TU/e technische universiteit eindhoven

/department of electrical engineering

A PC-based real-time simulation platform for evaluating hearing aid algorithms

Job Geurts

Practical training report

Chair: Measurement and Control Systems, Signal Processing Systems (MBS-SPS)
Company: GN ReSound AS
Internal supervisor: Ir. A.J. van den Berg
External supervisor: Dr. ir. B. de Vries
Date: 30-6-2005

Abstract

A device is needed in the future so that the algorithms in hearing aids (HA) can be further improved. This device will be called here a Wireless Hearing Aid Manager (WHAM). It is a compact hand-held device with a screen, an input button and a wireless transmitter. It can be thought of as a simplified Personal Digital Assistant (PDA) or mobile phone. The screen can display a number of programmable questions or statements, such as the HA sounds too loud, the HA sounds too noisy or the speech cannot be understood. The user can answer yes or no to these questions and the answers are sent to the HA. The HA will need a receiver for registration of these answers.

As a first step in realizing the WHAM, the real-time simulation platform RTProc was obtained from Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen. The RTProc system uses a PC for real-time audio processing. It is written in C++ and consists of three modules, namely a module that operates the audio device, a module that processes the input signal by applying an algorithm and a module that takes care of the communication between the other two modules. The last module is operated by means of a number of commands in Matlab. These adjustments were made to RTProc to make it suitable for evaluating HA algorithms in real-time:

- The first adjustment is that the warped FFT framework was coded in C++. It is a dynamic link library (DLL) that can be used by the RTProc system as an algorithm component. It performs a warped fast Fourier transform (FFT) on an input audio signal. An audio processing node in a DLL called GN ReSound algorithms then calculates spectral log-gain factors by applying some specific algorithms designed by GN ReSound. Finally, the warped FFT framework applies a warped finite impulse response (FIR) filter to the input signal. This results in the output audio signal. In addition, the warped FFT framework is capable of saving input audio signal fragments of the last ten seconds.
- The second adjustment is that a graphical user interface (GUI) called *WarpedFFT* was coded in Matlab. It can operate the RTProc system that uses the warped FFT framework as algorithm component and GN ReSound algorithms as audio processing node. *WarpedFFT* can do the following during runtime. Some settings of the audio device and the warped FFT framework can be adjusted, algorithms in GN ReSound algorithms can be enabled or disabled and their parameters can be adjusted, an internal state of an algorithm in GN ReSound algorithms can be monitored and the input audio signal fragment of the last ten seconds, the parameter settings and the user feedback can be saved.

In addition, the method was documented for adding an algorithm to the warped FFT framework with GN ReSound algorithms and subsequently updating the GUI that operates the framework with the algorithms. This documentation is needed because it has to be very simple to add an algorithm and to update the GUI.

Two database utilities have also been created that can be used with the RTProc system. The first utility is called *Database* and this utility makes it possible to create instantly and audio database for evaluating an algorithm in real-time. The second utility is called *Inputplayer* and this utility makes it possible to play an input audio signal fragment that was saved by the warped FFT framework.

As example of an application of the RTProc system, the PNR algorithm was evaluated in real-time by using the warped FFT framework for RTProc, the database creator and the input fragments player. This evaluation shows that the RTProc system is very useful. Many possible applications of the RTProc system can be thought of. However, there are currently some limitations attached to the use of the RTProc system. These limitations are that the RTProc system is currently not suitable for walking around with it on a laptop, the method that is used currently for saving the input audio signal fragment of the last ten seconds is not optimal and it disturbs the real-time processing and the user feedback that is saved by *WarpedFFT* is currently limited to a quality rating (QR).

Contents

1. Introduction

A device is needed in the future so that the algorithms in hearing aids (HA) can be further improved. This device will be called here a Wireless Hearing Aid Manager (WHAM). It is a compact hand-held device with a screen, an input button and a wireless transmitter. It can be thought of as a simplified Personal Digital Assistant (PDA) or mobile phone. The screen can display a number of programmable questions or statements such as:

- The HA sounds too loud.
- The HA sounds too soft.
- The HA sounds too noisy.
- The HA is whistling.
- The speech cannot be understood.

The user can answer yes or no to these questions and the answers are sent to the HA. The HA will need a receiver for registration of these answers.

The impact and future uses of the WHAM device are immense. Firstly, it can be used as a tool for algorithm tuning and development. A number of relevant questions with respect to the specific trial can be programmed in the WHAM and the HA records the user feedback along with the signal features and the parameter settings. The HA can also make direct adaptations to the parameters based on the current parameter settings, the signal features and the user feedback. This can lead to sophisticated tuning and machine learning applications, such as tuning of the noise reduction algorithm and tuning of the automatic volume control. Subsequently, users can be sent home with the WHAM and during trials they can gather a tremendous amount of detailed patient satisfaction data that otherwise cannot be obtained. In the end, real clients can be sent home with the WHAM and it allows them to tune their hearing aids online by providing feedback on simple questions that they can understand.

As a first step in realizing the WHAM, the real-time simulation platform RTProc was obtained from Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen. RTProc uses a PC for the real-time audio processing and an audio device is used for the input and output of signals. The microphone and speaker of a HA can be connected to the audio device to serve as an input and output device. A PC is used for the real-time processing because it is easier to write an algorithm on a PC than on a digital signal processor (DSP) of a HA and a HA has less processing power than a PC. RTProc is written in C++ and consists of three modules, namely a module that operates the audio device, a module that processes the input signal by applying an algorithm and a module that takes care of the communication between the other two modules. The last module is operated by means of a number of commands in Matlab.

To make RTProc suitable for evaluating algorithms of HA in real-time, a number of adjustments have to be made to the platform. These adjustments are:

- The capability has to be added to save the input audio signal.
- A graphical user interface (GUI) has to be developed in Matlab that makes it simple to operate RTProc. It has to be possible with this GUI to select the correct audio device and to adjust its settings, to select the correct algorithm and to adjust its parameters in real-time, to monitor an internal state of the selected algorithm and to save the input audio signal, the parameter settings and the user feedback.
- HA Algorithms of GN ReSound have to be implemented in RTProc and RTProc has to be able to adjust the parameters of these algorithms in real-time.

This practical training concerns the aforementioned adjustments and has six tasks. These tasks are:

- A warped fast Fourier transform (FFT) framework with three HA algorithms of GN ReSound, namely volume control (VC), automatic gain control (AGC) and perceptual noise reduction (PNR), has to be implemented in RTProc. In addition, this warped FFT framework has to be capable of saving input audio signal fragments.

- A GUI has to be developed in Matlab to operate the warped FFT framework with GN ReSound algorithms.
- The method has to be documented for adding an algorithm to the warped FFT framework with GN ReSound algorithms and subsequently updating the GUI that operates the framework with the algorithms.
- A database creator has to be developed in Matlab, which makes it possible to create instantly an audio database for evaluating an algorithm in real-time.
- An input fragments player has to be developed in Matlab, which makes it possible to play an input audio signal fragment that was saved by the warped FFT framework.
- As example of an application of the RTProc system, the PNR algorithm has to be evaluated in real-time by using the warped FFT framework for RTProc, the database creator and the input fragments player.

First, in chapter 2 is explained how the three modules of the RTProc system work and how the RTProc system can be operated. This chapter is a summary of [3]. Subsequently, in chapter 3 is described how the warped FFT framework is implemented in the RTProc system. In addition, it is described how the capability to save input audio signal fragments is added to this framework. It is also described how the VC, AGC and PNR algorithms are implemented in the warped FFT framework and the method is given for adding an algorithm to the framework with the algorithms. Finally, it is described how the Matlab GUI is developed that operates the warped FFT framework with GN ReSound algorithms and the method is given for updating the GUI if a new algorithm is added to the framework. Parts of this chapter are taken from [1], [2], [4] and [5]. After this, in chapter 4 is described how the database creator and the input fragments player are developed in Matlab. Subsequently, in chapter 5 is the evaluation of the PNR algorithm described that was performed as example of an application of the RTProc system. It was performed in real-time by using the warped FFT framework for RTProc, the database creator and the input fragments player. Finally, some conclusions are drawn in chapter 6 about the use of the warped FFT framework with GN ReSound algorithms for RTProc for real-time audio processing and some recommendations are given for further developments on RTProc, the warped FFT framework and the GN ReSound algorithms. These developments should ultimately lead to the realization of the WHAM.

In conclusion of this introduction, I would like to thank GN ReSound for giving me the opportunity to perform this practical training. In particular, I would like to thank Bert de Vries and Almer van den Berg for their help during the training.

2. The RTProc system

This chapter is a summary of [3]. The RTProc system is a real-time simulation platform that uses a PC for real-time audio processing. To simplify the development of a new real-time audio processing algorithm, RTProc has been separated into three functional modules, namely the driver, the algorithm and the host. Because of this separation into modules, the development of a new algorithm does not include any hardware intimate programming. A schematic of the architecture of the RTProc system can be seen in figure 2.1.

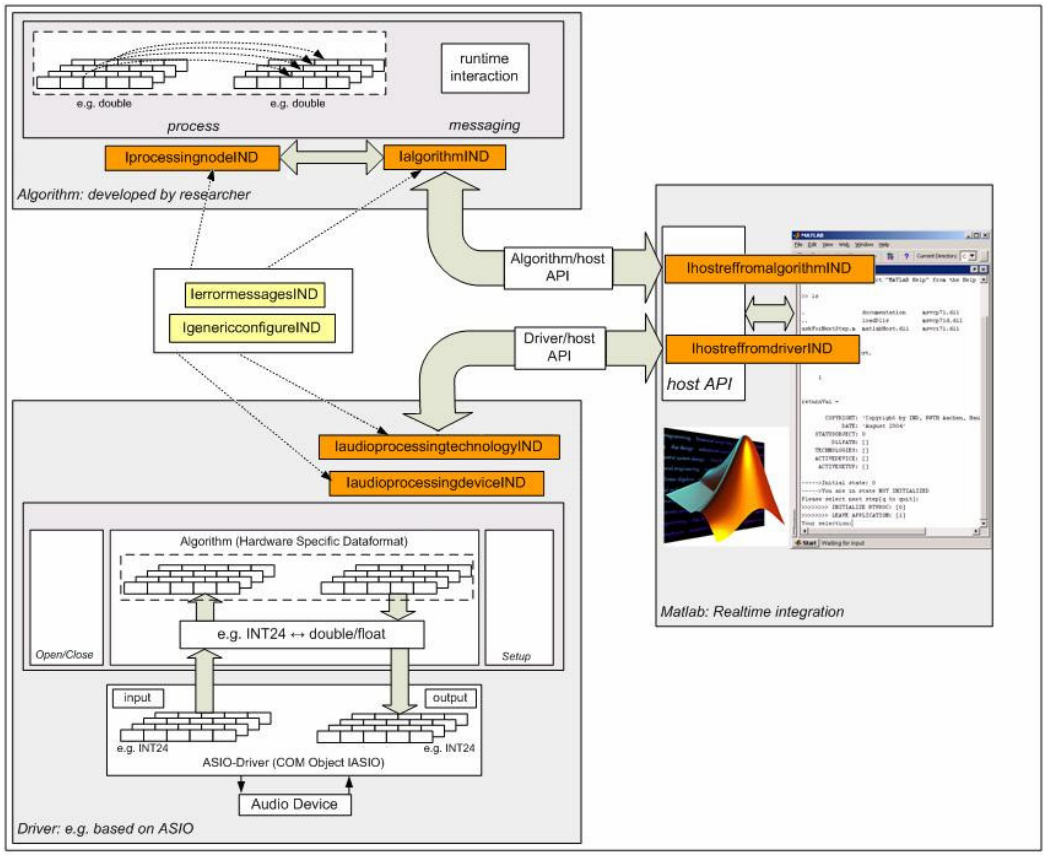

Figure 2.1: Schematic of the architecture of the RTProc system and the communication between the modules

