    Cnn_rho_l(n,l)=sum(r.*Cn_rk(n,:).*((reshape(Jn_lk(n,l,:),1,Nr))));
  end;
end;
% Now use the relation J(N-n,x)=((-1)^n)*J(n,x) to compute J(n,x) for n=Na/2+1, Na/2+2,..., Na-1
for n=2:2:Na/2 % Odd orders of Hankel transforms
  Cnn_rho_l(Na-n+2,:)=-conj(Cnn_rho_l(n,:));
end;
for n=3:2:Na/2 % Even orders of Hankel transforms
  Cnn_rho_l(Na-n+2,:)=conj(Cnn_rho_l(n,:));
end;
```

Then, from these known coefficients $\overline{c_{nn}^d(\rho_l)}$, the other coefficients for orders $N_a/2+1$ up to N_a-1 are computed by making use of equation (8-38). The code snippet also shows that distinct loops for the even and odd orders of the (discrete) Hankel transforms are used. This turned out to be faster than using one loop and taking the n-th power of -1. The final step to compute the estimated spectrum in polar coordinates is to take N_ρ one-dimensional FFTs of the spectral data on each (constant) radius in the polar Fourier domain (see Figure 8-3 and box 11 in the above flow diagram). After the spectrum has been computed, PolFFT calls MeasQualPFT to determine the quality of the spectrum in order to compare it to the current method and finally it returns.

8.6 Complexity and storage requirements of the DPFT via Hankel transforms

In this section, the complexity and storage requirements of the DPFT via Hankel transforms as computed in the previous section (i.e. without using fast Hankel transforms) is discussed in as far as they can be determined. In addition, the complexity in case fast Hankel transform algorithms are used, is discussed.
The computation method used in the simulations and described in the previous sections has some advantages and also some disadvantages. The main disadvantages are that the computation of the coefficients $J_n^d(r_p\rho_l)$ takes a lot of time and that a large amount of memory is required. The number of coefficients $J_n^d(r_p\rho_l)$ that is pre-computed and stored is:

$$NC = (N_a/2+1)N_rN_\rho. \tag{8-42}$$

For a typical situation, $N_r = N_\rho = 128$ and $N_a = 2048$ (see previous section). Thus, in that case about 16777216 storage cells are required. If each storage cell occupies four bytes in computer memory, a computer with an internal memory size of at least 64 MB is required, which is no problem nowadays. In addition, since normally only the start range of the analysis circular section changes, and the ranges in the Fourier domain and the orders for which $J_n^d(r_p\rho_l)$ must be known remain constant, a large set of these coefficients for a lot of ranges r_p can be pre-computed once and stored on disk. Later on, when a circular section at some specific start range must be analyzed, the relevant section of the block with the stored coefficients can be read from disk, which is much faster than computing the coefficients again. The advantage of pre-computing the discrete Bessel coefficients is that once they have been pre-computed, they *can be used for all images in a time series* due to the fact that after the one-dimensional interpolation in the azimuth direction of each image, all data is available on the same uniform polar grid. So, especially when the number of images in a time series is very large, this can be advantageous.

In order to analyze the total complexity of the computation of the DPFT via Hankel transforms, first the complexity of the separate processing stages is determined. The complexity of the one-dimensional interpolation is

$$C_1 N_r \cdot O(N_\theta) = C_1 \cdot O(N_r N_\theta), \tag{8-43}$$

where the constant C_1 depends on the interpolation method.

The complexity of the first set of N_r FFTs of length N_a is:

$$C_2 N_r \cdot O\{N_a \log_2(N_a)\} = C_2 \cdot O\{N_r N_a \log_2(N_a)\}, \tag{8-44}$$

where the constant C_2 depends on the specific FFT algorithm used.

If the complexity of the computation of one coefficient $J_n^d(x)$ is written as $C\{J_n^d\}$, the overall complexity of the computation of all coefficients $J_n^d(r_p\rho_l)$ is about

$$O\left(N_a N_r N_\rho / 2 \cdot C\left\{J_n^d\right\}\right). \tag{8-45}$$

Matlab simulations show that this is the most computation-intensive operation of the DPFT via Hankel transforms. Once the coefficients $J_n^d(r_p\rho_l)$ are known, the overall complexity of the discrete Hankel transforms is about (see equation (8-24) and Figure 8-3):

$$O\left(N_a N_r N_\rho / 2\right). \tag{8-46}$$

Finally, the complexity of the last set of N_ρ FFTs of length N_a is:

$$C_2 N_\rho \cdot O\{N_a \log_2(N_a)\} = C_2 \cdot O\{N_\rho N_a \log_2(N_a)\}, \tag{8-47}$$

where the constant C_2 is the same as in equation (8-44). So, the total complexity of the spectral analysis of SHIRA data via Hankel transforms in case the discrete Bessel coefficients are computed on the fly:

$$\begin{aligned} C_{tot} = {} & C_1 \cdot O(N_r N_\theta) + C_2 \cdot O\{N_r N_a \log_2(N_a)\} + O\left(N_a N_r N_\rho / 2 \cdot C\left\{J_n^d\right\}\right) \\ & + O\left(N_a N_r N_\rho / 2\right) + C_2 \cdot O\{N_\rho N_a \log_2(N_a)\}. \end{aligned} \tag{8-48}$$

If the discrete Bessel coefficients are pre-computed, the complexity is:

$$C_{tot} = C_1 \cdot O(N_r N_\theta) + C_2 \cdot O\{N_r N_a \log_2(N_a)\} + O\left(N_a N_r N_\rho / 2\right) + C_2 \cdot O\{N_\rho N_a \log_2(N_a)\}. \tag{8-49}$$

In case fast Hankel transform algorithms with $O(N \log_2 N)$ complexity (see Section 8.4) are used, instead of computing equations (8-24) and (8-32) separately, the overall complexity of the discrete Hankel transforms is (assuming that $N_\rho = N_r$):

$$C_3 N_a / 2 \cdot O\left(N_r \log_2 N_r\right) = C_3 / 2 \cdot O\left(N_a N_r \log_2 N_r\right), \tag{8-50}$$

where C_3 depends on the specific fast Hankel transform algorithm used, and the total becomes:

$$C_{tot} = C_1 \cdot O(N_r N_\theta) + C_2 \cdot O\{N_r N_a \log_2(N_a)\} + C_3 / 2 \cdot O\left(N_a N_r \log_2 N_r\right) + C_2 \cdot O\{N_\rho N_a \log_2(N_a)\}. \tag{8-51}$$

Now, a comparison between the complexity of the currently used spectral analysis method and the spectral analysis by means of the DPFT via Hankel transforms can be made. The total complexity of the currently used spectral analysis method is (equation (7-28)):

$$C_4 \cdot O\{N_x N_y\} + C_5 \cdot O\{N_x N_y \log_2(N_x N_y)\}, \tag{8-52}$$

where the constant C_4 depends on the two-dimensional interpolation method and the constant C_5 depends on the specific FFT algorithm used. It is difficult to compare this complexity to the one given by equation (8-51) because the various constants are not known. However, it seems reasonable to relate the constants in equation (8-51) to the constants in equation (8-52) in the following way. Because C_1 depends on the one-dimensional interpolation method used, it can probably be related to the constant C_4, which depends on the used two-dimensional interpolation method, by the following relation:

$$C_1 \cong \sqrt{C_4}. \tag{8-53}$$

In addition, it is reasonable to use the same kind of specific FFT algorithms, therefore

$$C_2 \cong C_5. \tag{8-54}$$

Finally, it is assumed that (see Section 8.4)

$$C_3 \approx 3C_5. \tag{8-55}$$

Now, if nominal values for the various parameters are chosen, the complexity for the spectral analysis by means of the DPFT via Hankel transforms given by equation (8-51) becomes

$$\begin{aligned} C_{tot} &= C_1 \cdot O(N_r N_\theta) + C_2 \cdot O\{N_r N_a \log_2(N_a)\} + C_3/2 \cdot O\left(N_a N_r \log_2 N_r\right) + C_2 \cdot O\{N_\rho N_a \log_2(N_a)\} \\ &= 128 \cdot 128 \cdot \sqrt{C_4} + 128 \cdot 2048 \cdot \log_2(2048) \cdot C_5 \\ &\quad + 3/2 \cdot 2048 \cdot 128 \cdot \log_2(128) \cdot C_5 + 128 \cdot 2048 \cdot \log_2(2048) \cdot C_5 \\ &= 16384 \cdot \sqrt{C_4} + 8519680 \cdot C_5 \end{aligned} \tag{8-56}$$

while the complexity of the currently used spectral analysis method given by equation (8-52) becomes

$$\begin{aligned} C_{tot} &= C_4 \cdot O\{N_x N_y\} + C_5 \cdot O\{N_x N_y \log_2(N_x N_y)\} \\ &= 128 \cdot 128 \cdot C_4 + 128 \cdot 128 \cdot \log_2(128 \cdot 128) \cdot C_5 \\ &= 16384 \cdot C_4 + 229376 \cdot C_5. \end{aligned} \tag{8-57}$$

By comparing equation (8-56) to (8-57) it can be seen that depending on the chosen interpolation scheme and the specific FFT algorithms used, either of the methods can be the fastest. The exact values can be computed if the constants are known, which is not the case.
In addition, it must be noted that if a circular section with a larger azimuth span is analyzed, the complexity of the new method increases slower than the complexity of the currently used method with comparable parameters. So, when large areas are analyzed, the new method is probably faster than the old one (however, this still has to be verified with the realistic values for the parameters C_4 and C_5). This can be a great advantage since the user probably wants to analyze an area as large as possible. Currently, relatively small areas are analyzed because of the computation times. However, this should be tested explicitly in practice with fast implementations of the Hankel transform and this is not done in the current project.

9 Quality measures for the spectra

In order to be able to measure the quality of the DPFT and to compare the DPFT to the current method, quality measures are required. As has been pointed out in Section 7.3, for the simulations an input field with one or more plane wave components is used. Fields with one plane wave component are used to measure the amount of distortion in the frequency domain and the location and width of the peak(s), while fields with two plane wave components are used to check whether the ratio between distinct components is preserved during the transformation. Since the coordinate systems and shapes of the Fourier plane of the DPFT differ from that of the current method, comparing both methods is not straightforward. The most important quality measure to determine is the *amount of distortion* in the spectra since this was the reason for investigating an alternative method. The location is measured in order to check whether the DPFT behaves as expected and the width can be used for the comparison of the spectral resolution of both methods. The m-file MeasQualCM (Appendix J) computes the values of the quality measures for the current method, while MeasQualPFT (Appendix K) calculates them in case the DPFT is used. Because these files contain a lot of irrelevant details (which are necessary, however), only their most important operations are described in the remainder of this section.
In order to be able to compare the amount of distortion, a signal-to-noise ratio has to be defined. This ratio is defined here as the amount of power contained in the peak of a plane wave component in the frequency domain divided by the power in the area surrounding the peak. Because of the finite data lengths (which in fact means that the data is tapered with rectangular windows), the peak in the frequency domain has a "sinc"-like shape, which implies that is has a certain width and that it is smeared out somewhat. This is the case for both the polar and Cartesian Fourier transforms. Therefore, this will not be considered to be distortion. In order to compute the power of the peak and the other quality measures, integrations or summations must be performed over the area where the peak is located. This area, called area of support, is defined as the area where the main lobe of the two-dimensional sinc-like pulse is located and it is determined in the following way. Firstly, it is observed that the contours of the peak have an elongated elliptical shape as can be seen in the example of Figure 9-1 (the dashed elongated shapes indicate the contour of the current method, while the solid ones indicate the contour of the PFT). For this reason, the area of support is assumed to be an ellipse.

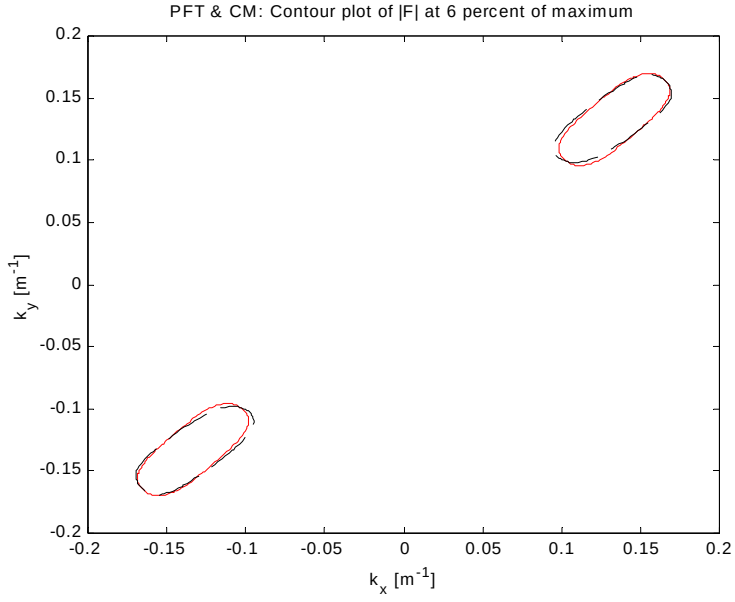

Figure 9-1 Contour plot of |F|

In addition, if the angle θ_{rot} of the centroid of the input data window (circular section or rectangle) is the same as the argument of the wave number, i.e. $\theta_{rot} = \arg(\vec{k}) = \tan^{-1}(k_y / k_x)$, one axis of the ellipse should point in the direction of the wave number position, i.e. in the ρ direction, and the other should