The driver module obtains in real-time, buffer for buffer, the input audio samples from a selected audio device and then converts these samples into a hardware independent data format such as double or float. These buffers are passed to the host module and from there to the algorithm module. The algorithm modifies the input signal by applying an algorithm and the resulting output signal is passed back to the host and then to the driver. The driver converts the output signal back to the hardware specific data format and this signal is then played on the selected audio device. The host module takes care of the communication between the driver module and the algorithm module. Clearly defined Application Programmable Interfaces (API) make this communication possible. The host is integrated into Matlab and can be operated by means of a number of commands in the command window.

The main functionality of the RTProc system APIs is to hide specific complex functionality behind simple APIs. Everything can be a driver module as long as its API is followed and everything can be an algorithm module as long as its API is followed. This means that on the driver side, a technology must provide access to all available audio devices. The user can select one of the devices. After selection, the audio device must return information on its capabilities, sample rate, buffer size, input and output channels and processing format. Finally, for processing the device must be informed to start processing.

Processing itself means to pass buffers for input and for output at the same time to a callback function in the context of the device. On the algorithm side, an algorithm must communicate to the user whatever it requires for processing, namely sample rate, buffer size, number of channels for input and output and processing format. After selection, the algorithm must be informed when it has to start processing and when it is processing, it must process the buffers for input to buffers for output. The communication between the modules of the RTProc system and the interfaces for each communication can be seen in figure 2.1. The modules of the RTProc system must be derived from these interfaces, namely:

- Algorithms must be derived from *IalgorithmIND*.
- Technologies must be derived from *IaudioprocessingtechnologyIND*.
- Devices must be derived from *IaudioprocessingdeviceIND*.
- Processing nodes in the frequency domain must be derived from *IprocessingnodeIND*.
- The host must be derived from *IhostreffromalgorithmIND* and *IhostreffromdriverIND*.
- All modules must be derived from *IerrormessagesIND* and *IgenericruntimeIND*.

IerrormessagesIND contains function calls to return an error code and an error description for an unsuccessful operation. *IgenericruntimeIND* contains function calls to provide information on the internal structures of a module for the purpose of generic runtime configuration.

The RTProc system uses an enumeration type when a specific selection has to be chosen among a certain fixed amount of available opportunities. This makes it easier to understand the meaning of a specific selection. All available enumeration types in the RTProc system are defined in *audioProcessingTypedefs.h* and can be found in appendix A. Table A.1 lists the hardware independent data formats that can be used by the driver module and the algorithm module. Table A.2 lists the error codes that the modules can return in case of a failure. Table A.3 lists the states in which the modules can be. These states are important because using the RTProc system means to navigate from state to state and to call the right function in the right state. Table A.4 lists the capabilities that a device can return to indicate the possible processing scenarios. Table A.5 lists the available messages from the device to the host. Table A.6 lists the available messages from the algorithm to the host.

The RTProc system has three operation modes for processing, namely offline, talkthrough and duplex. In offline mode, the driver module is disabled and the input audio samples are obtained from the hard disk and the output audio samples are written back to the hard disk. In talkthrough mode, the algorithm module is disabled and the input audio samples from the driver module are directly sent back to the driver module to play it on the selected audio device. In duplex mode, all modules are enabled and the RTProc system obtains, processes and plays audio signals in real-time. These three operation modes avoid the implementation of different software for offline and real-time testing of an algorithm and avoid programming errors when an algorithm is switched from the offline version to the real-time version.

When a new algorithm is tested in real-time, user runtime interaction is very important to test the impact of specific parameter values. Therefore, a mechanism has been added to the RTProc system for generic configuration of algorithms during runtime. Parameters can be modified by first getting a template that is a pointer with an associated data field. This data field is set by the host and then passed to the interface. The possible data fields that can be passed through this functional gateway can be seen in figure 2.2. The entry is a pointer to the data type *rtpGenericConfigure*. There are a specific number of *rtpGenericConfigureElement* elements in that data type. Each of those has fields to indicate the type, the purpose and to hold the data. The data field is an open pointer to another element that can hold the parameter. This parameter can be modified and by setting the template, the parameter change is executed in the algorithm. The RTProc system uses various enumeration types for the generic runtime configuration mechanism. These enumeration types are important, because the open pointer in the data field is dependent of these enumeration types. All available enumeration types for the generic runtime configuration mechanism are defined in *IgenericruntimeIND.h* and can be found in appendix A. Table A.7 lists the data types that can be passed by the data field in the current *rtpGenericConfigureElement*. Table A.8 lists the possible purposes of the currently passed data field. Table A.9 lists the possible results for the current data exchange. Table A.10 lists the types of strings that can be passed.

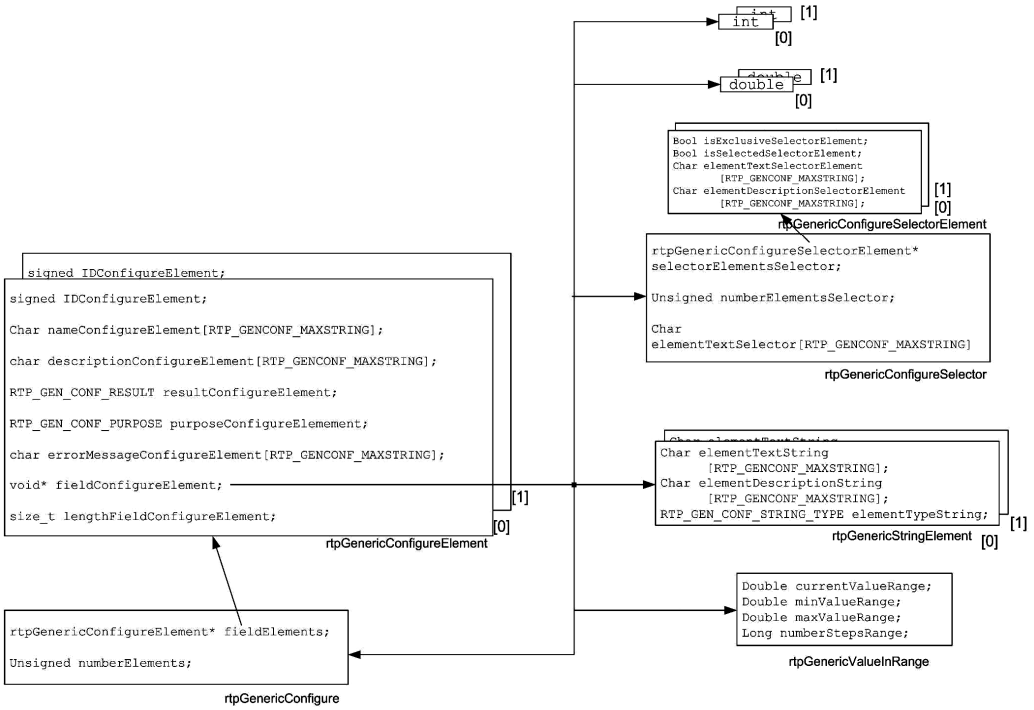

Figure 2.2: Possible data fields that can be passed through the functional gateway

The three modules of the RTProc system are discussed in more detail in the next three sections. First, the driver module is discussed, subsequently the algorithm module and finally the host module.

2.1 The driver module

The driver module consists of two components, namely a technology component and a device component. A technology is the technology principle to access an audio device. The technology component is located in a dynamic link library (DLL) and in a first state loaded at runtime. In this state, a user can access the name and description. After a technology has been selected, the next step is to activate it. After activation, the devices installed on the system will be returned. When the technology is not used anymore, it can firstly be deactivated and afterwards unselected. From here, it can be unloaded. A device component encapsulates the access to one audio device. The device component is returned from within a technology component and it is therefore not loaded from a DLL. After a device is returned by a technology, its name, description, ID and capability can be returned. In the next step a device must be selected and from here the device can be activated by calling the activate function. After this step, the driver module reports what the possible setups for processing are for this device, namely sample rate, buffer size, processing formats and number of channels for input and output. For all of these parameters, a setup value can be passed that will be used when the RTProc system is processing. When the selection of processing setup values has been completed, the current setup can be acknowledged for processing by calling preparing for processing. The setup for hardware parameters cannot be modified afterwards. If the prepare function has succeeded, the device can be driven to start processing.

There are currently two different technologies available, namely audio processing for audio streaming input output (ASIO) and audio processing for direct kernel streaming (directKS). ASIO is a technology that is based on the component object model (COM) technology from Microsoft. Steinberg Media AG Hamburg introduced this technology in order to provide a standard for high quality audio devices with low latency for professional audio applications. The latencies that can be reached with this technology are below 10 ms. During generic runtime configuration a selector can be selected to open the technology with or without a control panel specific for ASIO. In addition, another selector can be selected to pass or not pass the main window handle of the application to the ASIO control panel when loading

the ASIO technology. DirectKS is a technology that addresses the Windows driver model (WDM) directly in order to obtain access to audio devices. This bypasses the kernel mixer that is the reason for high latencies in the case of directX technology from Microsoft. DirectKS flushes buffers to the hardware that will either be filled with input audio samples or played on the selected audio device. The number of buffers that is flushed is in minimum two for a double buffering architecture. During generic runtime configuration, one parameter can be set for the number of buffers flushed for input and one parameter for output. In addition, another parameter for the default buffer size can be set.