point in the azimuth direction, i.e. in the ϕ direction. The same holds if the arguments differ $\pi/2$ rad. Figure 9-1 illustrates this for $\theta_{rot} = \arg(\vec{k}) = \pi/4$. The modulus of the wave number and the start range r_0 (see Section 7.3) are the most important parameters for the determination of the quality measures. As an example, comparisons are performed only for $\theta_{rot} = \arg(\vec{k}) = \pi/4$, i.e. $k_x = k_y$, in Chapter 10. Now, the problem of determining the area of support reduces to determining the lengths of the axes and the position of the ellipse in the frequency domain. In order to determine the length of the axis of the ellipse in a specific direction, a slice through the peak in that direction is considered. The considered slice in the azimuth direction is the function (or data set) on the arc through the peak with the origin as the center (see Figure 9-2). The considered slice in the range direction is the function on the line through the peak that starts in the origin.

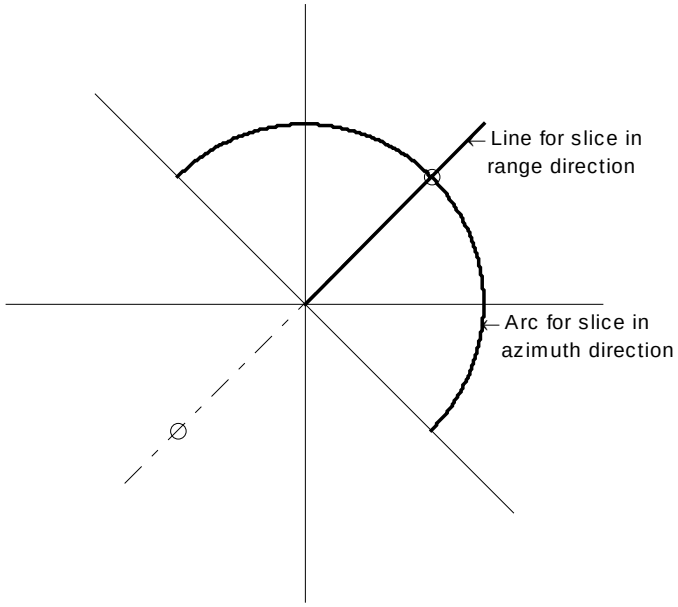

Figure 9-2 Definition of considered slices in the Fourier domain

This definition is rather arbitrary since also a Cartesian slice coordinate system, i.e. straight lines, can be positioned in the peak. However, the method that uses a straight line and an arc is preferred over the other because the quality will be determined in Chapter 10 as a function of the modulus of the wave number (among other parameters), and this parameter is constant on the circular arc. Also, the evaluated spectra on these slices of both the conventional method and the DPFT will be compared in Chapter 10. In order to do this, the spectral values obtained with the conventional method have to be interpolated onto the slice in the azimuth direction. This must be a fine interpolation; otherwise the spectra cannot be compared fairly. Therefore, and also in order to determine the other quality measures accurately, the frequency domain of the conventional method is upsampled by a factor 8 or even higher.

Only the area where the function (data) values satisfy certain conditions is considered to be the area of support. Since it is desired that the area of support starts where the main lobe starts to rise, only the area where the function rises faster than a certain adjustable value, and where it is higher than another adjustable value (that is proportional to the maximum of the function), is valid. When these restrictions are applied to the data on the slices, bounds for the ellipse in both the range and azimuth directions can be determined. Figure 9-3 shows an example of a possible location of the boundaries on the slices. The differences between the center of the peak and the boundaries determine the axes of the ellipse and hence the area of support (also see Appendix J and Appendix K).

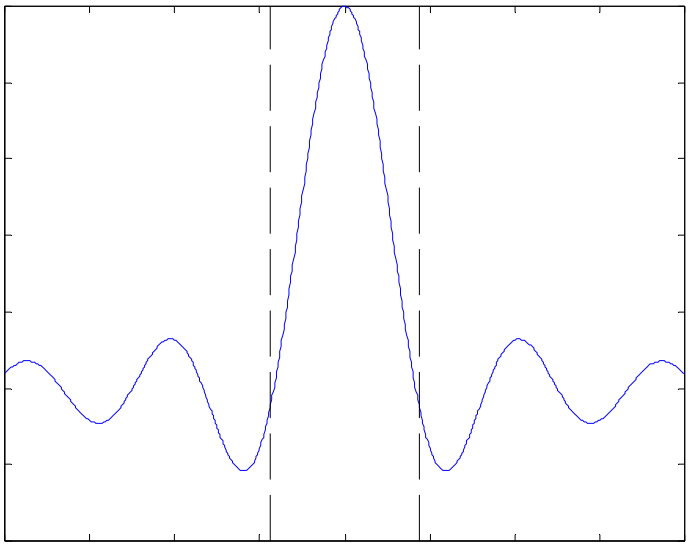

Figure 9-3 Determination of the axes of the ellipse of support

Moments of different order divided by the volume under the amplitude spectrum will be used as quality measures that provide information about the location of the peak(s) and about the effective widths (the width of a peak directly determines the spectral resolution). As a measure of the location of the peak of a function $h(x,y)$ in a specific direction, say the x-direction, the centroid (center of mass) of a function $h(x,y)$ in that specific direction given by the next formula is used:

$$m_x = \frac{\iint_{\Omega_{x,y}} x\,h(x,y)dxdy}{\iint_{\Omega_{x,y}} h(x,y)dxdy}. \tag{9-1}$$

It is the first-order moment in the x-direction divided by the volume of the function $h(x,y)$ on the area of support and it indicates where the function is mainly concentrated.
As a measure of the effective width of the peak in a specific direction, say the x-direction, the second-order moment in that direction divided by the volume is used:

$$m_{xx} = \sqrt{\frac{\iint_{\Omega_{x,y}} (x-m_x)^2\,h(x,y)dxdy}{\iint_{\Omega_{x,y}} h(x,y)dxdy}}. \tag{9-2}$$

In order to compute equations (9-1) and (9-2) for the amplitude spectrum over the ellipse of support, the ellipse is rotated to the positive k_x axis, after which the Cartesian definitions are used. In addition, since only discrete values of the spectrum are available, the integrals in the above equations are replaced by summations.
In addition to the quality measures defined by equations (9-1) and (9-2), the amplitude ratios of two peaks in the frequency domain corresponding to two plane wave components are computed by dividing the maximum of one peak by the maximum of the other.

10 Comparing the conventional method to the DPFT approach

10.1 Introduction

In order to make a decision about whether or not to use the DPFT instead of the current method, it is necessary to compare both methods in some way. Since the coordinate system and the shape of the analysis area of the DPFT differ from those of the conventional method, comparing them is not straightforward. The next section explains how the methods are compared, i.e. how the parameters for both methods are chosen in order to obtain a "fair" comparison. As has been pointed out in Section 7.1, in order to make a fair comparison the current method is altered and the resulting method is called the *conventional* method. With the conventional method, the data windows of both methods have the same area and they coincide as much as possible, which is not the case with the currently used method. The simulation of the conventional method is described in Section 10.3. Finally, the results of the comparisons are presented in Section 10.4.

10.2 Input data windows for comparison

Because the conventional method has to be compared to the PFT method, the size and location of the input rectangle of each image in a time series depend on the parameters chosen for the corresponding PFT analysis of approximately the same area. Thus, given the parameters of the circular section (position, size and sampling distances), the parameters for the input data windows of both methods have to be chosen in such a way that a fair comparison between the methods can be made. It seems reasonable to let both *shapes coincide* as much as possible (see Figure 10-1) and to choose *equal areas*, which implies that the energy of the data in both shapes is approximately the same. Moreover, the *same number of input data points* will be used in both methods. The first requirement implies that the parameters have to be chosen in such a way that the centroids ("centers of mass") of both shapes will coincide. The centroid is denoted by point P in the figure. The coordinates of the centroid P (of both the circular section and the rectangle) are computed by the formulae:

$$\begin{aligned} P_{c,x} &= \frac{1}{A}\iint_{\Omega_{x,y}} x\,dx\,dy = \frac{1}{A}\iint_{\Omega_{r,\theta}} r^2\cos(\theta)\,dr\,d\theta \\ &= \frac{1}{A}\int_{r_{\min}}^{r_{\max}}\int_{\theta_{\min}}^{\theta_{\max}} r^2\cos(\theta)\,dr\,d\theta = \frac{(r_{\max}^3 - r_{\min}^3)(\sin(\theta_{\max}) - \sin(\theta_{\min}))}{3A} \end{aligned} \tag{10-1}$$

and

$$\begin{aligned} P_{c,y} &= \frac{1}{A}\iint_{\Omega_{x,y}} y\,dx\,dy = \frac{1}{A}\iint_{\Omega_{r,\theta}} r^2\sin(\theta)\,dr\,d\theta \\ &= \frac{1}{A}\int_{r_{\min}}^{r_{\max}}\int_{\theta_{\min}}^{\theta_{\max}} r^2\sin(\theta)\,dr\,d\theta = \frac{(r_{\max}^3 - r_{\min}^3)(\cos(\theta_{\min}) - \cos(\theta_{\max}))}{3A}, \end{aligned} \tag{10-2}$$

where the area of the section A is given by:

$$A = \iint_{\Omega_{x,y}} dx\,dy = \iint_{\Omega_{r,\theta}} r\,dr\,d\theta = \int_{r_{\min}}^{r_{\max}}\int_{\theta_{\min}}^{\theta_{\max}e} r\,dr\,d\theta = \tfrac{1}{2}(r_{\max}^2 - r_{\min}^2)(\theta_{\max} - \theta_{\min}). \tag{10-3}$$

The parameters $r_{\min}$, $r_{\max}$, $\theta_{\min}$ and $\theta_{\max}$ are defined in Section 7.3 (also see Figure 7-8 and Appendix C). The next requirement is to choose equal areas for both shapes. This means that A must be equal to $N_x r_x N_y r_y$. As described in Section 7.2.3, the size of the rectangle is determined by the number of uniformly spaced grid points in the x-direction and y-direction N_x and N_y, and the grid sizes r_x and r_y (sampling distances) in both directions. The x-direction of the rectangle is now defined as the direction relative to the rectangle that aligns with the line through the middle of the circular section

(the line through OP in Figure 10-1), i.e. the radial direction of the circular section, and it is perpendicular to the azimuth direction. In the same way, the y-direction of the rectangle is defined as the direction relative to the rectangle that aligns with the direction perpendicular to the line OP, i.e. the azimuth direction of the circular section. Because the number of points in the rectangle must be equal to the number of points in the circular section, the product $N_x N_y$ must be equal to the product $N_r N_\theta$. Since the x-direction of the rectangle aligns with the line through the middle of the circular section, N_x is chosen to be equal to N_r, the number of samples of the circular section in that direction. Likewise, since the y-direction of the rectangle aligns with the azimuth direction, N_y is chosen to be equal to N_θ. The length of the rectangle in the x-direction is made equal to the length of the circular section in the radial direction. This implies that the sampling distance of the rectangle in the x-direction $r_x = \Delta r$.

The sampling distance of the rectangle in the y-direction can now be calculated from the equation $A = N_x r_x N_y r_y$:

$$r_y = \frac{A}{N_x r_x N_y} = \frac{A}{(N_r \Delta r) N\theta} = \frac{\frac{1}{2}(r_{\max}^2 - r_{\min}^2)(\theta_{\max} - \theta_{\min})}{(r_{\max} - r_{\min}) N_\theta} = \frac{\frac{1}{2}(r_{\max} + r_{\min})(\theta_{\max} - \theta_{\min})}{N_\theta} = \frac{\pi}{N_a}(r_{\max} + r_{\min}). \quad (10\text{-}4)$$

Note that the size of the rectangle in the y-direction is equal to the arc length of the arc defined by the intersection of the circular section with a circle with radius $(r_{\min} + r_{\max})/2$:

$$N_y r_y = \frac{A}{N_x r_x} = \frac{A}{N_r \Delta r} = \frac{\frac{1}{2}(r_{\max}^2 - r_{\min}^2)(\theta_{\max} - \theta_{\min})}{r_{\max} - r_{\min}} = \frac{(r_{\max} + r_{\min})}{2}(\theta_{\max} - \theta_{\min}). \quad (10\text{-}5)$$

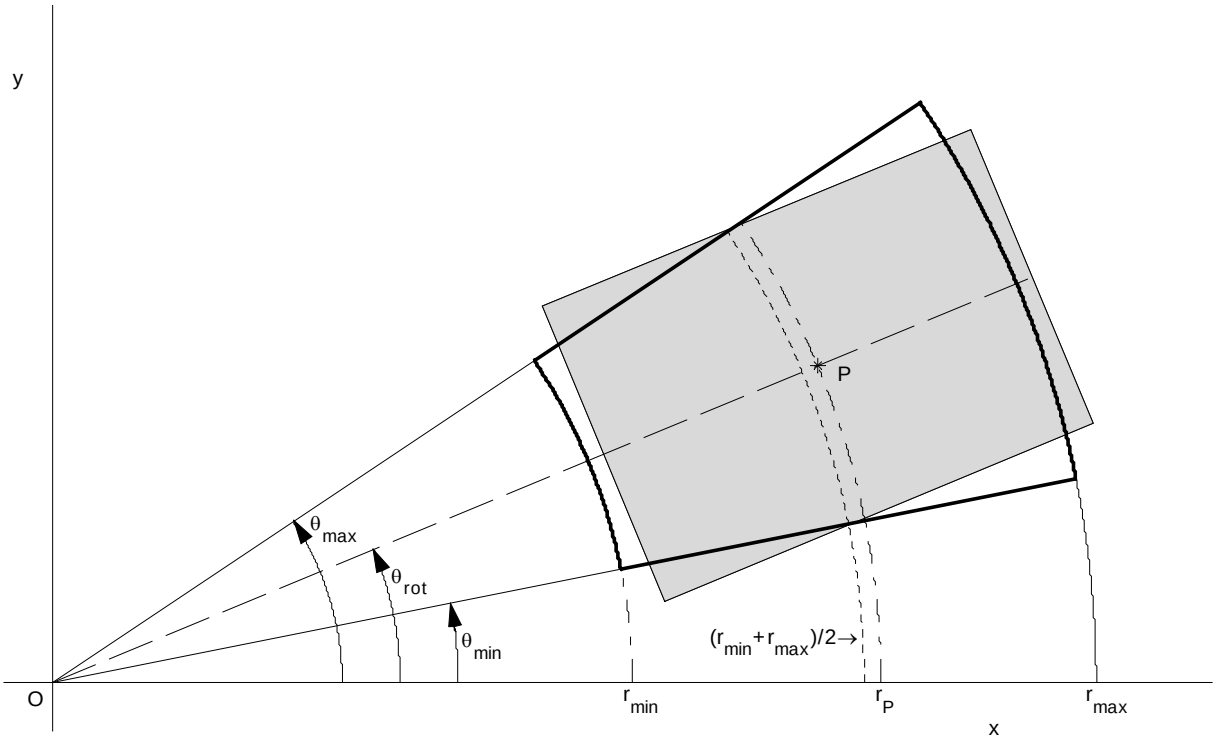

Figure 10-1 Areas for comparison of PFT to the conventional method

In order to let both shapes coincide as much as possible, the rectangle must be rotated with respect to the positive x-axis over the angle $\theta_{rot} = \tan^{-1}(P_{cy}/P_{cx})$, which is the angle of the line OP with the positive x-axis. Now, all parameters of the rectangle are known and its coordinates can be computed (see next section). The size and position of the rectangle with respect to the circular section as calculated with the described method are illustrated in the above figure. It also shows that the distance

r_P of the centroid P to the origin is larger than the mean of $r_{\min}$ and $r_{\max}$. This is due to the fact that the width of the circular section as measured perpendicular to OP becomes larger as the range grows. Now, all parameters of the rectangle are known, and the spectra can be computed. The result of the two-dimensional FFT also has to be rotated over θ_{rot} because the spatial area to be analyzed is also rotated over that angle. After this rotation, the quality of the spectra is measured by means of the quality measures defined in Chapter 9 (see Section 10.4 for the results).

10.3 Simulation of conventional method

The computation of the spectra by means of the conventional method is simulated by the m-file ConvMeth that is listed in Appendix I. Like the other simulation files, ConvMeth starts by calling the m-file in which the input parameters for the simulation are defined (see Appendix D). After that, the position and size of the rectangle are computed from the given parameters N_θ, θ_0, N_r and r_0 by means of the equations (10-1), (10-2), (10-3), etc. Then, the coordinates of the uniform rectangular grid are computed in the following way: first the middle of the rectangle is placed at the positive x-axis at a distance r_P from the origin, where r_P is the distance of the center P to the origin. Next, it is rotated over the angle θ_{rot} into its final position. Figure 10-1 shows that the available data is used more efficiently than with the current method.
Because the conventional method, like the other methods, has to start with polar input data, a circular section that encompasses the shaded rectangle is used as the input data window (see Figure 10-2). After the parameters of this encompassing circular section have been determined, the same actions as in CurrMeth are performed. The m-file GenInpData is called to generate the input field. Next, the uniform rectangular coordinates are converted to polar coordinates and the data sample points in the circular section are interpolated onto the sample points of the rectangle (thus, the interpolation is performed in the polar domain). After the two-dimensional interpolation, the data is windowed and the two-dimensional FFT is computed. Finally, the Cartesian coordinate system of the FFT samples is rotated over θ_{rot} in order to compensate for the rotation of the analysis rectangle.

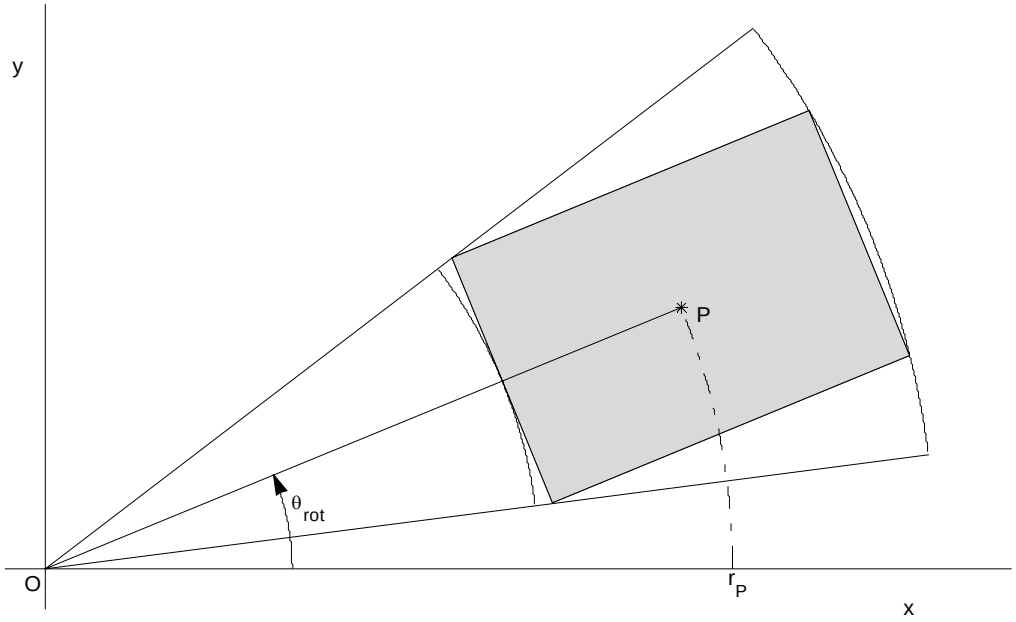

Figure 10-2 Position and orientation of rectangular analysis area for conventional method

10.4 Results of comparisons

This section describes the results of the comparison of the DPFT to the conventional method. The quality is measured as a function of the two parameters r_0 and $k = |\vec{k}|$ because these are essential for the comparison. The comparisons are made for three start ranges, namely $r_0 = 500\ m$, $r_0 = 1000\ m$ and $r_0 = 1500\ m$. The results are described in sections 10.4.1, 10.4.2 and 10.4.3 respectively. For all simulations described in this section the following parameters values are used: $N_\theta = 64$, $\theta_0 = 0.5492$ rad, $N_a = 1024$, $N_r = 64$, $\Delta r = 3.75\ m$, $N_\rho = 128$ and $A_d = 25\ \%$. These parameters are chosen in such a way that the analyzed area is comparable to the area analyzed with SHIRA (the only difference is that it is smaller in order to keep the computation times within acceptable limits). A lot of simulations with other parameter combinations have been tried in order to try to make the spectral distortions due to the NN interpolation method observed by TNO-FEL as high as possible. As has been explained in Section 7.2.2, the larger the wave numbers, the larger the distortion. The high spatial frequencies correspond to small wavelengths and therefore only a few sample points per wavelength are available. An example of this is given next. If the ratio of the maximum range to the minimum range is about 1.5 for example (which is the case for $r_0 = 500\ m$ with the above described parameter values), even the waves with the largest value of k, i.e. the smallest wave length, are oversampled near the origin by a factor 1.5. Hence, for most wave numbers the corresponding waves are heavily oversampled in the part of the data window that is closest to the origin. Consequently, the NN interpolation method performs reasonably well for most frequencies. Only for the largest values of k, some distortion is introduced. Also simulations showed that better interpolation methods such as bi-linear only yield minimal improvements.

However, the next sections show that the distortion that is present in the spectra, from whatever source it arises, is less with the DPFT than with the conventional method. Since the values used for k depend on the start range, they are given in each section separately. In all comparisons, the NN interpolation method is used by both the conventional method (two-dimensional interpolation) and the DPFT (one-dimensional interpolation) in order to make a fair comparison. For each range, tables that list the values of the most important quality measures for three values of k are given: one for k small, one for k in the middle of the allowed range and one for k large, near the maximum value given by equation (7-24). The symbols used in the tables have the following meaning: $(S/N)_{CM}$ and $(S/N)_{PFT}$ denote the signal-to-noise ratios of the conventional method and the DPFT respectively, $(EW_\rho)_{CM}$ and $(EW_\rho)_{PFT}$ denote the effective widths of the peaks (of the amplitude spectrum) in the range direction of the conventional method and the DPFT respectively, while $(EW_\phi)_{CM}$ and $(EW_\phi)_{PFT}$ denote the effective widths in the azimuth direction. In the tables presented in this chapter, the signal-to-noise ratios are expressed in dBs and the effective widths in m^{-1}.

Besides the tables, plots of slices through the peak in the spectrum in the range and azimuth directions (see Chapter 9) are given for each range and for the three values of k given in the tables. In all slice plots, the *dashed line indicates the slice of the conventional method and the solid line indicates the DPFT slice*. Six slice plots are arranged in one main figure. The slices in the range direction are at the left side of the main figure and the slices in the azimuth direction at the right side. The value of k increases from top to bottom for each row in the main figure. The individual plots in the main figure are referred to by numbers that are counted along the top row of the figure, then the second row, etc. All plots have been scaled for easy comparison and are expressed in dBs. The maximum value is set at 20 dB and all vertical axes range from –25 to +20 dB. In addition to these slice plots, plots shat show the signal-to-noise ratio as a function of k are used to illustrate the quality performance differences between both methods.

As has been described at the end of Chapter 9, the location of the peaks has also been checked. Because these locations were always correct for both methods, no results are listed in this chapter. The same holds for the amplitude ratios of two plane wave components. With both methods the ratio is preserved during the transformation and therefore no results are listed.

10.4.1 Comparison for data windows at small range

In the first set of comparisons, $r_0 = 500\ m$. The maximum value of k that satisfies the Nyquist criterion is about 0.55 m^{-1} (equation (7-24)). The following table lists the results for three values of k in the allowed range when no windowing is applied to the data.

Table 1 Comparison results for small range without windowing

$k = \lvert \vec{k} \rvert$	$(S/N)_{CM}$	$(S/N)_{PFT}$	$(EW_\rho)_{CM}$	$(EW_\rho)_{PFT}$	$(EW_\phi)_{CM}$	$(EW_\phi)_{PFT}$
0.0982	6.1623	6.4254	0.0108	0.0105	0.0104	0.0102
0.2945	4.7513	6.5172	0.0104	0.0106	0.0104	0.0103
0.4909	2.7079	7.0090	0.0106	0.0106	0.0108	0.0102

The second and third columns of Table 1 show that the signal-to-noise ratio as defined in Chapter 9 is better for the DPFT than for the conventional method. The gain in signal-to-noise ratio for large wave numbers is about 4.3 dB. The last four columns of Table 1 show that the effective width of the peak is also approximately the same. From this, it can be concluded that the spectral resolution of both methods is the same. In addition, it is constant as a function of k. Figure 10-3 shows the slices of the amplitude spectrum in the range and azimuth directions for the values of k listed in the first column of Table 1 (from top to bottom).

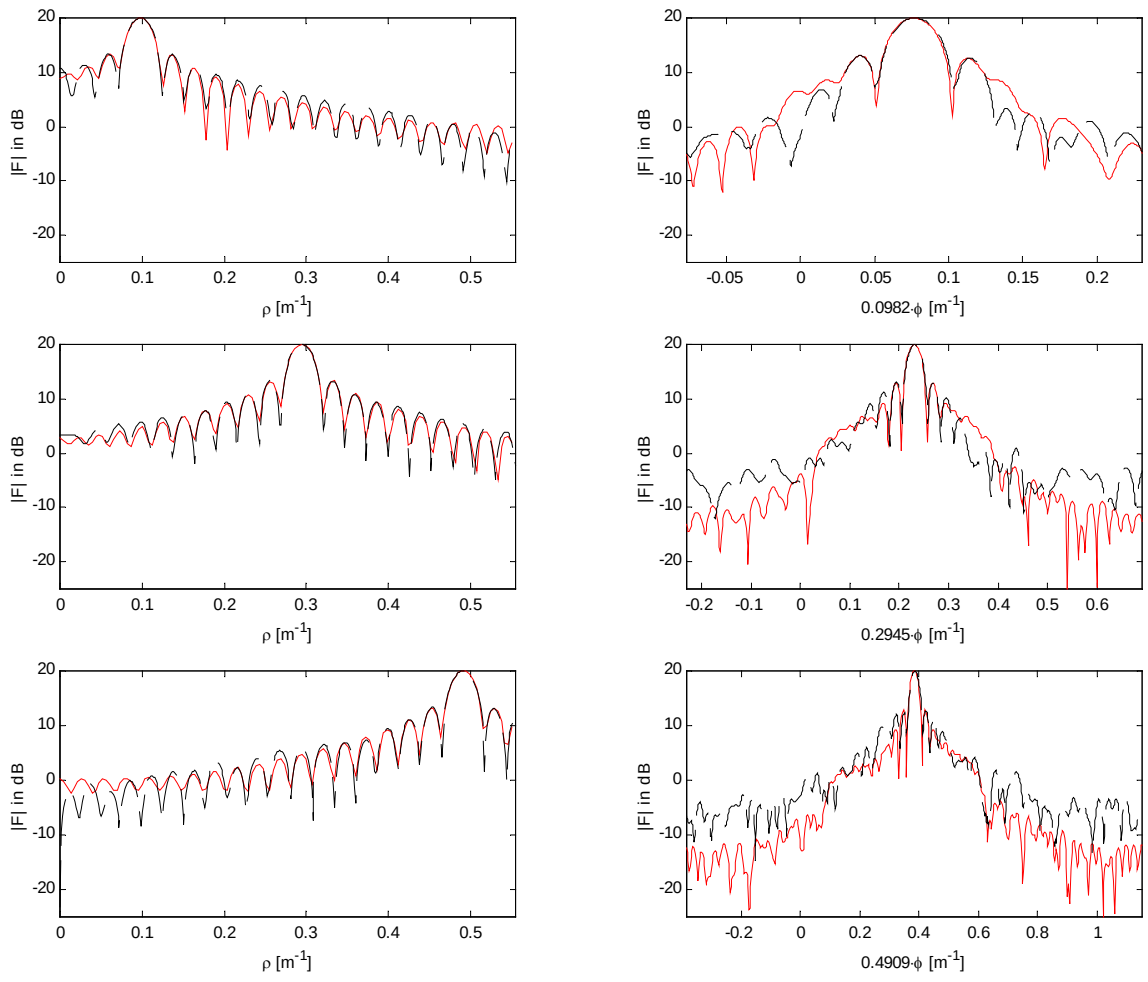

Figure 10-3 Slices of amplitude spectrum for small range, no windowing

All plots in the figure indicate that the quality of the conventional method and the DPFT *on these specific slices* is approximately the same. Only the slices in the azimuth direction for the middle and largest values of k show that the DPFT is somewhat better (about 5 dB). However, it must be noted that distortion can be present in the spectrum at places other than the considered slices. As can be concluded from second and third columns of Table 1, this is the case. Also Figure 10-4 illustrates that $(S/N)_{CM}$ decreases with increasing k, while $(S/N)_{PFT}$ remains approximately the same. From this figure it can be concluded that especially for large wave numbers, i.e. high spatial frequencies, the DPFT performs better.

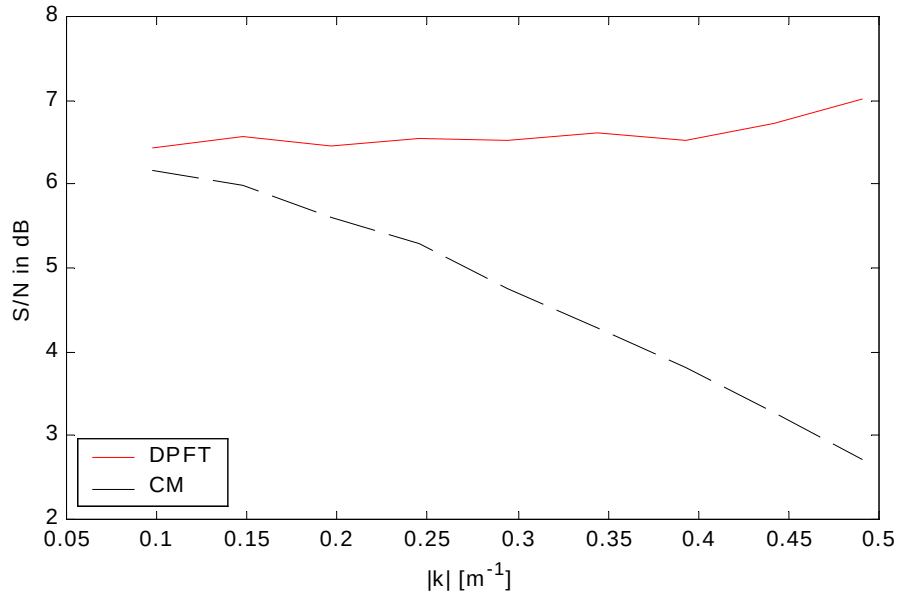

Figure 10-4 Signal-to-noise ratio of DPFT and CM as a function of $k = |\vec{k}|$ (small range, no windowing)

Table 2 lists the results for the same parameters in case Hanning windowing is applied to the data. This window is specified by the following equation:

$$w(n) = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N-1}\right)\right], \tag{10-6}$$

where

$$0 \le n \le N-1 \tag{10-7}$$

with *N* the number of samples in a specific direction and *n* the sample number.
The conventional method windows the data in the x and y directions and the DPFT in the range and azimuth directions (also see Appendix I and Appendix H respectively).

Table 2 Comparison results for small range with Hanning windowing

$k = \lvert\vec{k}\rvert$	$(S/N)_{CM}$	$(S/N)_{PFT}$	$(EW_\rho)_{CM}$	$(EW_\rho)_{PFT}$	$(EW_\phi)_{CM}$	$(EW_\phi)_{PFT}$
0.0982	21.4607	28.1290	0.0189	0.0194	0.0186	0.0183
0.2945	12.8177	27.9814	0.0189	0.0190	0.0194	0.0186
0.4909	8.3503	28.3145	0.0189	0.0190	0.0205	0.0187

The second and third columns of Table 2 show that again the signal-to-noise ratio is better for the DPFT than for the conventional method, especially for large wave numbers (here, the gain in signal-to-noise ratio is about 20 dB). The last four columns of Table 2 show that the effective width of the peak is approximately the same for both methods. Hence, the spectral resolution is the same.
Figure 10-5 shows the slices of the amplitude spectrum in the range and azimuth directions in case Hanning windows are used. All plots in the figure show that the quality of the DPFT *on these specific slices* is better. Especially the slices in the azimuth direction show that the DPFT performs much

better. In plots 4 and 6 the differences are even larger than 15 or 20 dB. The slices in the range direction show that the DPFT especially performs better for large values of k. So, the distortion introduced when transforming high frequencies is less for the DPFT. Since the only difference between Figure 10-3 and Figure 10-5 is that the data is windowed (after all other steps, such as the interpolation, have been carried out), it must be concluded from this that windowing in the range and azimuth directions in case of the DPFT is better suited to the problem than windowing in Cartesian directions, as is currently done. However, no satisfying explanation of this phenomenon has been found yet and hence more research is required. The impact of windowing in a certain direction on the polar frequency spectrum has to be determined.

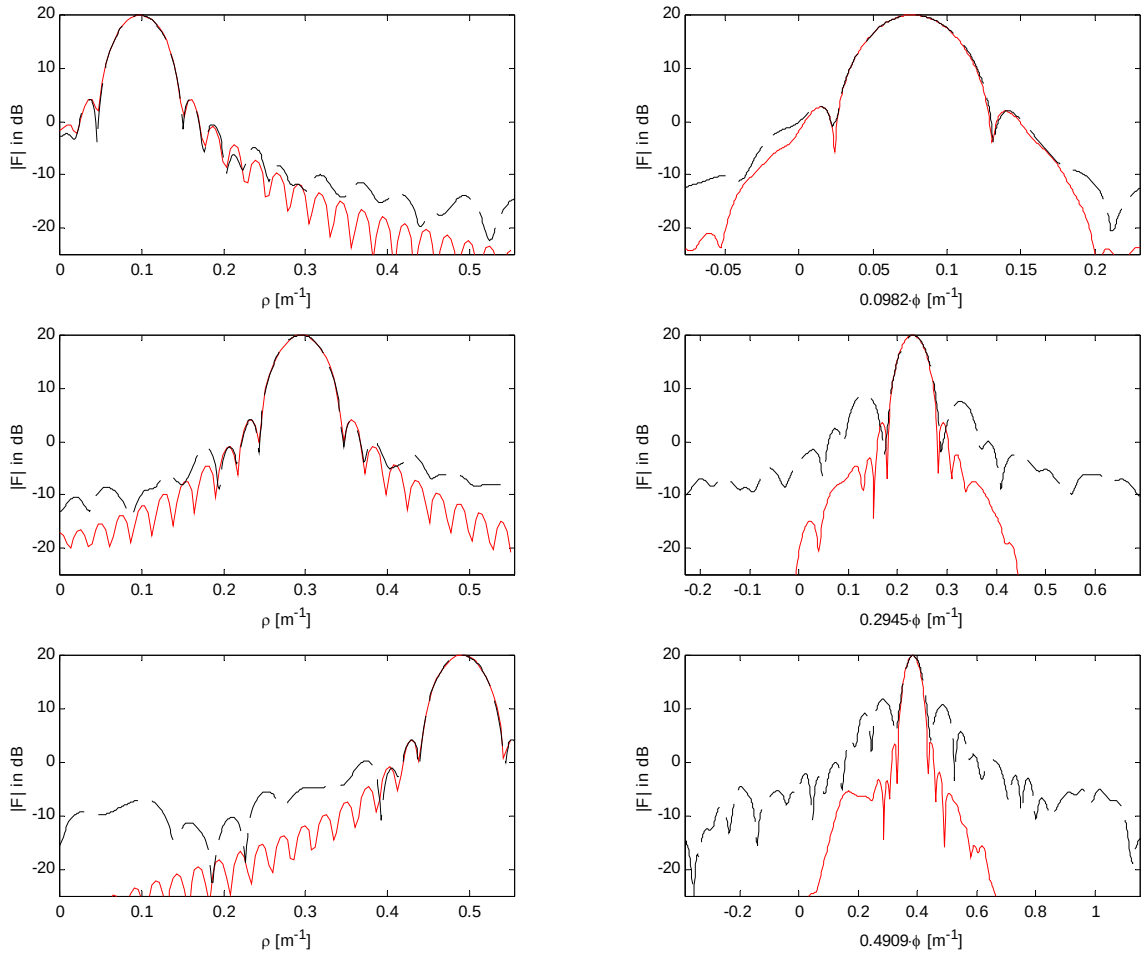

Figure 10-5 Slices of amplitude spectrum for small range, Hanning windowing

Figure 10-6 illustrates that $(S/N)_{CM}$ decreases as k increases, while $(S/N)_{PFT}$ approximately remains the same. Again, it is seen that especially for large wave numbers, the DPFT performs better. By comparing Figure 10-6 to Figure 10-4 and Table 2 to Table 1, it can be seen that the signal-to-noise ratios have increased significantly. For the conventional method, the improvements range from about 6 dB for large wave numbers to about 15 dB for small wave numbers, while for the DFPT the improvements are about 21 dB for all wave numbers. Thus, by tapering the data windows smoothly to zero at the edges, the height of the side lobes is greatly reduced. However, this is achieved at the expense of a wider main lobe, which implies that the *spectral resolution is reduced significantly*. The width of the peak(s) has increased by about 75 % in both directions (see the last four columns of Table 2 and Table 1).

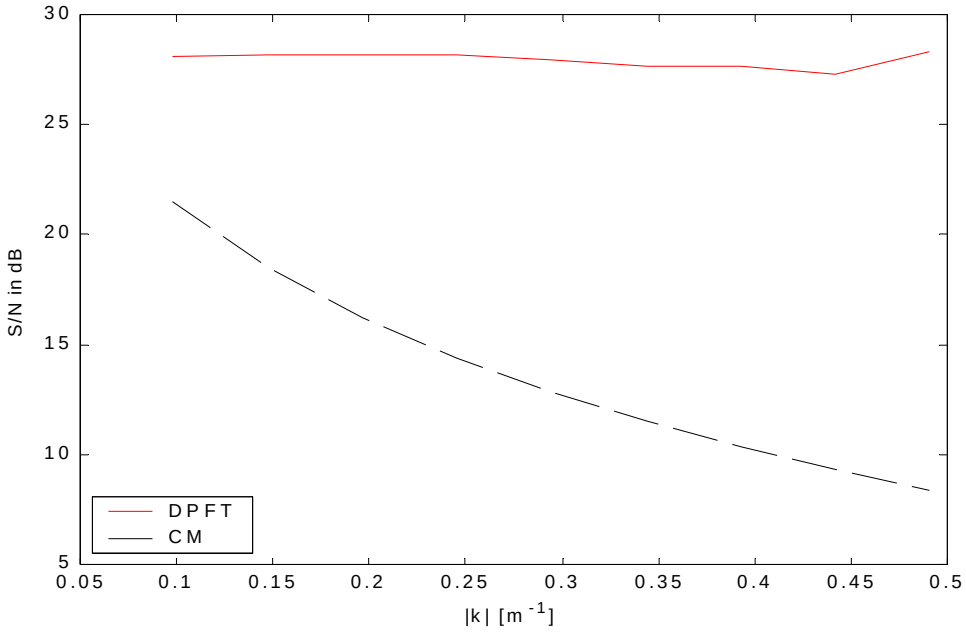

Figure 10-6 Signal-to-noise ratio of DPFT and CM as a function of $k = |\vec{k}|$ *(small range, Hanning windowing)*

10.4.2 Comparison for data windows at middle range

In the second set of comparisons, $r_0 = 1000\ m$. The maximum value of k that satisfies the Nyquist criterion is about 0.33 m^{-1}. The following table lists the results for three values of k in the allowed range when no windowing is applied to the data.

Table 3 Comparison results for middle range without windowing

$k = \lvert\vec{k}\rvert$	$(S/N)_{CM}$	$(S/N)_{PFT}$	$(EW_\rho)_{CM}$	$(EW_\rho)_{PFT}$	$(EW_\phi)_{CM}$	$(EW_\phi)_{PFT}$
0.0582	6.4255	6.6995	0.0102	0.0103	0.0058	0.0057
0.1745	5.7707	6.8189	0.0105	0.0107	0.0058	0.0058
0.2909	4.6425	7.7477	0.0104	0.0107	0.0057	0.0058