2.2 The algorithm module

The algorithm module provides the component that contains the audio processing algorithm. The algorithm component is located in a DLL and is loaded by the host module at runtime. Several functions can be called from the host module to provide information on state, name and description of the current algorithm component. The algorithm component can be selected first and after that activated. If it is active, the setups that the algorithm can deal with are returned and the supported processing formats can be displayed. This information is intended to help the user in finding a valid setup for audio processing if this algorithm is applied. The algorithm component can now be prepared for processing by calling the function prepare processing. This function verifies whether the algorithm accepts the setup that is used for processing and it lets the algorithm specify if it is ready for processing. If the prepare function has succeeded, the start processing function can be called to start processing. To stop processing, the terminate processing function can be called and after that the post processing function. In addition, the algorithm component provides a generic runtime configuration mechanism to modify its parameters.

Most algorithm components that are used in the algorithm module have a very similar functionality except for the setups available and the main processing routine. Based on this idea, there is an implementation of a simple talkthrough component *CalgorithmIND* that copies input buffers to output buffers. All functions required by the API are implemented in a very simple manner. This base class should be taken as the starting point for implementing a new algorithm, because all functions in *CalgorithmIND* can be reused. Only the functions where a different behaviour is wanted have to be implemented again.

In addition, most new algorithm components that will be implemented have the same standard structure. The schematic of such a standard algorithm component can be seen in figure 2.3.

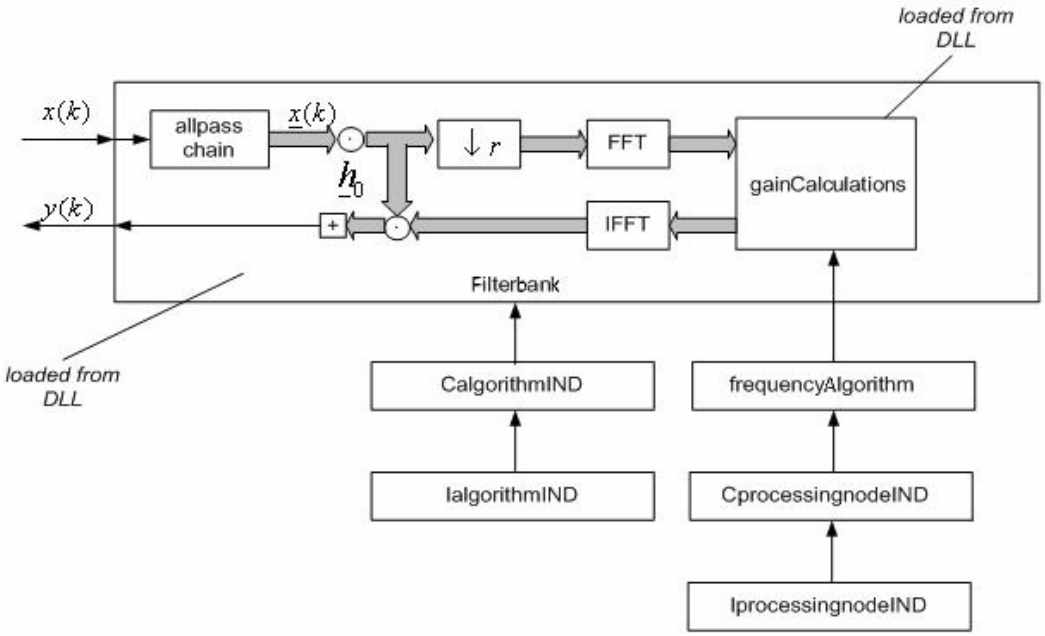

Figure 2.3: Schematic of a standard algorithm component

The standard algorithm component is derived from *CalgorithmIND* and is a kind of filter bank. It is located in a DLL and is loaded at runtime. The filter bank performs a warped FFT on the input audio signal. The resulting spectral coefficients are given to an audio processing node that calculates spectral gain factors by applying a specific algorithm. This processing node is located in a DLL and is loaded at runtime. The processing node is derived from *frequencyAlgorithm*, which is the base class for an audio processing node in the frequency domain and *frequencyAlgorithm* is derived from *CprocessingnodeIND*, which is the base class for a data processing node. The spectral gain factors are given back to the filter bank and an inverse fast Fourier transform (IFFT) is performed there to obtain gain factors in the time domain. These gain factors are used to apply a warped finite impulse response (FIR) filter to the input signal, which results in the output audio signal.

2.3 The host module

The host module is located in a DLL, but it is the main application so there is no need to dynamically load or unload it. The host controls the driver module and the algorithm module. However, the reference to the host is passed to the driver module. This reference is used during processing to pass the audio drivers from the driver to the host. It is also used during processing to report any unexpected situations from the driver to the host in order to stop processing or to display a message through a secure data display. The reference to the host is also passed to the algorithm module. This reference is used during processing to report any unexpected situations from the algorithm to the host in order to stop processing or to display a message through a secure data display.

Currently, the host module is integrated in Matlab. The host and from there the other two modules can be controlled from the Matlab command window through the MEX-interface that Matlab provides for access to native C-code. This is done by calling the MEX function:

$$[success \quad returnVal] = matlabHost(ID, \quad addParam) \tag{2.1}$$

where *ID*, *addParam*, *success* and *returnVal* are the functional ID that specifies which command is passed to the RTProc system, a parameter that depends in type and value on the functional ID, a Boolean that indicates the success of the command and a variable that depends in type and value on the functional ID and the success of the command respectively.

All MEX functions that are available to control the RTProc system can be found in appendix B, table B.1. Each function is specified with an enumeration expression, a functional ID and a short description.

The RTProc Matlab integration is based on three state machines, one for each module, with some dependencies between them. The state names in the machine for the host (system machine) start with the prefix *RTP_MX_SYSTEM*, the state names in the machine for the driver (hardware machine) with *RTP_MX_HW* and the state names in the machine for the algorithm (software machine) with *RTP_MX_SW*. In order to process audio in real-time in duplex mode, the states in the machines have to be switched until the state for processing is reached in every machine. There exists one specific function call to change from one state in a machine to another, which has to be called in the right state. The three state machines, the dependencies between them and the possible state transitions can be seen in figure 2.4. When the Matlab host is started, the system, the hardware and the software are uninitialised. The first thing that has to be done is to initialise the system. Next, the directory where the technology and algorithm components can be found has to be specified and then this directory has to be scanned for components. Now, the hardware and software are also initialised. The next thing to do is to select one of the technologies previously scanned and then to activate this technology. Now, one of the available devices has to be selected and then activated. In this state, the setup parameters of the device can be modified. Next, one of the algorithms previously scanned can be selected and then activated. Now, the system, hardware and software can be prepared for processing. If prepare for processing has succeeded, the final step is to start processing in the system, hardware and software.

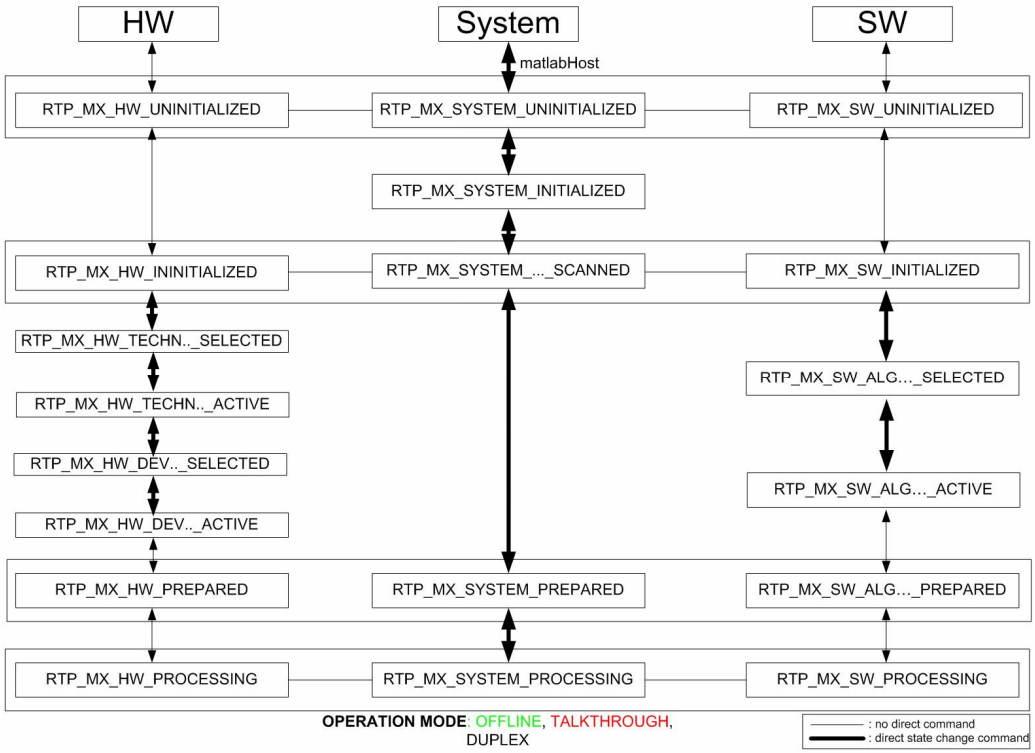

Figure 2.3: The three state machines, the dependencies between them and the possible state transitions

Just like the RTProc system, its Matlab implementation has specific enumeration expressions to simplify the understanding of selection IDs. These enumeration values can only be used in Matlab. All available enumeration types in the RTProc Matlab implementation can be found in appendix B. Table B.2 lists the available states for the system state machine, table B.3 for the hardware state machine and table B.4 for the software state machine. Table B.5 lists the available operation modes in the RTProc system. If the command called by a MEX function was unsuccessful, the *returnVal* variable contains a structure that displays the error code, the error message, the sub ID for the current error and the error level. Table B.6 lists the available error codes. There are four error levels available. Level I is for calling a wrong command at a wrong time. At level II, a component has reported an internal error. Errors at level III indicate that a component has reported an error in a function call. Level IV finally is for fatal exceptions.

Besides duplex mode, the RTProc system has two more operation modes for processing, namely offline and talkthrough. To use the offline mode, the host module has to be set into offline mode by calling the right MEX function. In this case, only an algorithm has to be selected before the system and software can be set into state processing. Now, instead of feeding in audio data from the driver module, a MEX function can be called to process an *M* by *N* input buffer, where *M* is the number of input channels and *N* the buffer size and return the output. The host has specific functions to set values for sample rate, buffer size, processing format and number of input and output channels in offline mode and the input buffer has to be in accordance with these values. To use the talkthrough mode, the host module has to be set into this mode. In this case, only a technology and device have to be selected before the system and hardware can be set into state processing. Now, the host does the processing of input buffers and the input buffers are simply copied to the output buffers.

When generic runtime configuration is used by calling the right MEX function, the internal data type of the RTProc generic runtime configuration is translated to a MEX format for the Matlab command window. If modifications have to be applied to the generic data, the first thing to do is to get this data. Then, specific modifications can be applied to it and after that, it has to be sent back to the RTProc system.

There are some tools available that help the use of the RTProc Matlab integration in order to understand its principles. Several tools transform the IDs of RTProc commands or enumeration expressions into names or descriptions or the other way around. There is also a navigation script available called *navigateRTProc.m* that helps the usage of the RTProc Matlab integration in a command line based way. This script can be called in any state of the RTProc Matlab integration. It scans the current state and presents a dialog that proposes the possible actions that can be undertaken in that state. An action can be entered and the script will call the right RTProc command. The script can be left at any state in order to prepare some data on the workspace and entered later again to continue at the same state at which the script was left before.

3. The warped FFT framework for RTProc

The warped FFT framework is an algorithm component that performs a warped FFT on an input audio signal, resulting in spectral coefficients. Spectral log-power coefficients are then calculated from these spectral coefficients and these spectral log-power coefficients are given to an audio processing node called GN ReSound algorithms. This audio processing node calculates spectral log-gain factors by applying some specific algorithms designed by GN ReSound. Subsequently, spectral linear-gain factors are calculated from the spectral log-gain factors in the warped FFT framework and an IFFT is performed to transform these spectral linear-gain factors into gain factors in the time domain. Finally, the warped FFT framework applies a warped FIR filter to the input signal, using the gain factors in the time domain. This results in the output audio signal. In addition, the warped FFT framework is capable of saving input audio signal fragments of the last ten seconds. This chapter discusses first some theory behind warped FFTs and warped FIR filters. Next, it describes how the warped FFT framework is coded in C++ to produce a DLL that can be used by the RTProc system as an algorithm component. After this, it discusses some theory behind the algorithms that are used to calculate the spectral log-gain factors. Subsequently, it describes how GN ReSound algorithms is coded in C++ to produce a DLL that can be used by the RTProc system as an audio processing node. In addition, the method is given for adding an algorithm that calculates spectral log-gain factors to GN ReSound algorithms. Finally, it describes how a GUI is coded in Matlab that can operate the RTProc system that uses the warped FFT framework as algorithm component and GN ReSound algorithms as audio processing node. In addition, the method is given for updating the GUI if a new algorithm is added to GN ReSound algorithms.

3.1 The warped FFT framework

This section discusses first some theory behind warped FFTs and warped FIR filters. Next, it describes how the warped FFT framework is coded in C++ to produce a DLL that can be used by the RTProc system as an algorithm component.

3.1.1 Theory

This section is borrowed from [2]. Frequency warping modifies the uniform frequency spacing common to most digital signal processing procedures to give a non-uniform frequency spacing. If the frequency warping is designed in the correct manner, it gives a system that has a frequency representation very close to the auditory Bark scale. Frequency warping is achieved by replacing the unit delays in a digital filter with first-order all-pass filters given by:

$$A(z) = \frac{z^{-1} - a}{1 - az^{-1}} \tag{3.1}$$

where *a* is the warping parameter.

For *a* real, setting a > 0 increases the frequency resolution at low frequencies and setting a < 0 increases the resolution at high frequencies. The optimal value for *a* to give a closest fit to the Bark frequency scale is for a sampling frequency of 16 kHz equal to 0.5756.

An FFT in the unwarped frequency domain produces values at uniformly spaced frequencies ω_k along the frequency axis $0 \le \omega \le 2\pi$. An FFT in the warped frequency domain has samples at the frequencies $\Omega(\omega_k)$ given by the transformation:

$$\Omega(\omega) = \tan^{-1}\left[\frac{(1-a^2)\sin(\omega)}{(1+a^2)\cos(\omega) + 2a}\right] \tag{3.2}$$

The properties of the warped FFT are similar to those of the unwarped FFT. For example, windowing the contents of the all-pass filtered data buffer is equivalent to smoothing the FFT

in the warped frequency domain and reduces the warped FFT side lobe level. For example, using a Hanning window is equivalent to a three-point frequency domain smoothing function in which each FFT bin is combined with its neighbour to either side. This property still holds in the warped frequency domain, but because the FFT bin frequency spacing has been warped, the frequency extent of the smoothing has also been warped by an equivalent amount. Thus, the frequency smoothing for the warped FFT is by a constant amount on the Bark frequency scale.

The transfer function of a warped FIR filter is the weighted sum of the outputs of each all-pass section and is given by:

$$\frac{Y(z)}{X(z)} = \sum_{k=0}^{K} b_k A^k(z) \tag{3.3}$$

where b_k and K are the FIR filter coefficients and the number of all-pass sections respectively.

The behaviour of a warped FIR filter is similar to that of its unwarped counterpart. In particular, forcing the filter coefficients b_k to have even symmetry for an unwarped FIR filter yields a linear-phase filter, in which the filter delay is independent of the actual coefficients chosen as long as the symmetry is preserved. Similarly, forcing even symmetry for the coefficients of a warped FIR filter gives a filter having a fixed frequency-dependent group delay that is independent of the actual filter coefficient values.

A combined warped FFT and warped FIR filter system can be seen in figure 3.1. The figure shows a sample-by-sample processing implementation, but block processing that works at brick rate is also possible. The cascade of first-order all-pass filters is called a tapped delay line and it is used for both the FFT and the FIR filter.

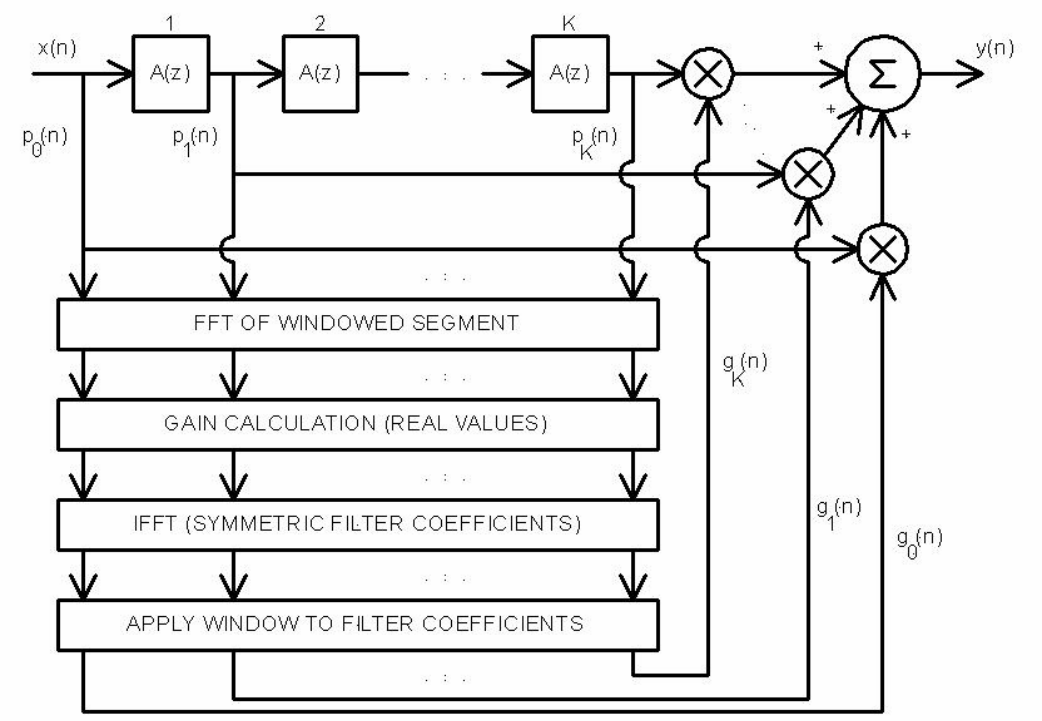

Figure 3.1: Combined warped FFT and warped FIR filter system

The input signal *x(n)* is first passed through the tapped delay line and the output of the k[th] all-pass section is given by:

$$\begin{aligned} p_0(n) &= x(n) \\ p_k(n) &= a\left[p_k(n-1) - p_{k-1}(n)\right] + p_{k-1}(n-1), \quad 1 \le k \le K \end{aligned} \tag{3.4}$$

Next, the sequence of delayed samples $p_k(n)$ is windowed and an FFT is then calculated using the windowed sequence. The result of the FFT is a spectrum sampled at a constant spacing on a Bark frequency scale. Because the data sequence is windowed, the spectrum is smoothed in the warped frequency domain, giving overlapping frequency bands. The log-power spectrum is computed from the warped FFT and the spectral log-gain factors are then computed from the warped power spectrum by a specific algorithm. The spectral linear-gain factors that are calculated from the spectral log-gain factors are pure real numbers, so the IFFT to give the warped time-domain filter results in a set of FIR filter coefficients that is real and has even symmetry. The system output is then calculated by convolving the delayed samples with the FIR filter coefficients:

$$y(n) = \sum_{k=0}^{K} g_k(n) p_k(n) \tag{3.5}$$

where $g_k(n)$ are the FIR filter coefficients.

In comparison with a conventional combined FFT and FIR filter system having the same FIR filter length, the combined warped FFT and warped FIR filter system will require more computational resources because of the all-pass filters in the tapped delay line. However, in many cases the warped FIR filter will be shorter than the conventional FIR filter that is needed to achieve the same degree of frequency resolution.

3.1.2 Implementation

The C++ project in which the warped FFT framework is implemented is called *WarpedFFT*. When this project is built, a DLL is produced that is called *WarpedFFT.dll*. This DLL can be used by the RTProc system as an algorithm component. The main source file of *WarpedFFT* is called *cgnref.cpp*. It is now described how *WarpedFFT* works. The parts of *cgnref.cpp* that are important for understanding this can be seen in appendix C.

Includes:

Cgnref.cpp uses three tables with a constant length that have to be loaded. These tables are a Hanning window, which is loaded from *hann31.tab*, a synthesis window, which is loaded from *synthwin32.tab* and a discrete Fourier transform matrix, which is loaded from *dftmtx32.tab*.