The results are very similar to the results of the comparisons for the small range. The signal-to-noise is better for the DPFT than for the conventional method. For large wave numbers, the gain is about 3 dB. The spectral resolution of both methods is the same and it is constant as a function of k. However, as compared to the small range, the resolution in the azimuth direction is much better for both methods (see the last two columns of Table 1 and Table 3). All plots in Figure 10-7 show that quality of the conventional method and the DPFT on the selected slices is approximately the same. Only the slices in the azimuth direction for the middle and largest values of k show that the DPFT is somewhat better. Figure 10-8 illustrates that $(S/N)_{CM}$ decreases as k increases, while $(S/N)_{PFT}$ even increases a little. Again, the DPFT performs better for the large wave numbers.

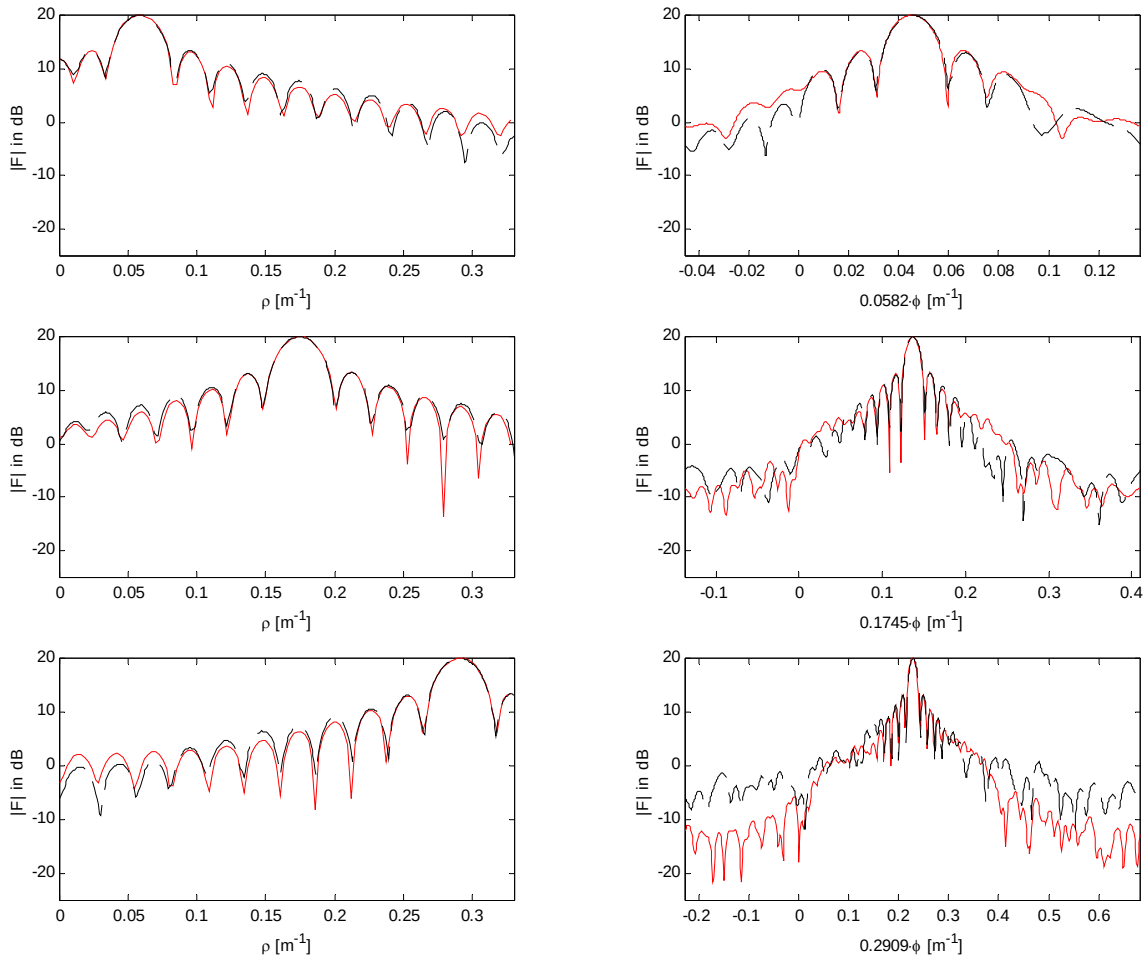

Figure 10-7 Slices of amplitude spectrum for middle range, no windowing

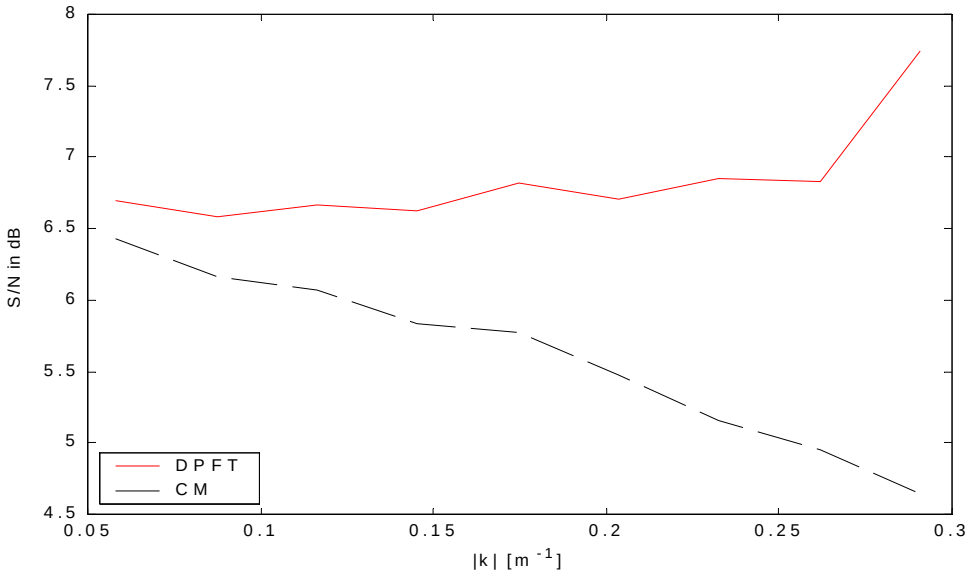

Figure 10-8 Signal-to-noise ratio of DPFT and CM as a function of $k = |\vec{k}|$ *(middle range, no windowing)*

The following table lists the results in case Hanning windowing is applied to the data.

Table 4 Comparison results for middle range with Hanning windowing

$k = \lvert \vec{k} \rvert$	$(S/N)_{CM}$	$(S/N)_{PFT}$	$(EW_\rho)_{CM}$	$(EW_\rho)_{PFT}$	$(EW_\phi)_{CM}$	$(EW_\phi)_{PFT}$
0.0582	23.1922	27.7587	0.0186	0.0207	0.0102	0.0100
0.1745	15.9146	27.5733	0.0187	0.0190	0.0103	0.0103
0.2909	11.7083	27.4308	0.0189	0.0183	0.0107	0.0103

Again, the results are very similar to the results of the comparisons for the small range. The signal-to-noise ratios have increased significantly and again the DPFT performs better than the conventional method. For large wave numbers, the gain is about 16 dB. The price paid by the spectral resolution is again about 75 % in both directions. Figure 10-9 shows the slices of the amplitude spectrum in the range and azimuth directions. All plots in the figure show that the quality of the DPFT is better, especially on the slices in the azimuth direction. For large wave numbers, the profit gained is the largest.

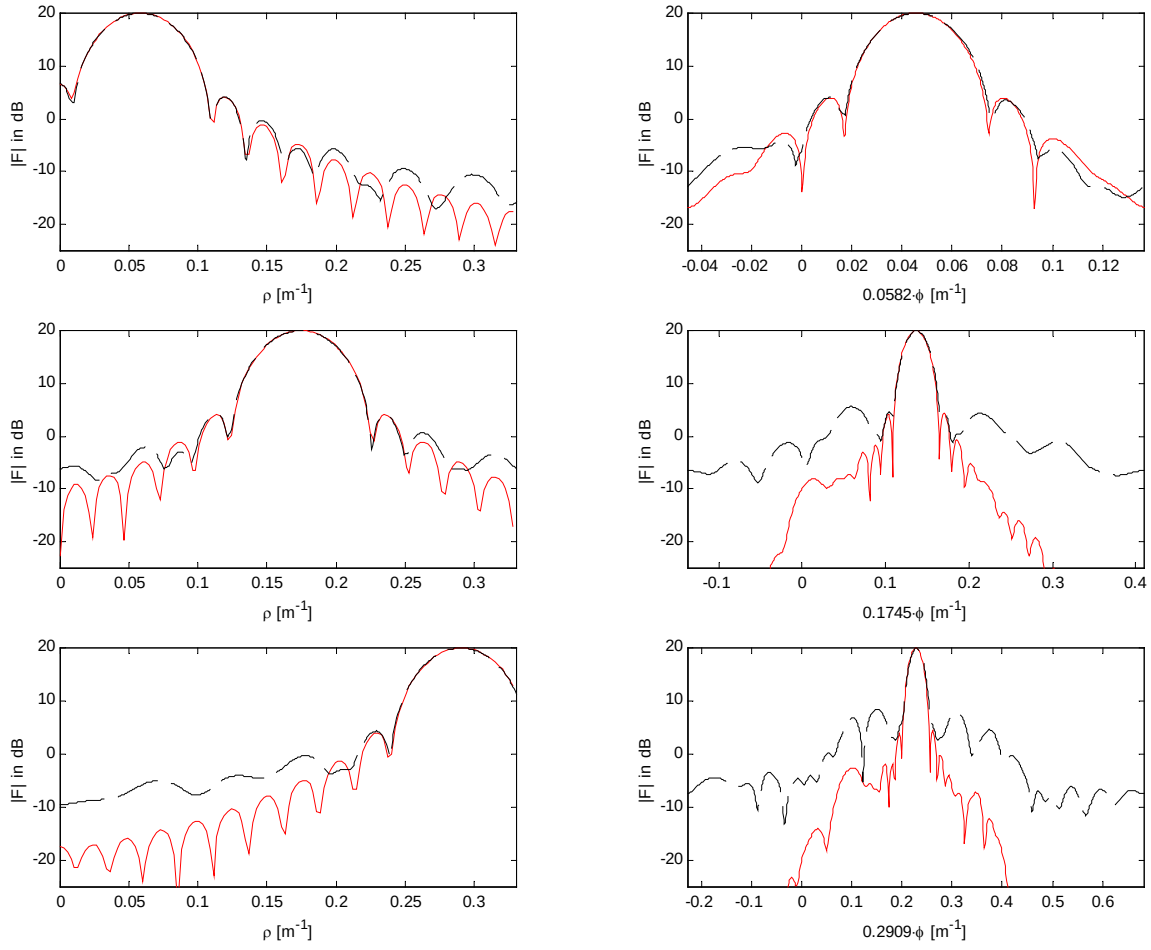

Figure 10-9 Slices of amplitude spectrum for middle range, Hanning windowing

Figure 10-10 again illustrates that $(S/N)_{CM}$ decreases as k increases, while $(S/N)_{PFT}$ remains the same. Also, it is seen that especially for large wave numbers the DPFT performs better and that the signal-to-noise ratios have increased significantly.

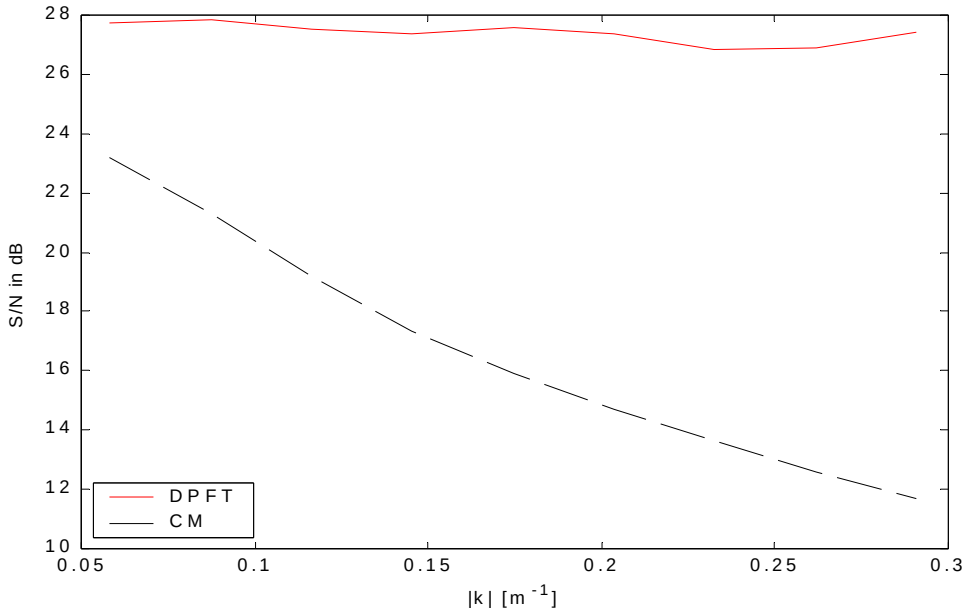

Figure 10-10 Signal-to-noise ratio of DPFT and CM as a function of $k = |\vec{k}|$ *(middle range, Hanning windowing)*

10.4.3 Comparison for data windows at large range

Finally, in the last set of comparisons, $r_0 = 1500\ m$. The maximum value of k that satisfies the Nyquist criterion is 0.23 m^{-1}. The following table lists the results when no windowing is applied to the data.

Table 5 Comparison results for large range without windowing

$k = \|\vec{k}\|$	$(S/N)_{CM}$	$(S/N)_{PFT}$	$(EW_\rho)_{CM}$	$(EW_\rho)_{PFT}$	$(EW_\phi)_{CM}$	$(EW_\phi)_{PFT}$
0.0418	6.6150	6.9530	0.0109	0.0117	0.0037	0.0039
0.1255	5.9460	6.6394	0.0105	0.0105	0.0040	0.0039
0.2091	5.3396	8.0114	0.0106	0.0105	0.0040	0.0039

Once again, the results are very similar to the results of the comparisons for the small and middle ranges. For large wave numbers, the gain is about 2.7 dB. As compared to the middle and large ranges, this gain has decreased, and the resolution in the azimuth direction has increased for both methods. Figure 10-11 shows the slices of the amplitude spectrum in the range and azimuth directions. Figure 10-12 shows $(S/N)_{CM}$ and $(S/N)_{PFT}$ as a function of k. It can be seen that the plots are very similar to the plots of the other considered ranges and therefore they are not discussed further.

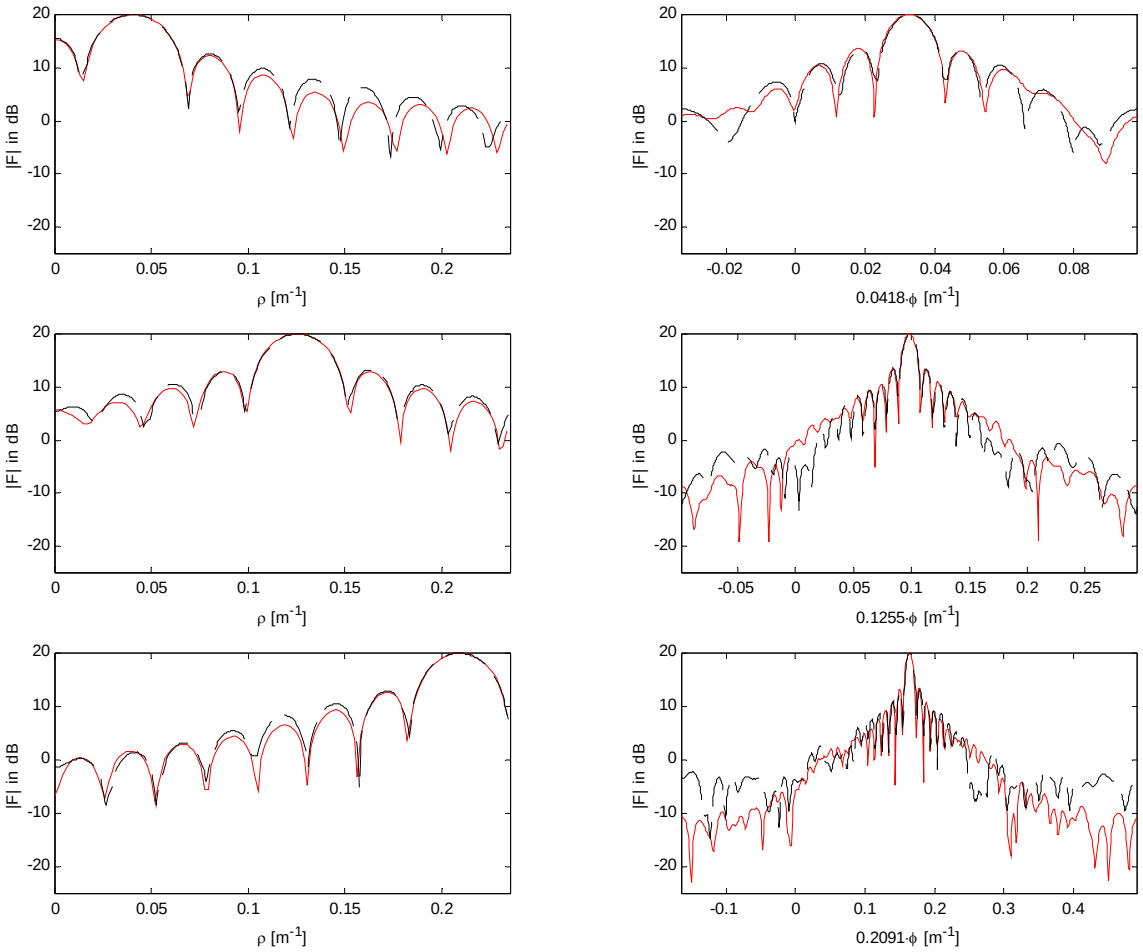

Figure 10-11 Slices of amplitude spectrum for large range, no windowing

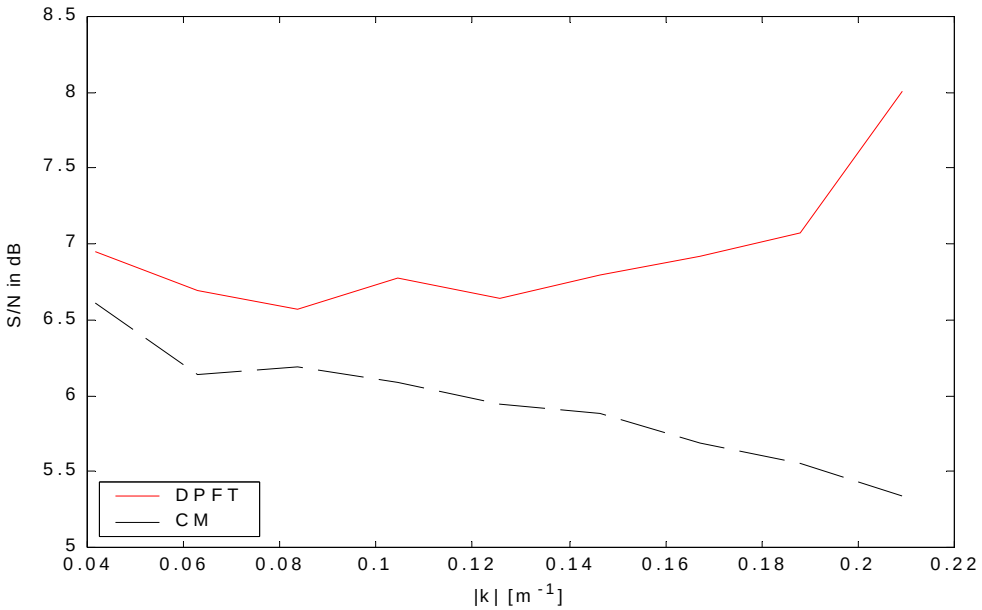

Figure 10-12 Signal-to-noise ratios of DPFT and CM as a function of (large range, no windowing)

Table 6 lists the results for three values of k in the allowed range when Hanning windowing is applied to the data.

Table 6 Comparison results for large range with Hanning windowing

$k = \lvert \vec{k} \rvert$	$(S/N)_{CM}$	$(S/N)_{PFT}$	$(EW_\rho)_{CM}$	$(EW_\rho)_{PFT}$	$(EW_\phi)_{CM}$	$(EW_\phi)_{PFT}$
0.0418	19.1469	19.0769	0.0174	0.0203	0.0070	0.0066
0.1255	17.9642	26.3254	0.0188	0.0191	0.0071	0.0072
0.2091	13.9388	27.4520	0.0189	0.0171	0.0072	0.0072

For large wave numbers, the gain is now about 13 dB. Again, as compared to the middle and large ranges, this gain has decreased. Figure 10-13 shows the slices of the amplitude spectrum in the range and azimuth directions and Figure 10-14 shows the signal-to-noise ratios as a function of k. Again, the plots are similar to the plots of the other considered ranges and are not discussed further.

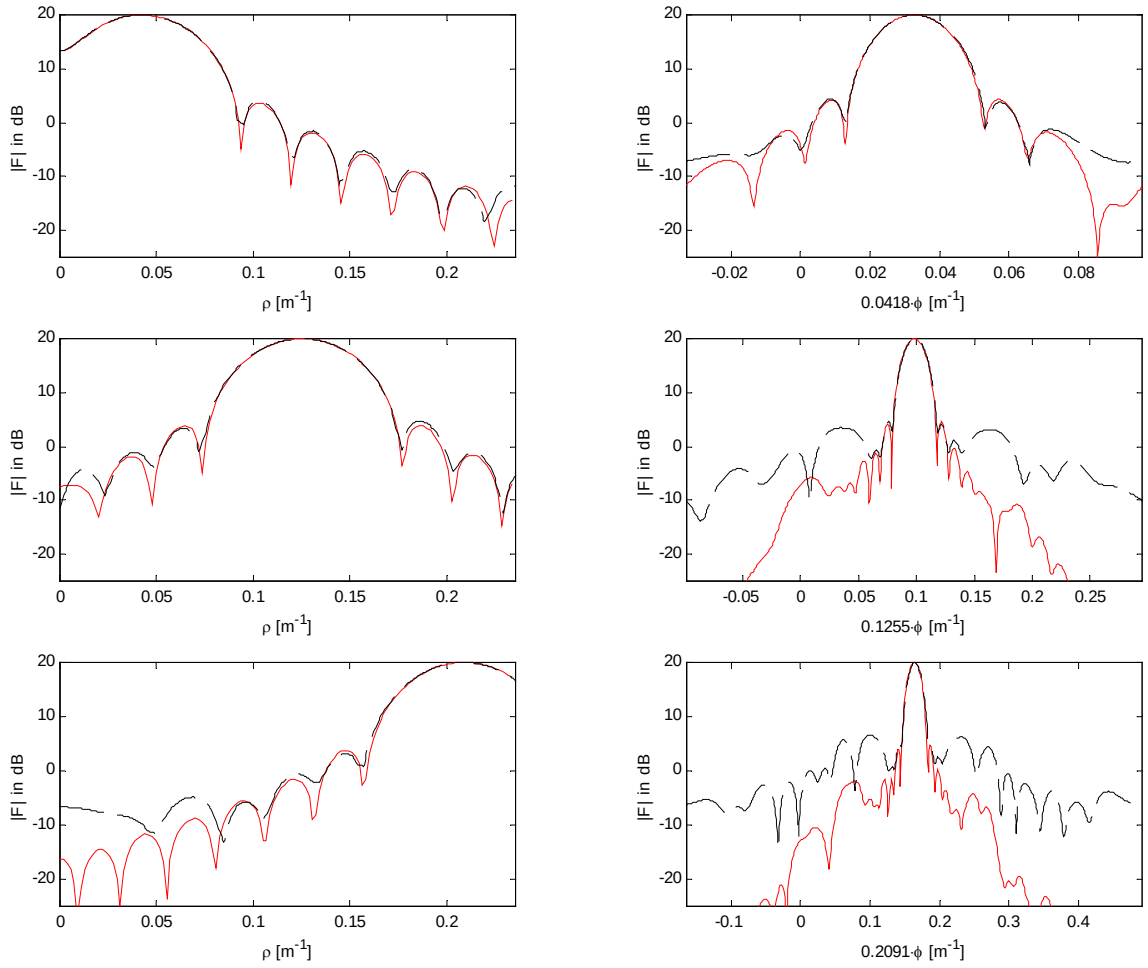

Figure 10-13 Slices of amplitude spectrum for large range, Hanning windowing

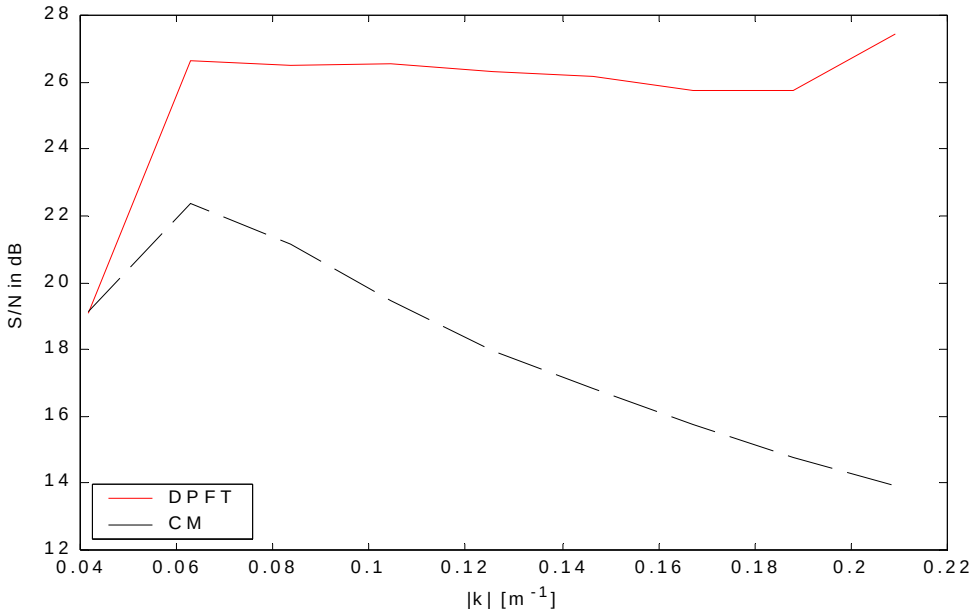

Figure 10-14 Signal-to-noise ratios of DPFT and CM as a function of $k = |\vec{k}|$ *(large range, Hanning windowing)*

11 Conclusions and recommendations regarding the spectral analysis

The second part of the project concerned the spectral analysis of the data obtained with SHIRA. The data is available on a polar grid that is nonuniform in the azimuth (angular) direction only. At present, the individual images out of a time series are first scan converted, i.e. they are resampled from the nonuniform polar grid onto a Cartesian grid by means of the Nearest Neighborhood (NN) interpolation method, and next the two-dimensional FFT is computed. However, TNO-FEL observed that the spectra obtained this way seemed to be distorted, especially for higher spatial frequencies. They expected that this distortion was caused by the Nearest Neighborhood interpolation of the available polar samples onto the Cartesian grid. This expectation has been examined and it was found that indeed some distortion was present in the spectra for high spatial frequencies (i.e. large wave numbers). However, due to the fact that the data is (heavily) oversampled for most wave number values in a large part of the data window, this effect only seems to be significant for large wave numbers. It is possible to eliminate the distortion due to the NN interpolation totally because methods exist that ideally reconstruct two-dimensional signals from nonuniform samples in polar coordinates if the function is band limited. However, this involves an unacceptably high complexity.

In order to try to diminish the distortion and to obtain better spectra in general, an alternative method named Discrete Polar Fourier Transform (DPFT) has been examined and its performance regarding the quality of the spectra has been compared to the currently used method. The DPFT is defined (here) as the Fourier transform that takes polar input data and produces polar output data. This method has been chosen because it intuitively seems better to directly use the polar input data instead of first converting them to Cartesian data (this way errors in the resampling process are avoided as much as possible). With the DPFT, the scan conversion step is eliminated and the two-dimensional Cartesian Fourier transformation of each image in a time series is replaced by the DPFT. A number of spectra (see for example equations (7-9), (7-10) and (7-11)) require that the data is available on a polar grid in the Fourier domain. With the current method, the Fourier data is available on a Cartesian grid and must therefore be resampled onto a polar grid. Hence, again distortions are introduced in the polar Fourier spectra due to the resampling, and these distortions propagate to the spectra computed by the mentioned equations. For this reason, it is desired that the Fourier data is directly available on a polar grid.

The formula for the continuous PFT has been discretized in order to obtain a formula for the DPFT. The direct computation of this DPFT-formula for all images in a time series takes a lot of time. Within the scope of this report, the computational complexity of the DPFT is of less importance. However, since the efficient computation will be important if the DPFT is to be used in practice (in case the comparison between the current method and the DPFT reveals that the DPFT performs better, and TNO-FEL decides to use the DPFT), and also for the simulations, an attempt has been made to compute the DPFT in a way similar to FFT algorithms. However, in agreement with other authors that claim that no FFT for the PFT is available [Briggs and Henson, 1995], it must be concluded that *no FFT like algorithm to compute the PFT currently exists or is likely to be invented in the future*. The properties of the kernel in the Fourier transform that are necessary for the development of the FFT are not found in the kernel of the PFT due to the sine term in the imaginary exponent of the kernel. However, a method for indirectly computing the DPFT via Hankel transforms that seems promising has been found in the literature [Stark, 1979]. With this method, the DPFT of a function that is sampled on a uniform polar grid is written as a Fourier series (DFT) in the angle in the Fourier plane. The fact that the data must be available on a uniform polar grid implies that a one-dimensional interpolation in the azimuth direction is required, instead of the two-dimensional interpolation with the current method (which introduces more errors). The coefficients of the Fourier series are called the circular harmonics of the DPFT and they are given by the n-th order discrete Hankel transforms of the circular harmonics of the input data function, which in turn are the DFT coefficients of the input data function for each constant radius (see equations (8-18), (8-23) and (8-24) and Figure 8-3). The described method seems promising because methods for the fast evaluation of Hankel transforms are reported in the literature (in this case ‘fast’ means that the Hankel transforms can be computed in a

time that is several, for example three, times the time required by an FFT). All methods try to compute a discrete implementation of the continuous Hankel transform and thereby make use of some interpolation scheme somewhere in the processing stage. Because it has been shown in Section 8.3 that the continuous Hankel transform integral may be approximated by a simple summation, it can be expected that the methods found in the literature can be sped up because the simplest interpolation scheme, namely zero-order hold, may be used.
Two Hankel transform methods have been found that can be suitable for SHIRA (see Section 8.4), namely a method that makes use of asymptotic expansions of the Bessel series in the Hankel transform, and a method that relies on the decomposition of the Hankel transform into a sine, a cosine and an inversion transform (using the Mellin approach). Both Hankel transform methods take equispaced data; therefore, at this point use can be made of the fact that the data is equispaced in the range direction. Due to the limited amount of time, these algorithms have not been implemented yet.
It has been shown that if a circular section with a larger azimuth span is analyzed, the complexity of the DFPT computed via fast Hankel transforms increases slower than the complexity of the currently used method with comparable parameters. So, when large areas are analyzed, the new method is probably faster than the old one. This can be a great advantage since the user probably wants to analyze an area as large as possible. Currently, relatively small areas are analyzed because of the computation times. However, this should be tested explicitly in practice with fast implementations of the Hankel transform and this has not been done in the current project.

The comparison between the DPFT and the current method was mainly based on the amount of distortion in the spectra computed with both methods. This amount was measured by means of a signal-to-noise ratio (see Chapter 9).