Constructor:

The *constructor* of *cgnref.cpp* sets the default setup for processing and allocates a generic runtime configuration template. The default setup is set by loading the default values for the parameters that are used by the combined warped FFT and warped FIR filter system. These parameters are *samplerate*, which sets the sample rate of the system, *numberBands*, which sets the number of frequency bands that is used by an audio processing node, *alpha*, which sets the warping parameter of the first-order all-pass filters, *blockSize*, which sets the brick rate of the block processing in the system and *algorithmDLL*, which sets the name of the DLL that is used as audio processing node. For generic runtime configuration, first a template with two elements is allocated. Then, the first element is pointed to a new template with five elements, one element for each parameter that is used by the system and these elements are filled. Next, the second element is pointed to a new template with three elements. The first element of this new template is used later as a recursive element for an audio processing node. The second element is used to select the option to save the input audio signal fragment of the last ten seconds. This element is first pointed to a new template with one element. Then, the new element is pointed to a new template with one selector that selects the save option and this selector is filled. The third element is used to specify the filename where the input audio signal fragment is written to. This element is also filled.

activateAlgorithm:

The function *activateAlgorithm* loads an audio processing node and initialises it. It then fills the recursive element of the generic runtime configuration template and

points it to the template from the audio processing node. After that, it activates *WarpedFFT* and sets its processing setup.

prepareProcessingAlgorithm:
The function *prepareProcessingAlgorithm* checks whether the processing setup of the algorithm is in accordance with the processing setup of the device.

startProcessingAlgorithm:
The function *startProcessingAlgorithm* is the last step before processing. It first checks whether the processing setup is in accordance with the processing setup of the audio processing node. Then, it allocates the memory that is required for processing. Finally, it initialises the all-pass chain, the plan for the FFT, the synthesis window, the Hanning window, the synthesis matrix required for the IFFT and the circular input buffer that stores the last ten seconds of the input audio signal. The tables that are used by the synthesis window, the Hanning window and the synthesis matrix are of a constant length. Therefore, only 17 can be used as the number of frequency bands in an audio processing node and as a result, only 32 can be used as the number of frequency bands in the system. This means that if the *numberBands* parameter is changed, *WarpedFFT* will not work anymore.

stopProcessing:
The function *stopProcessing* releases the memory that was previously allocated for processing and stops processing in the audio processing node.

process:
See the internal GN ReSound version of this report for a detailed description of this function.

deactivateAlgorithm:
The function *deactivateAlgorithm* deactivates *WarpedFFT* and unloads the audio processing node.

getGenericConfigureTemplate:
The function *getGenericConfigureTemplate* returns the template for generic runtime configuration. In state selected, the first element of the template is returned, with which the five parameters of the combined warped FFT and warped FIR filter system can be modified. In state processing, the second element of the template is returned, with which the template from an audio processing node can be reached, the option to save the input audio signal fragment of the last ten seconds can be selected and the filename where the input audio signal fragment is written to can be specified.

exchangeDataConfigure:
The function *exchangeDataConfigure* exchanges data with the template for generic runtime configuration. In state selected, data is exchanged with the first element of the template, with which the five parameters of the combined warped FFT and warped FIR filter system can be modified. In state processing, data is exchanged with the second element of the template, with which data can be exchanged with the template from an audio processing node, the option to save the input audio signal fragment of the last ten seconds can be selected and the filename where the input audio signal fragment is written to can be specified.

3.2 The GN ReSound algorithms

This section discusses first some theory behind the algorithms that are used to calculate the spectral log-gain factors. Subsequently, it describes how GN ReSound algorithms is coded in C++ to produce a DLL that can be used by the RTProc system as an audio processing node. In addition, the method is given for adding an algorithm that calculates spectral log-gain factors to GN ReSound algorithms.

3.2.1 Theory

This section summarizes relevant material from [1], [4] and [5]. There are currently three algorithms available in GN ReSound algorithms to calculate spectral log-gain factors. These algorithms are VC, AGC and PNR. VC sets its spectral log-gain factors to a desired value, independent of the input spectral log-power coefficients. It is a very simple algorithm and therefore its theory is not described in this section.

The AGC algorithm attempts to restore the normal loudness perception of the input audio signal. A block diagram of AGC can be seen in figure 3.2.

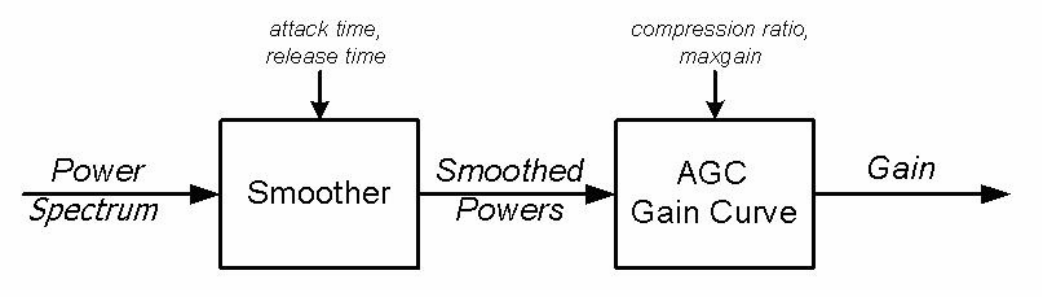

Figure 3.2: Block diagram of AGC

See the internal GN ReSound version of this report for the technical details of AGC.

The PNR algorithm attempts to suppress frequency bands with a low estimated signal-to-noise ratio (SNR). A block diagram of PNR can be seen in figure 3.4.

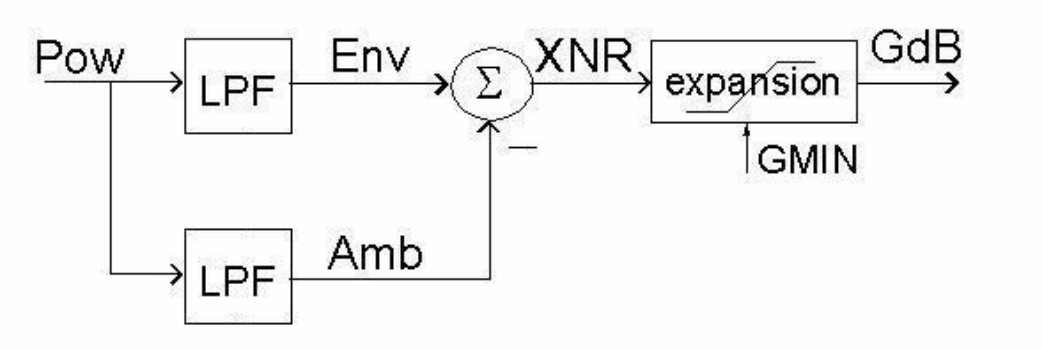

Figure 3.4: Block diagram of PNR

See the internal GN ReSound version of this report for the technical details of PNR.

3.2.2 Implementation

The C++ project in which GN ReSound algorithms is implemented is called *ReSoundAlgorithms*. When this project is built, a DLL is produced that is called *ReSoundAlgorithms.dll*. This DLL can be used by the RTProc system as an audio processing node. The main source files of *ReSoundAlgorithms* are called *ReSoundAlgorithms.cpp*, *Volume.cpp*, *Agc.cpp* and *Pnr.cpp*. It is now described how *ReSoundAlgorithms* works. The parts of *ReSoundAlgorithms.cpp*, *Volume.cpp*, *Agc.cpp* and *Pnr.cpp* that are important for understanding this can be seen in appendix D.

Constructor:

The *constructor* of *ReSoundAlgorithms.cpp* allocates a generic runtime configuration template. First, a template with four elements is allocated. The first element is needed for the algorithm selectors that select which algorithms that calculate spectral log-gain factors, namely VC, AGC and PNR, are active. The other three elements are needed as recursive elements for the three algorithms. Next, the first element is pointed to a

new template with one element, this element is pointed to a new template with three selectors, one selector for each algorithm and these selectors are filled.

initialize:

The function *initialize* initialises *ReSoundAlgorithms* and fills the recursive elements of the template for generic runtime configuration. The second element is pointed to the template from VC, the third element to the template from AGC and the fourth element to the template from PNR.

useSetupStartProcessing:

The function *useSetupStartProcessing* allocates the memory that is required for processing. Then, it initialises the three algorithms. Finally, it checks whether the processing setup is in accordance with the processing setup of the audio processing node.

stopProcessing:

The function *stopProcessing* first releases the memory that was previously allocated for processing. Then, it uninitialises the three algorithms.

algorithm:

See the internal GN ReSound version of this report for a detailed description of this function.

getGenericConfigureTemplate:

The function *getGenericConfigureTemplate* returns the template for generic runtime configuration. The algorithm selectors that select which algorithms that calculate spectral log-gain factors are active can be modified with the first element of the template. The template from VC can be reached with the second element of the template, the template from AGC with the third element of the template and the template from PNR with the fourth element of the template.

exchangeDataConfigure:

The function *exchangeDataConfigure* exchanges data with the template for generic runtime configuration. The algorithm selectors that select which algorithms that calculate spectral log-gain factors are active can be modified with the first element of the template. With the second element of the template can data be exchanged with the template from VC, with the third element of the template can data be exchanged with the template from AGC and with the fourth element of the template can data be exchanged with the template from PNR.

It is now described how the three algorithms that calculate spectral log-gain factors, namely VC, AGC and PNR, are implemented. VC is a very simple algorithm and therefore it is described very shortly. AGC and PNR are implemented in a similar way, therefore AGC is described completely and in addition the points of PNR that differ from AGC.

Volume:

The *constructor* of *Volume.cpp* loads the default value for the *volume* parameter and allocates a generic runtime configuration template with one element for the *volume* parameter. The function *algorithm* is the main processing routine and sets the output spectral log-gain factors equal to the *volume* parameter. The function *getGenericConfigureTemplate* returns the template for generic runtime configuration. The function *exchangeDataConfigure* exchanges data with the template for generic runtime configuration. The *volume* parameter can be modified with this template.

Constructor:

The *constructor* of *Agc.cpp* is empty.

init:

The function *init* allocates the memory that is required for processing. Then, it loads the default values of the parameters of the AGC algorithm and initialises the state of

the low pass filter in AGC. A parameter of the AGC algorithm can have a different value for each frequency band and therefore it is an array with size *numberBands*. The parameters are *ffup*, the forgetting factor for state up, *ffdn*, the forgetting factor for state down, *slp*, the slope of the gain curve, *ofst*, the offset of the gain curve and *gmax*, the maximum gain. Finally, *init* allocates a generic runtime configuration template. First, a template with five elements is allocated, one element for each parameter of AGC. Then, these elements are filled.

uninit:

The function *uninit* releases the memory that was previously allocated.

algorithm:

See the internal GN ReSound version of this report for a detailed description of this function.

getGenericConfigureTemplate:

The function *getGenericConfigureTemplate* returns the template for generic runtime configuration. The parameters of the AGC algorithm can be modified with the five elements of this template.

exchangeDataConfigure

The function *exchangeDataConfigure* exchanges data with the template for generic runtime configuration. The parameters of the AGC algorithm can be modified with the five elements of this template.