It has been shown that (in the absence of noise) the DPFT performs better than the current method for all ranges, especially when windows are used to smoothly taper the data to zero at the edges of the data window. However, the price paid by both the DPFT and the current method for the better signal-to-noise ratios in case windowing is used, is a reduced spectral resolution. In all cases (with and without windowing), most profit was obtained for large wave numbers.
In case windowing is used, the quality of the DPFT on a slice in the azimuth direction (see Chapter 9) is much better than the quality of the current method on that specific slice. From the obtained results, it must be concluded that windowing in the range and azimuth directions in case of the DPFT is better suited to the problem than windowing in Cartesian directions in case of the conventional method. However, no satisfying explanation of this phenomenon has been found yet and hence more research is required. Another rather remarkable result that followed from the comparisons is that the signal-to-noise ratio of the DPFT is rather independent of the modulus of the wave number, while the signal-to-noise ratio of the conventional method decreases with increasing wave numbers.
In addition to the signal-to-noise ratios, a comparison has been made between the spectral resolutions of both methods. It has been shown that the spectral resolution of the DPFT is approximately the same to that of the current method. Also the locations of the wave numbers (peaks) in the frequency domain have been checked and it was found that both methods correctly preserved the locations. Likewise, it has been tested whether the amplitude ratio of two plane wave components was preserved correctly during the transformations and this proved to be true also.
In conclusion, it can be stated that the DPFT is better suited to the SHIRA signal analysis problem than the conventional method in case no noise is present. Due to the limited amount of time, no simulations where the radar data was contaminated by noise have been carried out. Therefore, more simulations are required to investigate whether the DPFT still performs better when noise is present. Besides better spectra, the DPFT has the advantage that the data window is used as efficiently as possible.

References

[Abramowitz and Stegun, 1965]
Abramowitz, M. and Stegun, I.A., "Handbook of mathematical functions with formulas, graphs and mathematical tables", New York, Dover Publications, 1965.

[Addi-Data, 1999]
Addi-Data, " Technical description ADDICOUNT PA 1700-2 High-speed counter board", Addi-Data GmbH, Ottersweier, Germany, 1999.

[Baker, 1997]
Baker, A., "The Windows NT device driver book – A guide for programmers", Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1997.

[Bland, Laakso and Tarczynski]
Bland, D.M., Laakso, T.I. and Tarczynski, A., "Analysis of algorithms for nonuniform-time discrete Fourier transform", 1996, pp. 453-456.

[Briggs and Henson, 1995]
Briggs, W.L. and Henson, V.E., "The DFT – An owner's manual for the Discrete Fourier Transform", Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1995.

[Candel, 1981]
Candel, S.M., "Dual algorithms for fast calculation of the Fourier-Bessel transform", IEEE Transactions on Acoustics, speech and signal processing, Vol. ASSP-29, 1981, pp. 963-972.

[Cree and Bones, 1993]
Cree, M.J. and Bones, P.J., "Algorithms to numerically evaluate the Hankel transform", Comp. Math. Applic., Vol. 26, No. 1, 1993, pp. 1-12.

[Datum, 1996]
Datum, "bc630AT Real Time Clock Module – User's Guide", Datum Inc., San Jose, USA, 1996.

[Dekker and Newcomer, 1999]
Dekker, E.N. and Newcomer, J.M., "Developing Windows NT device drivers – A programmer's handbook", Addison-Wesley, Harlow, England, 1999.

[Dudgeon and Mersereau, 1984]
Dudgeon, D.E., Mersereau, R.M., "Multi-dimensional digital signal processing", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[Dutt and Rokhlin, 1993]
Dutt, A. and Rokhlin, V., "Fast Fourier transforms for nonequispaced data", SIAM J. Sci. Comput., Vol. 14, Nov. 1993, No. 6, pp. 1368-1393.

[Fraser, 1989]
Fraser, D., "Interpolation by the FFT revisited – An experimental investigation", IEEE Transactions on Acoustics, speech and signal processing, Vol. ASSP-37, May 1989, No. 5, pp. 665-675.

[Higgings and Munson, 1987]
Higgings, W.E. and Munson, D.C., "An algorithm for computing general integer-order Hankel transforms", IEEE Transactions on Acoustics, speech and signal processing, Vol. ASSP-35, 1987, pp. 86-97.

[Higgings and Munson, 1988]

Higgings, W.E. and Munson, D.C., “A Hankel transform approach to tomographic image reconstruction”, IEEE Trans. Med. Imag., Vol. 7, 1988, pp. 59-72.

[ICS650 Operating Manual]
Interactive Circuits And Systems Ltd., “ICS650 Operating Manual, ICS Ltd.”, Gloucester, Ontario, Canada.

[ICS650 Software Developers Kit User’s Manual]
Interactive Circuits And Systems Ltd., “ICS650 Software Developers Kit User’s Manual”, ICS Ltd., Gloucester, Ontario, Canada.

[Johnson, 1982]
Johnson, R.L., “Numerical methods, a software approach”, New York, John Wiley & Sons, 1982.

[Kleijweg and Greidanus, 1994]
Kleijweg, J.C.M. and Greidanus, H., “Wave and Current Measurements with a Navigation radar”, TNO Physics and Electronics Laboratory, The Hague, the Netherlands, 1994.

[Kleijweg, 1991]
Kleijweg, J.C.M., “SHIRA systeembeschrijving en eerste meetresultaten”, Report, TNO Physics and Electronics Laboratory, The Hague, the Netherlands, 1991.

[Knockaert, 2000]
Knockaert, L., “Fast Hankel transform by fast sine and cosine transforms: the Mellin connection”, IEEE Transactions on Signal Processing, Vol. 48, June 2000, No. 6, pp. 1695-1701.

[Laar, 1999]
Laar, J. van de, “Proposal for the implementation of a flexible radar data acquisition system”, Tech 5 B.V., Hardinxveld-Giessendam, The Netherlands,1999.

[Lynn, 1987]
Lynn, P.A., “Radar Systems”, Macmillan Education Ltd., London, U.K., 1987.

[Marks II, 1993]
Marks II, R.J., Advanced topics in Shannon sampling and interpolation theory, Berlin, Springer-Verlag, 1993.

[Marvasti, 1990]
Marvasti, F.A., “Reconstruction of 2-D signals from nonuniform samples in polar coordinates”, IEEE Transactions on circuits and systems, Vol. 37, April 1990, No. 4., pp. 576-568.

[Nguyen and Liu, 1999]
Nguyen, N. and Liu, Q.H., “The regular Fourier matrices and nonuniform fast Fourier transfomrs”, Vol. 21, 1999, No. 1, pp. 283-293.

[Oppenheim et al., 1978]
Oppenheim, A.V., Frisk, G.V. and Martinez, D.R., “An algorithm for the numerical evaluation of the Hankel transforms”, Proc. IEEE, Vol. 66, 1978, pp. 264-265.

[Oppenheim et al., 1980]
Oppenheim, A.V., Frisk, G.V. and Martinez, D.R., “Computation of Hankel transforms using projections”, J. Acoust. Soc. Am., Vol. 68, 1980, pp. 523-529.

[Piessens, 1982]

Piessens, R., “Automatic computation of Bessel function integrals”, Comp. Phys. Comm., Vol. 25, 1982, pp. 289-295.

[Seemann and Ziemer, 1995]
Seemann, J. and Ziemer, F., Computer simulation of imaging ocean wave fields with marine radar, Oceans’95, MTS/IEEE ‘Challenges of our changing global environment’, San Diego, pp. 1128-1133, 1995.

[Senet, Seemann and Ziemer, 1997]
Senet, C.M., Seemann, J. and Ziemer, F., An iterative technique to determine the near surface current velocity from time series of sea surface images, Oceans’97, ‘500 Years of ocean explorations’, Halifax, Canada, Oct. 1997.

[Sharafeddin et al., 1992]
Sharafeddin, O.A., Bowen, H.F., Kouri, D.J. and Hoffman, D.K., “Numerical evaluation of spherical Bessel transforms via fast Fourier transforms”, J. Comp. Phys., Vol. 100, 1992, pp. 294-296.

[Siegman, 1977]
Siegman, A.E., “Quasi fast Hankel transform”, Opt. Lett., Vol. 1, 1977, pp. 13-15.

[Sneddon, 1955]
Sneddon, I.N., Fourier Transforms, First Edition, Second Impression, McGraw-Hill Book Company, Inc., New York, Toronto, London, 1955.

[Stark, 1979]
Stark, H., “Sampling theorems in polar coordinates”, J. Opt. Soc. Am., Vol. 69, Nov. 1979, No. 11, pp. 1519-1524.

[Suter, 1991]
Suter, B.W., “Fast N-th order Hankel transform algorithm”, IEEE Transactions on Acoustics, speech and signal processing, Vol. ASSP-39, Feb. 1991, pp. 532-536.

[Talman, 1978]
Talman, J.P., “Numerical Fourier and Bessel transforms in logarithmic variables”, J. Comp. Phys., Vol. 29, 1978, pp. 35-48.

[Ware, 1998]
Ware, A.F., “Fast approximate Fourier transforms for irregularly spaced data”, SIAM Rev., Vol. 40, Dec. 1998, No. 4, pp. 838-856.

[Watson, 1966]
Watson, G.N., “Theory of Bessel functions”, Second Edition, The Syndics of the Cambridge University Press, London, 1966.

[Zakhor and Alvstad, 1992]
Zakhor, A. and Alvstad, G., “Two-dimensional polynomial interpolation from nonuniform samples”, IEEE Transactions on Signal Processing, Vol. 40, Jan. 1992, No. 1, pp. 169-180.

Appendix A. Parameter definitions for data acquisition

Parameter (unit)	Definition / meaning
PRF (Hz)	Pulse repetition frequency, i.e. the frequency at which the radar emits pulses
PRI = 1/PRF (s)	Pulse repetition interval, i.e. the reciprocal of the PRF
f_s (Hz)	Sampling frequency of the AD converter(s) on the data acquisition board
τ or tau (s)	Pulse width of the emitted pulses
T_R (s)	Rotation time of the radar antenna
N_{AE}	Number of pulses from the angular encoder per revolution of the radar
2D recording window	Two-dimensional part of the surface to be scanned
N_{SC}	Number of consecutive 2D recording windows in one measurement
T_M (s or min)	Measurement / recording time of an experiment
L (sweeps)	Length of host memory buffer
N_S (sweeps)	Number of sweeps in a measurement
N_M (sweeps)	Number of sweeps to be recorded in one revolution of the radar antenna
N_{RW} (samples)	Number of samples per sweep
S_R (samples or m)	Start range of the recording window
E_R (samples or m)	End range of the recording window
S_A (sweeps)	Start azimuth of the 2D recording window (the first sweep)
E_A (sweeps)	End azimuth of the 2D recording window (the last sweep)
ΔR (m)	Depth of a range bin

Appendix B. Specifications of the SHIRA data acquisition system

	Minimal value	Nominal value	Maximal value	Unit
PRF	600	1200	4000	Hz
f_S		40		MHz
N_{AE}		4096		pulses
T_R	4	1.5	1.2	s
N_{RW}	256 (960 m)	2048 (7680 m)	4096 (15360 m)	Samples or m
S_R	32 (120 m)	267 (1000 m)	3581 (13429 m)	Samples or m
N_M	256	400	2000	Sweeps
N_{SC}		256		scans

Appendix C. Parameter definitions used in discussion of spectral analysis and m-files

Parameter (m-file name)	Definition / meaning
N_θ (Nth)	Number of sweeps (pulse returns) in the data window of PFT
N_r (Nr)	Number of ranges in the data window of PFT
N_a (Na)	Nmber of angle bins in one revolution of the radar antenna
N_Φ (Nphi)	Nmber of angle bins in (polar) Fourier domain
N_ρ (Nrho)	Number of ranges in (polar) Fourier domain
Δr (DeltaR)	Sampling distance in the range direction
θ (th)	Uniformly spaced angles in data window
θ_{NU} (th_NU)	Nonuniformly spaced angles in analysis window
r_0 (r0)	Start (minimum) range of data window of PFT (circular section)
R	End (maximum) range of data window of PFT
r_P (rP)	The distance of the centroid P of the data windows to the origin
θ_0 (th_0)	Start (minimum) azimuth of data window of PFT
θ_0 (th_e)	End (maximum) azimuth of data window of PFT
θ_{rot} (th_rot)	The angle of the centroid P with the positive x-axis
A_d (Ad)	Maximum deviation of angular velocity due to wind, etc.
inpmeth	Interpolation method
P_{cx} (Pcx)	Center of mass of data window in x-direction
P_{cy} (Pcy)	Center of mass of data window in y-direction
window_r	Window for range direction
window_th	Window for azimuth direction
k_{x1} (kx1)	x-component of wave number of the sine component in the first quadrant
k_{y1} (ky1)	y-component of wave number of the sine component in the first quadrant
k_{x2} (kx2)	x-component of wave number of the sine component in the second quadrant
k_{y2} (ky2)	x-component of wave number of the sine component in the second quadrant
A	Amplitude of the sine component in the second quadrant (amplitude in first quadrant is 1)
N_x (Nx)	Number of sample (grid) points in x-direction of rectangular data window of conventional method
N_y(Ny	Number of sample (grid) points in y-irection of rectangular data window of conventional method
r_x (rx)	Sample distance (grid size) in x-direction of rectangular data window of conventional method
r_y (ry)	Sample distance (grid size) in y-direction of rectangular data window of conventional method

Appendix D. Example of file that defines the input parameters for the simulations