Pnr:

See the internal GN ReSound version of this report for a detailed description of the implementation of PNR.

The method for adding an algorithm that calculates spectral log-gain factors to *ReSoundAlgorithms* is described step by step in appendix E.

3.3 The Matlab GUI for the warped FFT framework

A GUI that is called *WarpedFFT* is developed, which makes it simple to operate the RTProc system that uses the warped FFT framework as algorithm component and GN ReSound algorithms as audio processing node. *WarpedFFT* is developed in Matlab with GUIDE, which is a GUI development environment that provides a set of tools that greatly simplify the process of creating a GUI. *WarpedFFT* selects the correct audio device and sets its settings and then selects the warped FFT framework as algorithm component and GN ReSound algorithms as audio processing node. Moreover, it is possible with *WarpedFFT* to do the following during runtime. Some settings of the audio device and the warped FFT framework can be adjusted, algorithms in GN ReSound algorithms can be enabled or disabled and their parameters can be adjusted, an internal state of an algorithm in GN ReSound algorithms can be monitored and the input audio signal fragment of the last ten seconds, the parameter settings and the user feedback can be saved. The user feedback is a quality rating (QR) of the output audio signal. It can have five categorical values, namely 1 for bad, 2 for poor, 3 for fair, 4 for good and 5 for excellent. The layout of *WarpedFFT* can be seen in figure 3.5 and the layout file is called *WarpedFFT.fig*. *WarpedFFT* uses *WarpedFFT.m* as the source file to control the layout file. The basic functions of *WarpedFFT*, such as opening function and output function, can be found in *WarpedFFT.m*. In addition, the functions belonging to the components of *WarpedFFT*, such as text boxes and buttons, can also be found in *WarpedFFT.m*. These functions are executed when their corresponding components are used. The other source files that *WarpedFFT* uses are called *initasio4all.m*, *uninit.m*, *startprocess.m*, *stopprocess.m*, *setblocksize.m*, *setbuffersize.m*, *setalpha.m*, *disablevolume.m*, *setvolume.m*, *disableagc.m*, *setagc.m*, *disablepnr.m*, *setpnr.m*, *interpolate.m*, *plotvad.m* and *saveinput.m*. It is now described how *WarpedFFT* works. VC is not treated in this description, AGC is fully treated and only the points of PNR that differ from AGC are treated, because *WarpedFFT* operates these three algorithms in the same way. The

parts of the source files that are important for understanding how *WarpedFFT* works can be seen in appendix F.

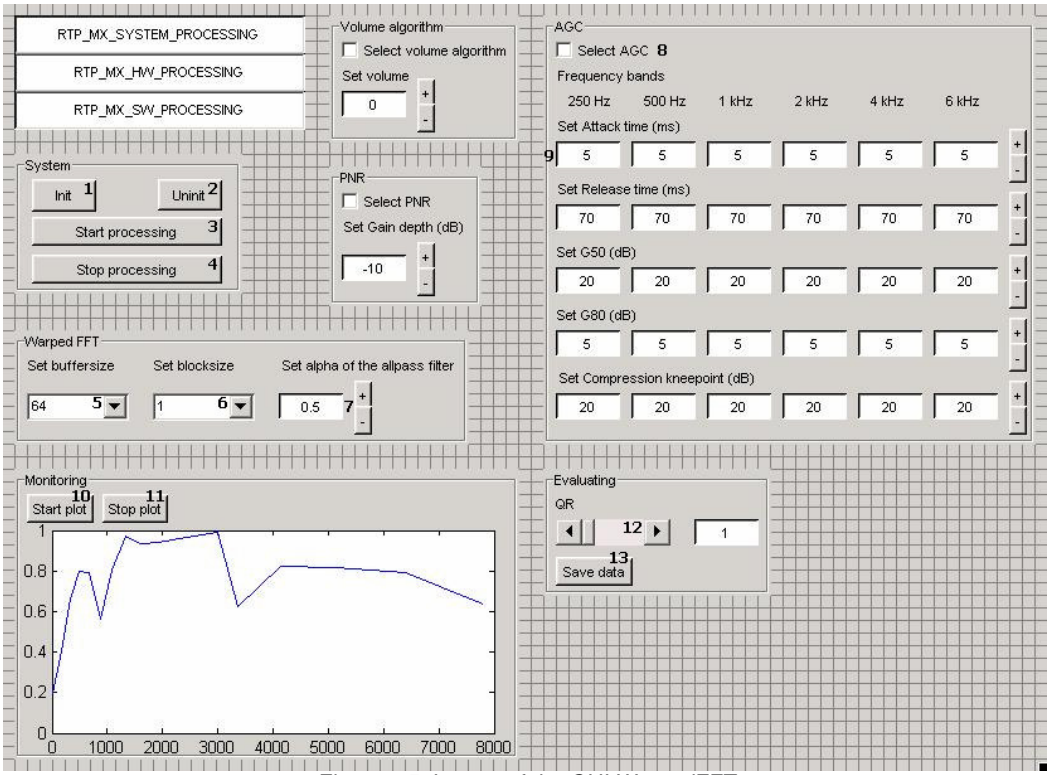

Figure 3.5: Layout of the GUI WarpedFFT

WarpedFFT_OpeningFcn:

The function *WarpedFFT_OpeningFcn* is executed just before *WarpedFFT* is made visible. This function initialises the variables, initialises the text boxes that display the states of the system, the hardware and the software by calling *update_textbox* and initialises the states of the components.

update_textbox:

The function *update_textbox* updates the text boxes that display the state of the system, the hardware and the software. It is called by a function in *WarpedFFT* when the states of the system, the hardware and the software have changed and the text boxes have to be updated.

1 *pushbutton1_Callback:*

The function *pushbutton1_Callback* is the initialise function. First, it initialises the RTProc system by calling *initasio4all*. This function loads the DLLs that can be used by the RTProc system, activates the technology ASIO, activates the audio device asio4all, sets the sample rate of this device, selects the input and output channels that are used by this device and activates the algorithm *warpedFFT* that uses *ReSoundAlgorithms* as audio processing node. Next, it sets the brick rate of the block processing in *WarpedFFT*, called *blocksize*, by calling *setblocksize*. This function deactivates the algorithm, gets the current *blocksize*, modifies the *blocksize*, sets the new *blocksize* and activates the algorithm. After this, it sets the *buffersize* of the audio device by calling *setbuffersize*. Subsequently, it sets the warping parameter of the first-order all-pass filters in *WarpedFFT*, called *alpha*, by calling *setalpha*. This function works in the same way as *setblocksize*. Finally, it calls *update_textbox* and updates the states of the components.

2 *pushbutton2_Callback:*

The function *pushbutton2_Callback* is the uninitialise function. First, it uninitialises the RTProc system by calling *uninit*. This function unselects the used algorithm, unselects the used device, unselects the used technology, unloads the DLLs and terminates the RTProc system. Next, it calls *update_textbox* and updates the states of the components.

3 *pushbutton3_Callback:*

The function *pushbutton3_Callback* is the start processing function. First, it starts processing in the RTProc system by calling *startprocess*. Then, it calls *update_textbox* and updates the states of the components. After this, it checks whether AGC is currently enabled or disabled and updates the state of AGC with this setting by calling *disableagc*. This function gets the current state of AGC, modifies the state and sets the new state. Next, it updates the parameters of AGC with the current settings by calling *setagc*, if AGC is enabled. The parameters of AGC that can be set in the GUI are the *attack time*, the *release time*, *G50*, *G80* and the *compression knee point*. These GUI parameters can be set for six frequency bands. *Setagc* interpolates the GUI parameters from six to seventeen frequency bands, because AGC uses seventeen frequency bands. Next, it calculates the AGC parameters from the GUI parameters. The *attack time* gives the forgetting factor for state up, the *release time* gives the forgetting factor for state down, *G50* and *G80* give the slope and the offset of the gain curve and the *compression knee point* gives the maximum gain. Finally, it gets the current parameters of AGC, modifies the parameters and sets the new parameters. After that, *pushbutton3_Callback* resets the QR to one.

4 *pushbutton4_Callback:*

The function *pushbutton4_Callback* is the stop processing function. First, it stops and deletes a timer that is needed for monitoring an internal state of an algorithm in GN ReSound algorithms. Next, it stops processing in the RTProc system by calling *stopprocess*. Then, it calls *update_textbox* and updates the states of the components.

5 *popupmenu2_Callback:*

The function *popupmenu2_Callback* modifies the *buffersize*. First, it gets the selected item from popupmenu2 and then, it uses this item to modify the *buffersize* by calling *setbuffersize*. This function also changes the contents of popupmenu3, because the *buffersize* has to be a multiple of the *blocksize*. Finally, it selects a new item in popupmenu3 and it uses this item to modify the *blocksize*.

6 *popupmenu3_Callback:*

The function *popupmenu3_Callback* modifies the *blocksize*. It gets the selected item from popupmenu3 and it uses this item to modify the *blocksize* by calling *setblocksize*.

7 *pushbutton25_Callback, pushbutton26_Callback and edit48_Callback:*

The functions *pushbutton25_Callback*, *pushbutton26_Callback* and *edit48_Callback* modify *alpha*. *Pushbutton25_Callback* adds 0.05 to *alpha*, *pushbutton26_Callback* subtracts 0.05 from *alpha* and *edit48_Callback* gets the contents from edit48 and uses it as *alpha*. All three functions call *setalpha* to modify *alpha*.

8 *checkbox2_Callback:*

The function *checkbox2_Callback* enables or disables AGC. First, it checks checkbox2 whether AGC is currently enabled or disabled. Then, it updates the state of AGC with this setting by calling *disableagc*. After this, it updates the parameters of AGC with the current settings by calling *setagc*, if AGC is enabled. Finally, it updates the states of the components.

It is now described how the parameters of AGC can be modified. Only the *attack time* is treated in this description, because WarpedFFT modifies the five parameters of AGC in the same way.

9 *pushbutton7_Callback, pushbutton8_Callback, edit5_Callback, edit6_Callback, edit7_Callback, edit8_Callback, edit9_Callback and edit10_Callback:*

The functions *pushbutton7_Callback*, *pushbutton8_Callback*, *edit5_Callback*, *edit6_Callback*, *edit7_Callback*, *edit8_Callback*, *edit9_Callback* and *edit10_Callback* modify the *attack time*. *Pushbutton7_Callback* adds one to the *attack time* for all six frequency bands, *pushbutton8_Callback* subtracts one from the *attack time* for all six frequency bands, *edit5_Callback* gets the contents from edit5 and uses it as the *attack time* for the first frequency band, *edit6_Callback* gets the contents from edit6 and uses it as the *attack time* for the second frequency band, *edit7_Callback* gets the contents from edit7 and uses it as the *attack time* for the third frequency band, *edit8_Callback* gets the contents from edit8 and uses it as the *attack time* for the fourth frequency band, *edit9_Callback* gets the contents from edit9 and uses it as the *attack time* for the fifth frequency band and *edit10_Callback* gets the contents from edit10 and uses it as the *attack time* for the sixth frequency band. All eight functions call *setagc* to modify the *attack time*.

PNR has only one parameter that can be set in the GUI. This GUI parameter is the *gain depth* and it can be set for one frequency band. In addition, PNR has an internal state that can be monitored by *WarpedFFT*. This state is the state of the VAD low pass filter, the *vadState*.

setpnr:

Setpnr updates the parameters of PNR with the current settings. It interpolates the *gain depth*, the GUI parameter, from one to seventeen frequency bands and then calculates the user parameter in the gain function, the PNR parameter, from the *gain depth*.

10 *pushbutton27_Callback:*

The function *pushbutton27_Callback* starts the monitoring of the *vadState*. First, it initialises a timer that calls every 0.1 second a function that plots the current *vadState*. This function is called *plotvad* and it gets the current *vadState* first. Then, it plots the *vadState* in *WarpedFFT*. Finally, it pauses 0.5 second to enable *WarpedFFT* to update. Next, *pushbutton27_Callback* updates the states of the components. Finally, it starts the timer that was initialised before.

11 *pushbutton28_Callback:*

The function *pushbutton28_Callback* stops the monitoring of the *vadState*. First, it stops and deletes the timer that is needed for the monitoring. Then, it updates the states of the components.

12 *slider2_Callback:*

The function *Slider2_Callback* gets the QR from the user.

13 *pushbutton29_Callback:*

The function *pushbutton29_Callback* saves the input audio signal fragment of the last ten seconds, the parameter settings and the user feedback. It saves several items to a text file. These items are the name of an audio file, the values of the *blocksize*, *buffersize* and *alpha*, the values of the parameters of the algorithms VC, AGC and PNR that are enabled and the QR. In addition, it saves the input audio signal fragment of the last ten seconds to the aforementioned audio file by calling *saveinput*. This function gets the template for generic runtime configuration from the warped FFT framework, selects the option to save the input audio signal fragment of the last ten seconds, specifies the filename where the input audio signal fragment is written to and sets the new template for generic runtime configuration. Then, *pushbutton29_Callback* stops and deletes the timer that is needed for the monitoring, disables all components, stops processing and pauses for 2.5 seconds, because saving the input audio signal fragment of the last ten seconds is a very heavy operation and it disturbs the real-time processing. Finally, *pushbutton29_Callback* starts processing again.

The method for updating the GUI if a new algorithm that calculates spectral log-gain factors is added to *ReSoundAlgorithms* is described step by step in appendix G.

4. Database utilities

In this chapter are two database utilities described. The first utility is called *Database* and this utility makes it possible to create instantly and audio database for evaluating an algorithm in real-time. The second utility is called *Inputplayer* and this utility makes it possible to play an input audio signal fragment that was saved by the warped FFT framework.

4.1 The database creator

A program called *Database* is developed that makes it possible to create instantly an audio database for evaluating an algorithm in real-time. *Database* has a GUI that makes it simple to operate it and it is developed in Matlab with GUIDE. It has two operation modes, namely write and read. In writing mode, the input signal filename, the input noise filename, the SNR of the output signal and the root mean square value (RMS) of the output signal can be specified. *Database* mixes the input signal and the input noise with the specified SNR and RMS to get the output signal. It loops through the input signal and the input noise with a buffer of ten seconds and changes in the four parameters will be incorporated by *Database* once every ten seconds. *Database* starts to read the input signal or input noise from the beginning, when a new filename is specified for it. In reading mode, an input database filename can be specified. *Database* reads from this file the actions that were performed in a specific write session and performs these actions to get the output signal. The actions that can be read from the file are the time when it has to start playing the output signal, the time when it has to stop playing the output signal and the time when it has to change the values of the four parameters and the new values of the parameters. The layout of *Database* can be seen in figure 4.1 and the layout file is called *Database.fig*. *Database* uses *Database.m* as the source file to control the layout file. The basic functions of *Database*, such as opening function and output function, can be found in *Database.m*. In addition, the functions belonging to the components of *Database*, such as text boxes and buttons, can also be found in *Database.m*. These functions are executed when their corresponding components are used. The other source file that *Database* uses is called *AudioRead.m*. It is now described how *Database* works. The parts of the source files that are important for understanding this can be seen in appendix H.

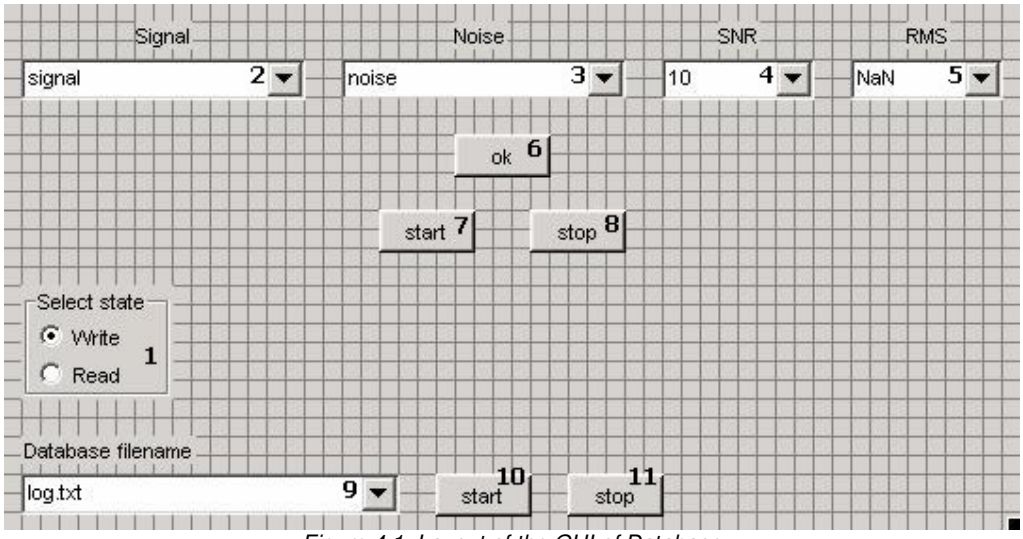

Figure 4.1: Layout of the GUI of Database

Database_OpeningFcn:

The function *Database_OpeningFcn* is executed just before *Database* is made visible. This function initialises the variables, including the output database filename, initialises the states of the components and populates the popup menus that control the parameters of *Database* by calling *set_popupmenus*.

set_popupmenus:

The function *set_popupmenus* updates the popup menus that control the four parameters of *Database* and the input database filename. First, it gets the contents of the current directory of the signal filename. Then, it selects all directory names and all wave filenames from these contents and displays them in the signal popup menu. After this, it gets the contents of the current directory of the noise filename. Next, it selects all directory names and all wave filenames from these contents and displays them in the noise popup menu. In addition, it adds an empty field to the noise popup menu to make it possible to select no input noise. Subsequently, it displays the standard values for the SNR in the SNR popup menu and the standard values for the RMS in the RMS popup menu. After that, it gets the contents of the current directory of the input database filename. Finally, it selects all directory names and all text filenames from these contents and displays them in the input database popup menu.

1 *uipanel1_SelectionChangeFcn:*

The function u*ipanel1_SelectionChangeFcn* selects exclusively radiobutton1 or radiobutton2. If radiobutton1 is selected, it first gets the values of the four parameters from their popup menus. Next, it updates the states of the components and it sets *Database* in writing mode. If radiobutton2 is selected, it first gets the input database filename from its popup menu. Then, it updates the states of the components and it sets *Database* in reading mode.

2 *popupmenu1_Callback:*

The function *popupmenu1_Callback* gets the new input signal filename from its popup menu and saves it to a temporary variable. The new input signal filename stays there until it is confirmed. First, *popupmenu1_Callback* gets the selected item from popupmenu1. Subsequently, it checks whether the selected item is a directory or a wave filename. If it is a directory, it changes the current directory of the signal filename to that directory and it calls *set_popupmenus* to update popupmenu1 with the new contents. If it is a wave filename, it saves this filename to a temporary variable.

3 *popupmenu2_Callback:*

The function *popupmenu2_Callback* gets the new input noise filename from its popup menu and saves it to a temporary variable. The new input noise filename stays there until it is confirmed. *Popupmenu2_Callback* works in the same way as *popupmenu1_Callback*.

4 *popupmenu3_Callback:*

The function *popupmenu3_Callback* gets the new SNR from its popup menu and saves it to a temporary variable. The new SNR stays there until it is confirmed. First, *popupmenu3_Callback* gets the selected SNR value from popupmenu3. Then, it saves this value to a temporary variable.

5 *popupmenu4_Callback:*

The function *popupmenu4_Callback* gets the new RMS from its popup menu and saves it to a temporary variable. The new RMS stays there until it is confirmed. *Popupmenu4_Callback* works in the same way as *popupmenu3_Callback*.

6 *pushbutton2_Callback:*

The function *pushbutton2_Callback* confirms the new values of the four parameters, which are stored in the temporary variables. First, it checks whether the new input signal filename in the temporary variable is different from the input signal filename that is currently used. If this is the case, the input signal filename is changed to the new filename. In addition, *SSTART* is set to zero to make sure that the function that reads the input signal file reads the new file from the beginning and *SPOW* is set to zero to make sure that the function that reads the input signal file knows that a new file is used. Next, it performs the same action for the input noise filename. After this, it checks whether the new SNR in the temporary variable is different from the SNR that is currently used. If this is the case, the SNR is changed to the new value. In addition,

GRMS and *GDB* are set to zero to make sure that the function that mixes the input signal and the input noise knows that a new SNR is used. Subsequently, it performs the same action for the RMS. After that, it writes the time, the input signal filename, the input noise filename, the SNR and the RMS to the output database file if *Database* is in writing mode. Finally, it updates the states of the components if that is necessary.

7 *pushbutton3_Callback:*

The function *pushbutton3_Callback* starts playing the output signal. First, it writes the time and that the action taken was start playing to the output database file and updates the states of the components, if *Database* is in writing mode. Then, it starts playing the output signal by calling *player*.