```
% Input parameters

%clear;
format compact;
% Parameters for nonuniformly spaced angles in spatial domain
Nth=64              % Number of sweeps in data-window
th_0=0.57           % Start angle of data, i.e. angle of the first sweep, in radians
Na=1024             % Number of sweeps in one revolution of the radar antenna
DeltaTh=2*pi/Na;    % Mean or nominal spacing between the angles of the sweeps
Ad=25               % Maximum angle (angular velocity) deviation in percent
% Parameters for uniformly spaced ranges in spatial domain
Nr=64               % Number of ranges in data-window
r0=500              % Start range of data in meters
DeltaR=3.75         % Sample distance

% Parameters for uniformly spaced ranges and angles in Fourier domain
Nrho=128            % Number of ranges in Fourier domain
Nphi=Na             % Number of sweeps in Fourier domain

% Wavenumber of first plane wave component (normally in first quadrant)
MaxK=0.54;
modk=MaxK*[1:11]/11;
kx1=sqrt(1/2)*modk(10)
ky1=kx1
% Wavenumber of second plane wave component (normally in second quadrant)
%th_diff=pi/35;
kx2=0
ky2=0
%kx2=kx1*cos(th_diff)-ky1*sin(th_diff)
%ky2=kx1*sin(th_diff)+ky1*cos(th_diff)
A2=0%.6         % Amplitude of the second plane wave component

% Choose interpolation method
inpmeth='nearest';      % Nearest neighbor interpolation
%inpmeth='linear';      % Bilinear interpolation
%inpmeth='cubic';       % Bicubic interpolation
%inpmeth='spline';      % Spline interpolation

% Windows that can be used: NONE, BARTLETT, BLACKMAN, BOXCAR, CHEBWIN, HANNING, KAISER and TRIANG
window_r_x='none'; % Window for tapering in range direction (PFT) cq. x-direction (CM)
window_th_y='none';     % Window for tapering in azimuth direction (PFT) cq. y-direction (CM)
```

Appendix E. GenInpData: m-file that simulates a plane wave input field on a polar grid

```
% This file generates the polar input data with nonuniformly
% spaced angles and uniformly spaced ranges

% The following parameters have been defined in the input parameters file:
%  Parameters for nonuniformly spaced angles in spatial domain
%     Nth:          Number of sweeps in data window
%     th_0:         Start angle of data, i.e. angle of the first sweep, in radians
%     Na:           Number of angle bins in one revolution of the radar antenna
%     Ad:           Maximum angle (angular velocity) deviation in percent
%  Parameters for uniformly spaced ranges in spatial domain
%     Nr:           Number of ranges in data window
%     r0:           Start range of data in meters
%     DeltaR:           Sampling distance
%  Parameters for uniformly spaced ranges and angles in Fourier domain
%     Nrho:         Number of ranges in Fourier domain
%     Nphi:         Number of angle bins in Fourier domain
%  Wavenumber of first plane wave component (normally in first quadrant)
%     kx1:          x-component of wavenumber of the first plane wave component
%     ky1:          y-component of wavenumber of the first plane wave component
%  Wavenumber of second plane wave component (normally in second quadrant)
%     kx2:          x-component of wavenumber of the second plane wave component
%     ky2:          y-component of wavenumber of the second plane wave component
%     A:            Amplitude of the second plane wave component
%  inpmeth:      Interpolation method
%  window_r:     Window for range direction
%  window_th:    Window for azimuth direction

% First, compute some derived parameters of the area that is associated with the
% discrete sample locations (circular section)
r_max=r0+Nr*DeltaR-DeltaR/2;            % Maximum range
r_min=r0-DeltaR/2;                      % Minimum range
th_max=th_0+Nth*2*pi/Na-pi/Na;          % Maximum angle
th_min=th_0-pi/Na;                      % Minimum angle

% Define data window input coordinates
r=r0+DeltaR*[0:Nr-1];      % Define uniform intput ranges
% Define nonuniform input data angles
rand('seed',0);                % Initialize random generator with seed
AngNoise = (Ad/100)*DeltaTh*(2*(rand(1,Nth+2)-0.5));         % Maximum angle deviation is 25 %, take 2 extra sweeps
th_NU = th_0+DeltaTh*[0:Nth+1]-DeltaTh + AngNoise;         % Nonuniform angles, 2 extra sweeps

% Define polar grid with nonuniformly spaced angles and uniformly spaced ranges:
[R_NU,TH_NU]=meshgrid(r,th_NU);

% Input wavenumbers have to satisfy the Nyquist criterion
modk1=sqrt(kx1^2+ky1^2);
argk1=atan2(kx1,ky1);           % Angle of k1 with respect to the positive ky axis
modk2=sqrt(kx2^2+ky2^2);
argk2=atan2(kx2,ky2);           % Angle of k2 with respect to the positive ky axis
% Check whether Nyquist criterion is satisfied
maxk=min((pi/(2*r_max*sin((1+Ad/100)*pi/Na))),pi/DeltaR)
if ( (modk1 >= maxk) | (modk2 >= maxk) )
%   error('|k| must be smaller than pi/(2*r_max*sin((1+Ad/100)*pi/Na)) and pi/DeltaR!!!');
   disp('|k| must be smaller than pi/(2*r_max*sin((1+Ad/100)*pi/Na)) and pi/DeltaR!!!');
end
% Redefine rho_max in order to fairly compare the PFT to the conventional method (equal areas in freq. domain)
rho_max=maxk

% Simulate data window (first index is theta, second index is r)
% For the quality measurements, it is assumed that the highest peak is in the first
% (and third) quadrant and the lowest in the second (and fourth).
g_r_th_nu=sin(kx1*R_NU.*cos(TH_NU) + ky1*R_NU.*sin(TH_NU)) + A2*sin(kx2*R_NU.*cos(TH_NU) +
ky2*R_NU.*sin(TH_NU));
% Plot data
[xnu,ynu]=pol2cart(TH_NU,R_NU);
%plot(xnu,ynu,'k.');axis('equal');xlabel('x'); ylabel('y');title('Nonuniform polar input grid');pause;close;
%surf(xnu,ynu,g_r_th_nu);xlabel('x'); ylabel('y');title('Nonuniform polar input data');
%pause;view(2);axis('equal');pause;close;

if (~exist('Nx') & ~exist('Ny'))
```

```
   Nx=Nr;
   Ny=Nth;
end;

% Prepare tapering windows
if ~(strcmp(window_r_x,'none'))
   wind_r_x=eval([window_r_x,'(',num2str(Nx),')']);          % Window for range or x direction
   wind_r_x=repmat(wind_r_x',Ny,1);
end;
if ~(strcmp(window_th_y,'none'))
   wind_th_y=eval([window_th_y,'(',num2str(Ny),')']);        % Window for azimuth or y direction
   wind_th_y=repmat(wind_th_y,1,Nx);
end;
```

Appendix F. CurrMeth.m: m-file that simulates spectral analysis by means of the current method

```
% Currently used method for computing the Fourier transform of polar data:
%   * Generate a uniform rectangular grid for the 2D FFT, i.e. select the
%     location (Pcx,Pcy) and size (Nx,rx,Ny,ry) of the analysis rectangle
%   * Convert rectangular coordinates to polar coordinates
%   * Interpolate given data to this polar grid (interpolation in polar domain)
%   * Fourier transformation, filtering, etc.
% Alternative method (takes more time):
%   * Generate a uniform rectangular grid for the 2D FFT
%   * Convert nonuniform polar coordinates to uniform rectangular coordinates
%   * Interpolate given data to this rectangular grid (interpolation in
%     cartesian domain)
%   * Fourier transformation, filtering, etc.

clear;format compact;      % Prepare environment
paramfile='CurrParams1';       % Name of file that contains the input parameters
% Get input parameters from the file specified in the previous line
eval(paramfile);
sx=Nx*rx;
sy=Ny*ry;
% Construct grid for rectangle
xx=rx*[-Nx/2:Nx/2-1]+Pcx+rx/2;
yy=ry*[-Ny/2:Ny/2-1]+Pcy+ry/2;
[xb,yb]=meshgrid(xx,yy);

% Define Nth, th0, Nr, r0 in order to select a circular section that encompasses the rectangle
th_min=thmin(Pcx,Pcy,sx,sy);
th_max=thmax(Pcx,Pcy,sx,sy);
r_min=rmin(Pcx,Pcy,sx,sy);
r_max=rmax(Pcx,Pcy,sx,sy);
th_0=th_min+pi/Na;
Nth=ceil((th_max-th_min)/DeltaTh);
r0=r_min+DeltaR/2;
Nr=ceil((r_max-r_min)/DeltaR);

% For an overview of the parameter and variable names, see the m-file
% GenInpData, which is called next.
GenInpData3;% Generate the polar input data, that is nonuniform in the azimuth direction
% The simulated data are now available in g_r_th_nu (first index is theta, second index is r)

% Convert uniform rectangular grid to polar coordinates
t_start=clock;
[thi,ri]=cart2pol(xb,yb);
t_cart2pol=etime(clock,t_start);
disp(['Conversion of uniform rectangular grid to polar coordinates took ',num2str(t_cart2pol),' seconds.']);

% Plot comparison areas
[xp,yp]=pol2cart(TH_NU,R_NU);
%plot(xp,yp,'y.');hold on;plot(xb,yb,'.');axis('equal');
%xlabel('x');ylabel('y');title('Areas analyzed by conventional method (rectangle) and PFT (circular section)');pause;close;

% 2D interpolation of polar data onto uniform cartesian grid
t_start=clock;
g_xy=interp2(R_NU,TH_NU,g_r_th_nu,ri,thi,inpmeth);
t_2D_intp=etime(clock,t_start);
disp(['2D ',inpmeth,' interpolation in polar domain took ',num2str(t_2D_intp),' seconds.']);
%surf(xb,yb,g_xy); xlabel('x');ylabel('y');title('Interpolated data in selected window');
%pause;view(2);axis('equal');pause;

% Now, multiply g_xy by windows in both the x and y directions
t_start=clock;   % Measure windowing time
if ~(strcmp(window_r_x,'none'))
   g_xy=g_xy.*wind_r_x;
end;
if ~(strcmp(window_th_y,'none'))
   g_xy=g_xy.*wind_th_y;
end;
t_windowing=etime(clock,t_start);
disp(['Windowing took ',num2str(t_windowing),' seconds.']);
%surf(xb,yb,g_xy); xlabel('x');ylabel('y');title('Interpolated and windowed data in selected window');
%pause;view(2);axis('equal');pause;
```

```
% Compute 2D FFT
t_start=clock;
F=fftshift(fft2(g_xy));
t_FFT=etime(clock,t_start);
disp(['Computation of 2D cartesian FFT took ',num2str(t_FFT),' seconds.']);

% Display total computation time
tot_comp_time=t_cart2pol+t_2D_intp+t_FFT;
disp(['Total computation took ',num2str(tot_comp_time),' seconds.',sprintf('\n')]);

% Construct grid for rectangle in Fourier domain
Nkx=Nx;
Nky=Ny;
kxx=(2*pi/Nkx)*[-Nkx/2:Nkx/2-1]/rx;
kyy=(2*pi/Nky)*[-Nky/2:Nky/2-1]/ry;
[kxb,kyb]=meshgrid(kxx,kyy);  % x and y of rectangle before rotation
Fkxky=F;

% Display amplitude spectrum
surf(kxb,kyb,abs(Fkxky));xlabel('k_x'); ylabel('k_y');zlabel('|F|');title('Amplitude spectrum');
%pause;view(2);axis('equal');pause;
%surf(kxb,kyb,10*log10(abs(Fkxky)));xlabel('k_x'); ylabel('k_y');zlabel('|F| (dB)');title('Amplitude spectrum in dB');
%pause;view(2);axis('equal');pause;close;
```

Appendix G. PolFftD: m-file that computes the DPFT directly

```
% PolFftD.m
% Polar Fourier Transform calculated directly

clear;
format compact;
paramfile='ParamsSmRng';    % Name of file that contains the input parameters
% Get input parameters from file
eval(paramfile);

% For an overview of the parameter and variable names, see the m-file
% GenInpData.m, which is called next.

GenInpData3;% Generate the polar input data, that is nonuniform in the azimuth direction
% The simulated data are now available in g_r_th_nu (first index is theta, second index is r)

% PFT-specific parameters
th=th_0+DeltaTh*[0:Nth-1];     % Define nominal (equally spaced) input data angles
[R,TH]=meshgrid(r,th);         % Polar grid with angles uniformly distributed

% Do 1D interpolation in azimuth direction in order to obtain the data on a uniform polar grid
t_start=clock;   % Measure interpolation time
g_r_th=interp1(TH_NU(:,1),g_r_th_nu,TH(:,1),inpmeth);
t_1D_intp=etime(clock,t_start);
disp(['1D interpolation in azimuth direction took ',num2str(t_1D_intp),' seconds.']);

% Now, multiply g_r_th by windows in both the range and azimuth directions
t_start=clock;   % Measure windowing time
if ~(strcmp(window_r_x,'none'))
   g_r_th=g_r_th.*wind_r_x;
end;
if ~(strcmp(window_th_y,'none'))
   g_r_th=g_r_th.*wind_th_y;
end;
t_windowing=etime(clock,t_start);
disp(['Windowing took ',num2str(t_windowing),' seconds.']);

% Plot interpolated input data, which are now available on a uniform grid
[x,y]=pol2cart(TH,R);
%surf(x,y,g_r_th);
%xlabel('x');ylabel('y');title('Input data interpolated onto a uniform polar grid');
%pause;view(2);pause;close;

% Define polar grid in Fourier domain, first index is phi, second is rho
phi=(2*pi/Nphi)*[0:Nphi-1];
rho=rho_max/Nrho*[0:Nrho-1];% rho_max, the maximum modulus of the wavenumbers, is computed in GenInpData
[RHO,PHI]=meshgrid(rho,phi);

% Compute DPFT directly
t_start=clock;   % Measure computation time of DPFT
F=zeros(Nphi,Nrho);
j=sqrt(-1);
for l=1:Nrho
   tic
   for m=1:Nphi
      for i=1:Nth
         F(m,l)=F(m,l)+sum(r.*g_r_th(i,:).*exp(-j*(r*rho(l)*sin(th(i)+phi(m)))));
      end;
   end;
   toc
   l
end;
t_DPFT_direct=etime(clock,t_start);
disp(['Direct computation of discrete polar Fourier transform took ',num2str(t_DPFT_direct),' seconds.']);
F=F/(Na*Nr);

% Display total computation time
tot_comp_time=t_1D_intp+t_windowing+t_DPFT_direct;
disp(['Total computation took ',num2str(tot_comp_time),' seconds.',sprintf('\n')]);
```

```
[kxx, kyy]=pol2cart(PHI,RHO);
surf(kxx,kyy,abs(F));
xlabel('k_y');ylabel('k_x');zlabel('|F|');title('Directly computed DPFT');
pause;view(2);axis('equal');pause;
surf(kxx,kyy,10*log10(abs(F)));
xlabel('k_y');ylabel('k_x');zlabel('|F| in dB');title('Directly computed DPFT in dB');
pause;view(2);axis('equal');pause;close;
```

Appendix H. PolFFT: m-file that computes the DPFT indirectly via Hankel transforms

```
% File name: PolFFT.m
% Polar Fourier Transform calculated as follows:
%   Input data --> {FFT} --> Cn(Rk) --> {Hankel transform} --> Cnn(rho[l]) --> {FFT} --> Polar spectrum

%clear;
format compact;
paramfile='ParamsSmRng';     % Name of file that contains the input parameters
% Get input parameters from file
eval(paramfile);

% For an overview of the parameter and variable names, see the m-file
% GenInpData.m, which is called next.

GenInpData;                  % Generate the polar input data, that is nonuniform in the azimuth direction
% The simulated data are now available in g_r_th_nu (first index is theta, second index is r)

% PFT-specific parameters
th=th_0+DeltaTh*[0:Nth-1];   % Define nominal (equally spaced) input data angles
[R,TH]=meshgrid(r,th);       % Polar grid with angles uniformly distributed

% Do 1D interpolation in azimuth direction in order to obtain the data on a uniform polar grid
t_start=clock;               % Measure interpolation time
g_r_th=interp1(TH_NU(:,1),g_r_th_nu,TH(:,1),inpmeth);
t_1D_intp=etime(clock,t_start);
disp(['1D interpolation in azimuth direction took ',num2str(t_1D_intp),' seconds.']);

% Now, multiply g_r_th by windows in both the range and azimuth directions
t_start=clock;               % Measure windowing time
if ~(strcmp(window_r_x,'none'))
   g_r_th=g_r_th.*wind_r_x;
end;
if ~(strcmp(window_th_y,'none'))
   g_r_th=g_r_th.*wind_th_y;
end;
t_windowing=etime(clock,t_start);
disp(['Windowing took ',num2str(t_windowing),' seconds.']);

% Plot interpolated input data, which are now available on a uniform grid
[x,y]=pol2cart(TH,R);
%surf(x,y,g_r_th);
%xlabel('x');ylabel('y');title('Input data interpolated onto a uniform polar grid');
%pause;view(2);pause;close;

% Define input data over entire revolution (i.e. pad g_r_th with zeros)
t_start=clock;               % Measure padding time
f_r_th=zeros(Na,Nr);
if th_0 < 0
   th0_ind=round(mod(th_0,2*pi)/DeltaTh);          % Index for start angle of data
   f_r_th([th0_ind:Na],:)=g_r_th([1:Na-th0_ind+1],:);
   f_r_th([1:Nth-Na+th0_ind-1],:)=g_r_th([Na-th0_ind+2:Nth],:);
else
   th0_ind=round(th_0/DeltaTh);          % Index for start angle of data
   f_r_th(th0_ind+[1:Nth],:)=g_r_th;
end;
t_padding=etime(clock,t_start);
disp(['Padding g_r_th took ',num2str(t_padding),' seconds.']);

% Display padded data
thf=(2*pi/Na)*([0:(Na-1)]);
[RF,THF]=meshgrid(r,thf);
[xf,yf]=pol2cart(THF,RF);
%surf(xf,yf,f_r_th);xlabel('x');ylabel('y');title('(Windowed) Input data padded with zeros');
%pause;view(2);pause;close;

% Define polar grid in Fourier domain, first index is phi, second is rho
phi=(2*pi/Nphi)*[0:Nphi-1];
rho=rho_max/Nrho*[0:Nrho-1];% rho_max, the maximum modulus of the wavenumbers, is computed in GenInpData
[RHO,PHI]=meshgrid(rho,phi);

% Compute discrete Bessel coefficients Jn(n,rho[l],r[k])
```

```
t_start=clock;                      % Measure computation time of discrete Bessel coefficients
if(~exist('Jn_lk'))
  Jn_lk=zeros(Na/2+1,Nrho,Nr);
     thh=(2*pi/Na)*([0:(Na/2-1)]);% Define angles needed for computing the discrete Bessel coefficients
     disp(['Memory for discrete Bessel coefficients allocated...',sprintf('\n'),...
                'Computing Bessel coefficients and Hankel transforms...',sprintf('\n')]);
  for n=1:Na/2+1
    tic
    for l=1:Nrho
      for k=1:Nr
        Jn_lk(n,l,k)=2*sum(cos((n-1)*thh-r(k)*rho(l)*sin(thh)))+((-1)^(n-1))-1;
      end;
    end;
    n
    toc
  end;
  t_bess=etime(clock,t_start);
  disp(['Computation of discrete Bessel coefficients took ',num2str(t_bess),' seconds.']);
end;

% Compute the Fourier coefficients Cn(rk) (circular harmonics) by means of Nr 1D FFT's
t_start=clock;                      % Measure computation time of FFT
Cn_rk=fft(f_r_th,[],1);             % size(Cn_rk) = Na by Nr
t_FFT1=etime(clock,t_start);
disp(['Computation of first FFT took ',num2str(t_FFT1),' seconds.']);

% Compute the coefficients Cnn(rho[l]) by means of Hankel transforms
t_start=clock;                      % Measure computation time of discrete Hankel transforms
Cnn_rho_l=zeros(Na,Nrho);
for n=1:Na/2+1
  for l=1:Nrho
    Cnn_rho_l(n,l)=sum(r.*Cn_rk(n,:).*((reshape(Jn_lk(n,l,:),1,Nr))));
  end;
end;
% Now use the relation J(N-n,x)=((-1)^n)*J(n,x) to compute J(n,x) for n=Na/2+1, Na/2+2,..., Na-1
for n=2:2:Na/2  % Odd orders of Hankel transforms
  Cnn_rho_l(Na-n+2,:)=-conj(Cnn_rho_l(n,:));
end;
for n=3:2:Na/2  % Even orders of Hankel transforms
  Cnn_rho_l(Na-n+2,:)=conj(Cnn_rho_l(n,:));
end;
t_hankel=etime(clock,t_start);
disp(['Computation of discrete Hankel transforms took ',num2str(t_hankel),' seconds.']);

% Compute the Polar Fourier Data F(phi[m],rho[l]) by means of Nrho 1D FFT's (size(F) = Na by Nrho)
t_start=clock;                      % Measure computation time of FFT
F=fft(Cnn_rho_l,[],1);
t_FFT2=etime(clock,t_start);
disp(['Computation of second FFT took ',num2str(t_FFT2),' seconds.']);
F=F/Na;   % Scale F

% Display total computation time
tot_comp_time=t_1D_intp+t_windowing+t_padding+t_bess+t_FFT1+t_hankel+t_FFT2;
disp(['Total computation took ',num2str(tot_comp_time),' seconds.',sprintf('\n')]);

% Display amplitude spectrum of PFT
% Plot data till maxk
[kxx,kyy]=pol2cart(PHI,RHO);
%surf(kxx,kyy,abs(F));
%xlabel('k_y [m^{-1}]');ylabel('k_x [m^{-1}]');zlabel('|F|');title('Amplitude spectrum of PFT');pause;view(2);axis('equal');pause;
%surf(kxx,kyy,10*log10(abs(F)));
%xlabel('k_y [m^{-1}]');ylabel('k_x [m^{-1}]');zlabel('|F| (dB)');title('Amplitude spectrum of PFT in dB');
%pause;view(2);axis('equal');pause;close;

% Now, start the QUALITY MEASUREMENTS
MeasQualPFT7;

%rhof=rho_max/32*[0:32-1];   % rho_max, the maximum modulus of the wavenumbers, is computed in GenInpData
%phif=(2*pi/256)*[0:256-1];
%[RHOf,PHIf]=meshgrid(rhof,phif);
%[kxxf,kyyf]=pol2cart(PHIf,RHOf);
%figure;surf(kxxf,kyyf,abs(F(1:4:Nphi,1:4:Nrho)));
%xlabel('k_y [m^{-1}]');ylabel('k_x [m^{-1}]');zlabel('|F|');title('Amplitude spectrum of PFT');
%axis([-rho_max rho_max -rho_max rho_max 0 1.1*max(max(abs(F)))]);
```

Appendix I. ConvMeth: m-file that simulates spectral analysis by means of the conventional method

```
% Currently used method for computing the Fourier transform of polar data:
%   * Generate a uniform rectangular grid for FFT, i.e. select a rectangle
%   * Convert rectangular coordinates to polar coordinates
%   * Interpolate given data to this polar grid (interpolation in polar domain)
%   * Fourier transformation, filtering, etc.
% Alternative method (takes more time):
%   * Generate a uniform rectangular grid for FFT
%   * Convert non-uniform polar coordinates to uniform rectangular coordinates
%   * Interpolate given data to this rectangular grid (interpolation in
%     cartesian domain)
%   * Fourier transformation, filtering, etc.

clear;format compact;      % Prepare environment
paramfile='ParamsSmRng';     % Name of file that contains the input parameters
% Get input parameters from file
eval(paramfile);
% For an overview of the parameter and variable names, see the m-file
% GenInpData3.m

% Now, the parameters that would be used with the PFT are known. Calculate
% the parameters of the rectangle that coincides with the circular section
% as much as possible.
% Compute r_max, r_min, th_min and th_max from the new parameters Nth, th0, Nr, r0
r_max=r0+Nr*DeltaR-DeltaR/2;              % Maximum range
r_min=r0-DeltaR/2;                        % Minimum range
th_max=th_0+Nth*2*pi/Na-pi/Na;    % Maximum angle
th_min=th_0-pi/Na;                        % Minimum angle
rho_maxorg=min((pi/(2*r_max*sin((1+Ad/100)*pi/Na))),pi/DeltaR)


A=(r_max^2-r_min^2)*(th_max-th_min)/2;       % Area of circular section of PFT counterpart
Pcx=(r_max^3-r_min^3)*(sin(th_max)-sin(th_min))/(3*A);
Pcy=(r_max^3-r_min^3)*(cos(th_min)-cos(th_max))/(3*A);
th_rot=(th_min+th_max)/2;                          % Compute rotation angle (or: atan2(Pcy,Pcx))
rP=sqrt(Pcx^2+Pcy^2);
Nx=Nr;
Ny=Nth;
rx=DeltaR;
ry=pi*(r_min+r_max)/Na;
sx=Nx*rx;
sy=Ny*ry;
% Construct grid for rectangle
xx=rx*[0:Nx-1]+rP-sx/2+rx/2;
yy=ry*[0:Ny-1]-sy/2+ry/2;
[xb,yb]=meshgrid(xx,yy);  % x and y of rectangle before rotation
xr=xb*cos(th_rot)-yb*sin(th_rot);% x and y of rectangle after rotation
yr=xb*sin(th_rot)+yb*cos(th_rot);

% Re-define Nth, th0, Nr, r0 in order to select a circular section that encompasses the rectangle
th_min=atan2(-sy/2,rP-sx/2)+th_rot;
th_max=atan2(sy/2,rP-sx/2)+th_rot;
r_min=rP-sx/2;
r_max=sqrt(((rP+sx/2)*cos(th_rot)+sy/2*sin(th_rot))^2+((rP+sx/2)*sin(th_rot)-sy/2*cos(th_rot))^2);
th_0=th_min+pi/Na;
Nth=ceil((th_max-th_min)/DeltaTh);
r0=r_min+DeltaR/2;
Nr=ceil((r_max-r_min)/DeltaR);

% For an overview of the parameter and variable names, see the m-file
% GenInpData, which is called next.
GenInpData3;% Generate the polar input data, that is nonuniform in the azimuth direction
% The simulated data are now available in g_r_th_nu (first index is theta, second index is r)

% Convert uniform rectangular grid to polar coordinates

t_start=clock;
[thi,ri]=cart2pol(xr,yr);
thi=mod(thi,2*pi);
t_cart2pol=etime(clock,t_start);
disp(['Conversion of uniform rectangular grid to polar coordinates took ',num2str(t_cart2pol),' seconds.']);
```

```
% Plot comparison areas
[xp,yp]=pol2cart(TH_NU,R_NU);
%plot(xp,yp,'y.');hold on;plot(xr,yr,'.');axis('equal');
%xlabel('x');ylabel('y');title('Areas analyzed by conventional method (rectangle) and PFT (circular section)');pause;close;
t_start=clock;
g_xy=interp2(R_NU,TH_NU,g_r_th_nu,ri,thi,inpmeth);
t_2D_intp=etime(clock,t_start);
disp(['2D ',inpmeth,' interpolation in polar domain took ',num2str(t_2D_intp),' seconds.']);
%surf(xr,yr,g_xy); xlabel('x');ylabel('y');title('Interpolated data in selected window');
%pause;view(2);axis('equal');pause;

% Now, multiply g_xy by windows in both the x and y directions
t_start=clock;   % Measure windowing time
if ~(strcmp(window_r_x,'none'))
   g_xy=g_xy.*wind_r_x;
end;
if ~(strcmp(window_th_y,'none'))
   g_xy=g_xy.*wind_th_y;
end;
t_windowing=etime(clock,t_start);
disp(['Windowing took ',num2str(t_windowing),' seconds.']);


% Compute FFT
t_start=clock;
F=fftshift(fft2(g_xy));
t_FFT=etime(clock,t_start);
disp(['Computation 2D cartesian FFT took ',num2str(t_FFT),' seconds.']);

% Display total computation time
tot_comp_time=t_cart2pol+t_2D_intp+t_windowing+t_FFT;
disp(['Total computation took ',num2str(tot_comp_time),' seconds.',sprintf('\n')]);

% Construct grid for rectangle in Fourier domain
Nkx=Nx;
Nky=Ny;
kxx=(2*pi/Nkx)*[-Nkx/2:Nkx/2-1]/rx;
kyy=(2*pi/Nky)*[-Nky/2:Nky/2-1]/ry;
[kxb,kyb]=meshgrid(kxx,kyy);  % x and y of rectangle before rotation
kxr=kxb*cos(th_rot)-kyb*sin(th_rot);% x and y of rectangle after rotation
kyr=kxb*sin(th_rot)+kyb*cos(th_rot);
Fkxky=F;

% Display amplitude spectrum
%surf(kxr,kyr,abs(Fkxky));xlabel('k_x [m^{-1}]'); ylabel('k_y [m^{-1}]');zlabel('|F|');title('Amplitude spectrum');
figure;surf(kxb,kyb,abs(Fkxky));xlabel('k_x [m^{-1}]'); ylabel('k_y [m^{-1}]');zlabel('|F|');title('Amplitude spectrum');
%pause;view(2);axis('equal');pause;
%surf(kxr,kyr,10*log10(abs(Fkxky)));xlabel('k_x [m^{-1}]'); ylabel('k_y [m^{-1}]');zlabel('|F| (dB)');title('Amplitude spectrum in dB');
%pause;view(2);axis('equal');pause;close;

% Upsample in frequency domain
u=8;
g_xy_u=zeros(u*size(g_xy));
g_xy_u(1:Ny,1:Nx)=g_xy;
F2=fftshift(fft2(g_xy_u));
% Construct grid for rectangle in Fourier domain
Nkx2=u*Nx;
Nky2=u*Ny;
kxx2=(2*pi/Nkx2)*[-Nkx2/2:Nkx2/2-1]/rx;
kyy2=(2*pi/Nky2)*[-Nky2/2:Nky2/2-1]/ry;
[kxb2,kyb2]=meshgrid(kxx2,kyy2);  % x and y of rectangle before rotation
kxr2=kxb2*cos(th_rot)-kyb2*sin(th_rot);% x and y of rectangle after rotation
kyr2=kxb2*sin(th_rot)+kyb2*cos(th_rot);
Fkxky2=F2;

% Display amplitude spectrum
%surf(kxr2,kyr2,abs(Fkxky2));xlabel('k_x [m^{-1}]'); ylabel('k_y [m^{-1}]');zlabel('|F|');title('Amplitude spectrum');
%pause;view(2);axis('equal');%pause;
%surf(kxr2,kyr2,10*log10(abs(Fkxky2)));xlabel('k_x [m^{-1}]'); ylabel('k_y [m^{-1}]');zlabel('|F| (dB)');title('Amplitude spectrum in
dB');
%pause;view(2);axis('equal');pause;close;

MeasQualCM;
```

Appendix J. MeasQualCM: m-file that computes the quality measures of spectra computed by means of the conventional method

```
% File name: MeasQualCM.m
% Measure quality of conventional method

p2=100;
pd2=10; % Fraction of maximum of absolute value of F that will be used for the determination of the area of support
AF2=abs(Fkxky2);

% Check Parseval
En_gxy2=sum(sum(g_xy_u.^2));
En_Fkxky2=sum(sum(AF2.^2));
EnRatio2=En_Fkxky2/En_gxy2;
disp(['Energy in Fkxky / energy in g_xy = ',num2str(EnRatio2)]);

[iky02,ikx02]=find((kxr2<=0) | (kyr2<=0));
Fanl12=AF2;
ind12=sub2ind(size(Fanl12),iky02,ikx02);
Fanl12(ind12)=0;
%surf(kxr2,kyr2,abs(Fanl12));xlabel('k_x [m^{-1}]'); ylabel('k_y [m^{-1}]');zlabel('|F|');title('Amplitude spectrum');
%pause;view(2);axis('equal');

[max12,ky_I2]=max(Fanl12);
[max12,kx_I2]=max(max12);
ky_I2=ky_I2(kx_I2);
kxmax2=kxx2(kx_I2);
kymax2=kyy2(ky_I2);
rho_m=sqrt(kxmax2^2+kymax2^2);
phi_m=(atan2(kymax2,kxmax2)+th_rot)*180/pi;
kxmaxr2=kxmax2*cos(th_rot)-kymax2*sin(th_rot);
kymaxr2=kxmax2*sin(th_rot)+kymax2*cos(th_rot);

% Plot slice in range direction
thr=th_rot-argk1;
k1x=rho_maxorg*cos(thr)/400*[0:400];
k1y=-rho_maxorg*sin(thr)/400*[0:400];
v1=interp2(kxb2,kyb2,AF2,k1x,k1y,'linear');
rhov1=sqrt(k1x.^2+k1y.^2);
%figure;plot(rho,v1);axis([0 rho_maxorg 0 max12]);title('CM, Slice in range direction');
figure;plot(rhov1,10*log10(100/max12*v1));title('CM, Slice in range direction');
axis([0 rho_maxorg -20 20]);

% Plot slice in azimuth direction
%psi=-thr-pi/4+2*pi/Na*[0:Na/4-1];kax=modk1*cos(psi);kay=modk1*sin(psi);
%v2=interp2(kxb2,kyb2,AF2,kax,kay,'linear');
%figure;plot(modk1*(psi+th_rot),fliplr(v2));title('CM, Slice in azimuth direction');
%axis([modk1*(psi(1)+th_rot) modk1*(psi(Na/4)+th_rot) 0 max12]);
%phiv2=psi+th_rot;
%figure;plot(modk1*phiv2,10*log10(100/max12*v2));title('CM, Slice in azimuth direction');
%axis([modk1*(psi(1)+th_rot) modk1*(psi(Na/4)+th_rot) -20 20]);

psiv4=-thr-pi/2+pi/Na*[0:Na-1];kaxv4=modk1*cos(psiv4);kayv4=modk1*sin(psiv4);
v4=interp2(kxb2,kyb2,AF2,kaxv4,kayv4,'linear');
phiv4=psiv4+th_rot;
figure;plot(modk1*phiv4,10*log10(100/max12*v4));title('CM, Slice in azimuth direction');
axis([modk1*min(phiv4) modk1*(max(psiv4)+th_rot) -20 20]);


% Slice in unrotated kx-direction
%figure;plot(kxx2,10*log10(100/max12*AF2(ky_I2,:)));title('CM, Slice in unrotated kx-direction');
%axis([min(kxx2) max(kxx2) -20 20]);
% Slice in unrotated ky-direction
%figure;plot(kyy2,10*log10(100/max12*AF2(:,kx_I2)));title('CM, Slice in unrotated ky-direction');
%axis([min(kyy2) max(kyy2) -20 20]);


psiv3=-thr-0.6*pi+1.2*pi/Na*[0:Na-1];kaxv3=modk1*cos(psiv3);kayv3=modk1*sin(psiv3);
v3=interp2(kxb2,kyb2,AF2,kaxv3,kayv3,'linear');
phiv3=psiv3+th_rot;
%figure;plot(modk1*phiv3,10*log10(100/max12*v3));title('CM, Slice in azimuth direction');
%axis([modk1*(psiv3(1)+th_rot) modk1*(psiv3(Na)+th_rot) -20 20]);
```

```
if exist('v1v4.mat')
   delete v1v4.mat;
end;
save v1v4 v1 v4 max12 rhov1 phiv4;

disp(['Highest peak of abs(F) at kx = ',num2str(kxmaxr2),...
      ' 1/m (theor.: ',num2str(kx1),...
      ') and ky = ',num2str(kymaxr2), ' 1/m (theor.: ',num2str(ky1),')']);

% Determine area of support surrounding the maximum value

% Align peak at positive kx-axis:

% rho direction limits
[maxv1, rho_I1]=max(v1);
midrho=rhov1(rho_I1);
fd11=diff(v1(1:rho_I1));
minderv=max(abs(fd11));
i1=max(find( (fd11<=minderv/pd2) & (abs(v1(1:rho_I1-1))< 0.4*max12) ));
if isempty(i1)
   irhomin=1;
else
   irhomin=min(i1+1,rho_I1);
end;
lowrho=rhov1(irhomin);
rz11=midrho-lowrho;

fd12=diff(v1(rho_I1:length(rhov1)));
if isempty(fd12)
   rz12=rz11;
   irhomax=length(rhov1);
else
   maxderv=min(fd12);
   i2=min(find( (fd12>=maxderv/pd2) & (v1(rho_I1+1:length(rhov1))< 0.4*max12) ));
   if isempty(i2)
      irhomax=length(rhov1);
   else
      irhomax=max(i2+rho_I1-1,rho_I1);
   end;
   rz12=rz11;
end;
uprho=rhov1(irhomax);
rz_rng=min(rz11,rz12);


% Angle direction
[maxv2, phi_I1]=max(v3);
midphi=phiv3(phi_I1);
fd11=diff(v3(1:phi_I1));
minderv=max(abs(fd11));
i1=max(find((fd11<=minderv/pd2) & (v3(1:phi_I1-1)< 0.4*max12) ));
if isempty(i1)
   iphimin=1;
else
   iphimin=min(i1+1,phi_I1);
end;
lowphi=phiv3(iphimin);
rz11=rhov1(rho_I1)*(midphi-lowphi);

fd12=diff(v3(phi_I1:length(phiv3)));
maxderv=min(fd12);
i2=min(find( (fd12>=maxderv/pd2) & (v3(phi_I1+1:length(phiv3))< 0.4*max12) ));
if isempty(i2)
   iphimax=length(phiv3);
else
   iphimax=max(i2+phi_I1-1,phi_I1);
end;
upphi=phiv3(iphimax);
rz12=rhov1(rho_I1)*(upphi-midphi);
rz_angl=modk1*sin(min(rz11,rz12)/modk1);

% Rotate peak to positive x-axis
kxpr=kxb2*cos(thr)-kyb2*sin(thr);% x and y of rectangle after rotation
kypr=kxb2*sin(thr)+kyb2*cos(thr);
```

```
% Construct ellips
EP=((kxpr-rho_m).^2)/(rz_rng^2) + (kypr.^2)/(rz_angl^2);
[I_ky2,I_kx2]=find( (Fanl12 >= max12/p2) & (EP<1));

%G=zeros(size(Fanl12));
%indc=sub2ind(size(Fanl12),I_ky2,I_kx2);
%G(indc)=Fanl12(indc);
%figure;mesh(kxpr,kypr,G);
%view(2);axis equal;


% Computations over area of support
ind=sub2ind(size(AF2),I_ky2,I_kx2);

% Compute first order moments over area of support
rhophi_den=sum(AF2(ind)');
rhom1_num=sum(kxpr(ind).*AF2(ind));
rhophim1_num=sum(kypr(ind).*AF2(ind));
rho_est=rhom1_num/rhophi_den;
phi_est=(atan2(rhophim1_num/rhophi_den,modk1)+argk1)*180/pi ;
disp(['Estimated modulus k: ',num2str(rho_est),' [1/m] (theor.: ',num2str(modk1),')']);
disp(['Estimated azimuth: ',num2str(phi_est),' [degrees] (theor.: ',num2str(argk1*180/pi),')']);

% Compute second order moments over ellipse of support
rhom2_num=sum(((kxpr(ind)-modk1).^2).*AF2(ind));
rhophim2_num=sum((kypr(ind).^2).*AF2(ind));
rho_est_width=sqrt(rhom2_num/rhophi_den);
azim_est_width=sqrt(rhophim2_num/rhophi_den);
disp(['Effective width in modulus k direction: ',num2str(rho_est_width),' [1/m]']);
disp(['Effective width in azimuth direction: ',num2str(azim_est_width),' [1/m]']);

% Compute mixed second-order moment over ellipse of support
rhorhophim2_num=sum(abs(kxpr(ind)-modk1).*abs(kypr(ind)).*AF2(ind));
eff_width=sqrt(rhorhophim2_num/rhophi_den);
disp(['Effective width of peak (mixed second-order moment): ',num2str(eff_width),' [1/m]']);
disp([sprintf('\n')]);

% Compute S/N ratio
[iky_sz,ikx_sz]=find(kxr2<0);
[iky_gz,ikx_gz]=find(kxr2>=0);
ind_sz=sub2ind(size(AF2),iky_sz,ikx_sz);
ind_gz=sub2ind(size(AF2),iky_gz,ikx_gz);
FN=AF2;
FN(ind)=0;
FN(ind_sz)=0;
Ns=sum(sum(FN(ind_gz).^2));
S=sum(sum(AF2(ind).^2));
SN=10*log10(S/Ns);
disp(['S/N ratio = ', num2str(SN),' dB']);

% Determine amplitude of second peak
%[iky22,ikx22]=find( ((kxr2<=0) & (kyr2<=0)) | (kxr2>=0));
%Fanl22=AF2;
%ind22=sub2ind(size(Fanl22),iky22,ikx22);
%Fanl22(ind22)=0;
%max22=max(max(Fanl22));
%surf(kxr2,kyr2,abs(Fanl22));xlabel('k_x [m^{-1}]'); ylabel('k_y [m^{-1}]');zlabel('|F|');title('Amplitude spectrum');
%pause;view(2);axis('equal');
%AmpRatio2=max22/max12;
%disp(['The ratio between the amplitudes of peak 1 and peak 2 is: ',num2str(AmpRatio2)]);

ResultsCM(1)=rho_m;
ResultsCM(2)=phi_m;
ResultsCM(3)=rho_est;
ResultsCM(4)=phi_est;
ResultsCM(5)=rho_est_width;
ResultsCM(6)=azim_est_width;
ResultsCM(7)=eff_width;
ResultsCM(8)=EnRatio2;
ResultsCM(9)=SN;
```

Appendix K. MeasQualPFT: m-file that computes the quality measures of spectra computed by means of the PFT

```
% File name: MeasQualPFT.m
% Measure quality of computed DPFT

p=100; % Fraction of maximum of absolute value of F that will be used for the determination of the area of support
pd=10;     % Derivative control

% Check Parseval
En_g=sum(sum(abs(R.*(g_r_th.^2))));
En_F=sum(sum(RHO.*(abs(F).^2)));
EnRatio=En_F/En_g;
disp(['Energy in F / energy in g_r_th = ',num2str(EnRatio)]);

% Determine location of highest peak of abs(F) in first quadrant
F_Q_Quadr1=abs(F([1:Nphi/4],:));          % Quality measures of highest peak of abs(F) in first quadrant
% Define polar coordinates in Fourier domain: first quadrant for analysis
phi_q=(2*pi/Nphi)*[0:(Nphi/4-1)];      % First quadrant, instead of the entire revolution
[RHO_Q,PHI_Q]=meshgrid(rho,phi_q);   % Grid in polar Fourier domain

[max1,phi_I1]=max(F_Q_Quadr1);                  % phi_I is index where maximum of abs(F) occurs for each rho
[max1,rho_I1]=max(max1);                        % rho_I is rho index where maximum of abs(F) occurs
phi_I1=phi_I1(rho_I1);                          % phi index where maximum occurs
phi_m1=phi(phi_I1);                             % Angle (in radians) where maximum occurs
rho_m1=rho(rho_I1);                             % Modulus of wavenumber (in 1/m) where maximum occurs
DeltaRho=rho_max/Nrho;                          % Sample distance of rho

disp(['Highest peak of abs(F) at phi = ',num2str((180/pi)*phi_m1),...
     ' degrees (theor.: ',num2str(argk1*180/pi),...
     ') and rho = ',num2str(rho_m1), ' 1/m (theor.: ',num2str(modk1),')']);

kxm=rho_m1*sin(phi_m1);
kym=rho_m1*cos(phi_m1);
disp(['Measured (kx,ky): (',num2str(kxm),',',num2str(kym),...
     '), theoretical: (',num2str(kx1),',',num2str(ky1),')']);

% Determine area of support surrounding the maximum value
% Range direction
midrho=rho(rho_I1);
fd11=diff(abs(F(phi_I1,1:rho_I1)));
minderv=max(abs(fd11));
i1=max(find( (fd11<=minderv/pd) & (abs(F(phi_I1,1:rho_I1-1))< 0.4*max1) ));
if isempty(i1)
   irhomin=1;
else
   irhomin=min(i1+1,rho_I1);
end;
lowrho=rho(irhomin);
rz11=midrho-lowrho;

fd12=diff(abs(F(phi_I1,rho_I1:length(rho))));
if isempty(fd12)
   rz12=rz11;
   irhomax=length(rho);
else
   maxderv=min(fd12);
   i2=min(find( (fd12>=maxderv/pd) & (abs(F(phi_I1,rho_I1+1:length(rho)))< 0.4*max1) ));
   if isempty(i2)
      irhomax=length(rho);
   else
      irhomax=max(i2+rho_I1-1,rho_I1);
   end;
   rz12=rz11;
end;
uprho=rho(irhomax);
rz_rng=min(rz11,rz12);

% Angle direction
midphi=phi(phi_I1);
fd11=diff(abs(F(1:phi_I1,rho_I1)));
minderv=max(abs(fd11));
```

```
i1=max(find((fd11<=minderv/pd) & (abs(F(1:phi_I1-1,rho_I1))< 0.4*max1) ));
if isempty(i1)
   iphimin=1;
else
   iphimin=min(i1+1,phi_I1);
end;
lowphi=phi(iphimin);
rz11=rho(rho_I1)*(midphi-lowphi);

fd12=diff(abs(F(phi_I1:length(phi),rho_I1)));
maxderv=min(fd12);
i2=min(find( (fd12>=maxderv/pd) & (abs(F(phi_I1+1:length(phi),rho_I1))< 0.4*max1) ));
if isempty(i2)
   iphimax=length(phi);
else
   iphimax=max(i2+phi_I1-1,phi_I1);
end;
upphi=phi(iphimax);
rz12=rho(rho_I1)*(upphi-midphi);
rz_angl=modk1*sin(min(rz11,rz12)/modk1);

% Construct ellips
EP=(RHO_Q.*cos(PHI_Q)*cos(phi_m1)+RHO_Q.*sin(PHI_Q)*sin(phi_m1)-rho_m1).^2/(rz_rng^2)+...
  (-RHO_Q.*cos(PHI_Q)*sin(phi_m1)+RHO_Q.*sin(PHI_Q)*cos(phi_m1)).^2/(rz_angl^2);
[I_phi1,I_rho1]=find((F_Q_Quadr1 >= max1/p) & (EP<1));
%G=zeros(size(F_Q_Quadr1));
%indc=sub2ind(size(F_Q_Quadr1),I_phi1,I_rho1);
%G(indc)=F_Q_Quadr1(indc);
%[xc,yc]=pol2cart(PHI_Q,RHO_Q);
%figure;mesh(xc,yc,G);
%view(2);axis equal

AF=abs(F);
if exist('v1v4.mat')
   load v1v4;
end;

% Range direction
%figure;plot(rho,abs(F(phi_I1,:)));axis([0 rho_max 0 max1]);hold;title('PFT, slice in \rho direction');
%xlabel('\rho [m^{-1}]');ylabel('|F|');title('PFT, slice of |F| in range direction');
figure;plot(rho,10*log10(100/max1*AF(phi_I1,:)),'r');hold;
xlabel('\rho [m^{-1}]');ylabel('|F| in dB');%title('PFT and current method: slice of |F| in \rho direction');
axis([0 rho_max -25 20]);
plot(rhov1,10*log10((100/max12*v1)),'k--');legend('PFT','CM');
%plot([rho(min(I_rho1)), rho(min(I_rho1))], [-20 20], 'g');
%plot([rho(max(I_rho1)), rho(max(I_rho1))], [-20 20], 'g');%pause;close;

% Angular direction
azimsl=zeros(1,Na/2);
azimsl(1:Na/8)=AF(7*Na/8+1:Na,rho_I1);
azimsl(Na/8+1:Na/2)=AF(1:3*Na/8,rho_I1);
dimpl=zeros(1,Na/2);
dimpl(1:Na/8)=modk1*(phi(7*Na/8+1:Na)-2*pi);
dimpl(Na/8+1:Na/2)=modk1*phi(1:3*Na/8);
azimslfl=fliplr(azimsl);
figure;plot(dimpl,10*log10(100/max1*(azimslfl)),'r');hold;
xlabel([sprintf('%1.4f',modk1),'\cdot\phi [m^{-1}]']);ylabel('|F| in dB');
%title('PFT and current method: slice of |F| in azimuth direction');
axis([dimpl(1) max(dimpl) -25 20]);
plot(modk1*phiv4,10*log10(100/max12*v4),'k--');%legend('PFT','CM');
%plot([rho(rho_I1)*phi(min(I_phi1)), rho(rho_I1)*phi(min(I_phi1))], [-20 20], 'g');
%plot([rho(rho_I1)*phi(max(I_phi1)), rho(rho_I1)*phi(max(I_phi1))], [-20 20], 'g');%pause;close;


% Computations over area of support

ind1=sub2ind(size(F_Q_Quadr1),I_phi1,I_rho1);
% Compute estimate argument of k1
phi1m1_num=sum(rho(I_rho1).*phi_q(I_phi1).*F_Q_Quadr1(ind1)');
phi1m_den=sum(rho(I_rho1).*F_Q_Quadr1(ind1)');
phi_est=phi1m1_num/phi1m_den;
disp(['Estimated argument of k: ',num2str(phi_est*180/pi),' (degrees) (theor.: ',num2str(argk1*180/pi),')']);
% Compute first order moments over area of support
[kyaos,kxaos]=pol2cart(phi(I_phi1),rho(I_rho1));
kyaosr=kyaos*cos(argk1)+kxaos*sin(argk1);
kxaosr=-kyaos*sin(argk1)+kxaos*cos(argk1);
```

```
%figure;plot(kyaosr,kxaosr,'.');
rhophi_den=sum(F_Q_Quadr1(ind1)');
rhom1_num=sum(kyaosr.*F_Q_Quadr1(ind1)');
rho_est=rhom1_num/rhophi_den;
disp(['Estimated modulus k: ',num2str(rho_est),' [1/m] (theor.: ',num2str(modk1),')']);
% Compute second order moments over ellipse of support
rhom2_num=sum(((kyaosr-modk1).^2).*F_Q_Quadr1(ind1)');
rhophim2_num=sum((kxaosr.^2).*F_Q_Quadr1(ind1)');
rho_est_width=sqrt(rhom2_num/rhophi_den);
azim_est_width=sqrt(rhophim2_num/rhophi_den);
disp(['Effective width in modulus k direction: ',num2str(rho_est_width),' [1/m]']);
disp(['Effective width in azimuth direction: ',num2str(azim_est_width),' [1/m]']);
% Compute mixed second-order moment over ellipse of support
rhorhophim2_num=sum(abs(kyaosr-modk1).*abs(kxaosr).*F_Q_Quadr1(ind1)');
eff_width=sqrt(rhorhophim2_num/rhophi_den);
disp(['Effective width of peak (mixed second-order moment): ',num2str(eff_width),' [1/m]']);
disp([sprintf('\n')]);

% Compute S/N ratio
FN=abs(F);
FN_Q_Quadr1=F_Q_Quadr1;
FN_Q_Quadr1(ind1)=0;
FN([1:Nphi/4],:)=FN_Q_Quadr1;
FN([Nphi/2+1:Nphi],:)=0;
Ns=sum(sum(RHO([1:Nphi/2],:).*FN([1:Nphi/2],:).^2));
S=sum(sum(rho(I_rho1).*((F_Q_Quadr1(ind1)').^2)));
SN=10*log10(S/Ns);
disp(['S/N ratio = ', num2str(SN),' dB']);

% We only need to know the maximum value of the peak in the second quadrant
% Determine location of lowest peak of abs(F) in second quadrant
F_Q_Quadr2=abs(F([Nphi/4+1:Nphi/2],:));
max2=max(max(F_Q_Quadr2));
AmpRatio=max2/max1;
%disp(['The ratio between the amplitudes of peak 1 and peak 2 is: ',num2str(AmpRatio)]);

ResultsPFT(1)=rho_m1;
ResultsPFT(2)=phi_m1;
ResultsPFT(3)=rho_est;
ResultsPFT(4)=phi_est;
ResultsPFT(5)=rho_est_width;
ResultsPFT(6)=azim_est_width;
ResultsPFT(7)=eff_width;
ResultsPFT(8)=EnRatio;
ResultsPFT(9)=SN;
```