8 *pushbutton4_Callback:*

The function *pushbutton4_Callback* stops playing the output signal. First, it stops and deletes a timer that is needed for playing the output signal. Next, it stops the audio player. After this, it resets some variables that are used by the function that reads the input signal file and the input noise file and that mixes them. Subsequently, it writes the time and that the action taken was stop playing to the output database file and updates the states of the components, if *Database* is in writing mode. Finally, it changes the output database filename.

player:

Player plays the output signal. First, it stops and deletes the timer that is needed for playing the output signal, if this timer exists. Then, it calls *AudioRead*. This function reads ten seconds from the input signal file and the input noise file from the points where it halted the last time when it was called. Then, it mixes the input signal and the input noise with the specified SNR and RMS to get the output signal. Next, *player* makes a new audio player with the output signal. After this, it waits until the audio player from the last time *player* was called has stopped running. Then, it starts playing the new audio player. Subsequently, it initialises a timer that calls *player* after a start delay. This start delay is equal to the duration of the audio player that has just started minus 0.5. Finally, it starts the timer that was initialised before.

AudioRead:

AudioRead reads ten seconds from the input signal file and the input noise file from the points where it halted the last time when it was called. Then, it mixes the input signal and the input noise with the specified SNR and RMS to get the output signal. First, it reads the size of the input signal file. Then, it calculates the points in samples where it has to start and stop reading the input signal file. After this, it checks whether the start point is smaller than the input signal size. If this is not the case, the start point and the stop point are recalculated. Next, it checks whether the stop point is smaller than the input signal size. If this is not the case, the stop point is recalculated. Subsequently, the input signal file is read from the start point to the stop point. Then, the duration of the input signal fragment is saved. After that, the output signal is set to the input signal fragment and the start point for reading the input signal in the next call of *AudioRead* is set, if no input noise file is selected. If an input noise file is selected, *AudioRead* reads the size of the input noise file. Then, it calculates the points in samples where it has to start and stop reading the input noise file. After this, it checks whether the start point is smaller than the input noise size. If this is not the case, the start point and the stop point are recalculated. Next, it checks whether the stop point is smaller than the input noise size. If this is not the case, the stop point is recalculated. Subsequently, the input noise file is read from the start point to the stop point. Then, the duration of the input noise fragment is saved. After that, the input signal fragment is shortened if the input noise fragment is shorter than the input signal fragment. Subsequently, the start point for reading the input signal and the start point for reading the noise signal in the next call of *AudioRead* are set. Next, *AudioRead* calculates the power of the input signal if a new file is used for it with the function *rctGetPow*. Then, it calculates the power of the input noise if a new file is used for it. After this, the input signal fragment and the input noise fragment are

mixed with the specified SNR. Finally, the output signal is scaled to the specified RMS. The power of the output signal needs to be recalculated for this purpose, if a new SNR or RMS is used.

9 *popupmenu5_Callback:*

The function *popupmenu5_Callback* gets the input database filename from its popup menu. First, it gets the selected item from popupmenu5. Subsequently, it checks whether the selected item is a directory or a text filename. If it is a directory, it changes the current directory of the input database filename to that directory and it calls *set_popupmenus* to update popupmenu5 with the new contents. If it is a text filename, it saves this filename to the variable for the input database filename.

10 *pushbutton5_Callback:*

The function *pushbutton5_Callback* starts reading the input database file. First, it updates the states of the components. Next, it gets the input database filename from its popup menu and it opens this file for reading. Finally, it starts the database reader by calling *reader*.

11 *pushbutton6_Callback:*

The function *pushbutton6_Callback* stops reading the input database file. First, it stops and deletes a timer that is needed for reading the input database file. Next, it stops playing the output signal. Then, it updates the states of the components.

reader:

Reader reads the database. First, it stops and deletes the timer that is needed for reading the input database file, if this timer exists. Next, if the *reader* is called for the first time, it reads from the input database file the first action that has to be performed and it saves this action to a first group of variables. The first action is the time when *reader* has to change the values of the four parameters and the new values of the parameters. Then, it reads from the input database file the second action that has to be performed and it saves this action to a second group of variables. The second action can be the time when *reader* has to start playing the output signal, the time when it has to stop playing the output signal or the time when it has to change the values of the four parameters and the new values of the parameters. If the *reader* is not called for the first time, it moves the contents of the second group of variables to the first group of variables. Then, it reads from the input database file the first new action that has to be performed and it saves this action to the second group of variables. The first new action can be the time when *reader* has to start playing the output signal, the time when it has to stop playing the output signal or the time when it has to change the values of the four parameters and the new values of the parameters. After this, *reader* moves the four parameters in the first group of variables to the temporary variables. Subsequently, it waits until the time has come to perform the action in the first group of variables. Then, it performs this action. If the action in the first group of variables is to start playing the output signal, then it calls *pushbutton3_Callback* to perform this action. If the action in the first group of variables is to stop playing the output signal, then it calls *pushbutton4_Callback* to perform this action and after that, it is finished. If the action in the first group of variables is to change the values of the four parameters, then it calls *pushbutton2_Callback* to perform this action. Next, *reader* initialises a timer that calls *reader* after a start delay. This start delay is equal to the time between the action in the second group of variables and the action in the first group of variables minus one. Finally, it starts the timer that was initialised before.

4.2 The input fragments player

A program called *Inputplayer* is developed that makes it possible to play an input audio signal fragment that was saved by the warped FFT framework. *Inputplayer* has a GUI that makes it simple to operate it and it is developed in Matlab with GUIDE. First, an input audio signal fragment filename can be specified in *Inputplayer*. After that, the parameter settings and the

user feedback belonging to the specified input audio signal fragment are read from the file that was written at the same time as the input audio signal fragment file and are then displayed. Now, *Inputplayer* can start or stop playing the specified input audio signal fragment. This fragment can be played in normal mode or in loop mode, in which the input audio signal fragment is repeated until it is stopped. The layout of *Inputplayer* can be seen in figure 4.2 and the layout file is called *Inputplayer.fig*. *Inputplayer* uses *Inputplayer.m* as the source file to control the layout file. The basic functions of *Inputplayer*, such as opening function and output function, can be found in *Inputplayer.m*. In addition, the functions belonging to the components of *Inputplayer*, such as text boxes and buttons, can also be found in *Inputplayer.m*. These functions are executed when their corresponding components are used. The other source file that *Inputplayer* uses is called *convert.m*. It is now described how *Inputplayer* works. The parts of the source files that are important for understanding this can be seen in appendix I.

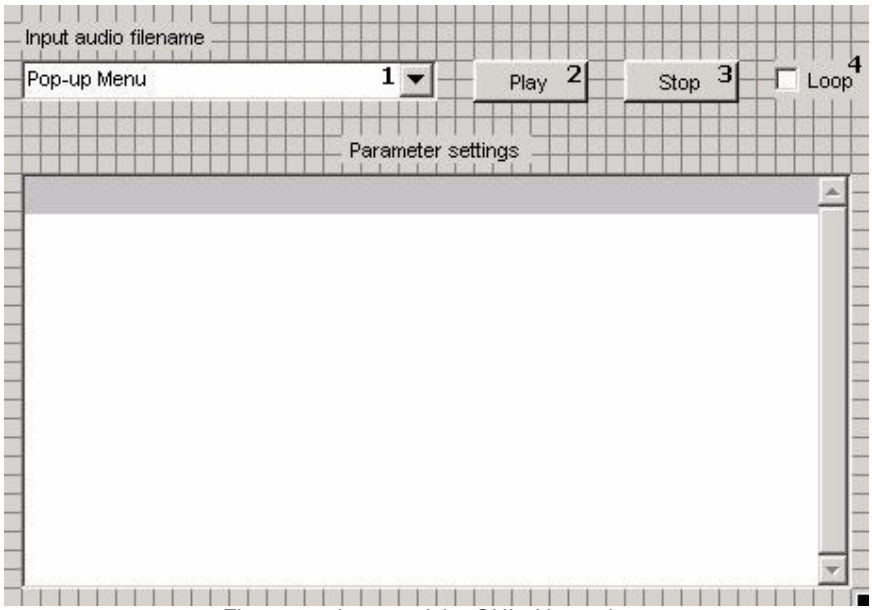

Figure 4.2: Layout of the GUI of Inputplayer

Inputplayer_OpeningFcn:

The function *Inputplayer_OpeningFcn* is executed just before *Inputplayer* is made visible. This function initialises the variables, initialises the states of the components and populates the popup menu that controls the input audio signal fragment filename by calling *set_popupmenu*.

set_popupmenu:

The function *set_popupmenu* updates the popup menu that controls the input audio filename. First, it gets the contents of the current directory of the input audio filename. Then, it selects all directory names and all text filenames that start with audio_input_ from these contents and displays them in the input audio popup menu.

1 *popupmenu1_Callback:*

The function *popupmenu1_Callback* gets the input audio filename from its popup menu, reads the parameter settings and the user feedback belonging to the specified input audio file from the file that was written at the same time as the input audio file and displays the parameter settings and the user feedback. First, *popupmenu1_Callback* gets the selected item from popupmenu1. Subsequently, it checks whether the selected item is a directory or a text filename. If it is a directory, it changes the current directory of the input audio filename to that directory and it calls *set_popupmenu* to update popupmenu1 with the new contents. If it is a text filename, it gets the contents of the current directory of the input audio filename. Then, it looks in every text file that starts with log_ for the input audio filename. If it is found, the

parameter settings and the user feedback that are written in the text file and that belong to the input audio filename are displayed in listbox1. If the input audio filename is not found, an error description is written to listbox1. Finally, *popupmenu1_Callback* updates the states of the components.

2 *pushbutton1_Callback:*

The function *pushbutton1_Callback* starts playing the input audio signal fragment. First, it updates the states of the components. Then, it plays the input audio signal fragment by calling *convert*. This function opens the input audio file, scans the audio data in this file, plays the audio data and then closes the input audio file. Next, *pushbutton1_Callback* pauses until the input audio signal fragment has stopped playing. After this, it repeats the last two steps if *Inputplayer* is in loop mode. Finally, it updates the states of the components.

3 *pushbutton2_Callback:*

The function *pushbutton2_Callback* stops playing the input audio signal fragment.

4 *checkbox1_Callback:*

The function *checkbox1_Callback* sets *Inputplayer* in normal mode or in loop mode.

5. Evaluation of the PNR algorithm with RTProc

In this chapter is the evaluation of the PNR algorithm described that was performed as example of an application of the RTProc system. It was performed in real-time by using the warped FFT framework for RTProc, the database creator and the input fragments player. First, two methods are described that can be used to perform the evaluation of the PNR algorithm. Then, the experimental design is described that was used to perform the evaluation. Subsequently, the results of the evaluation are given. Finally, some conclusions are drawn about the PNR algorithm.

5.1 Evaluation methods

The first method creates a database with audio fragments that are interesting for the PNR algorithm by listening to the real-time output of the warped FFT framework for RTProc that has the PNR algorithm enabled. This method works as follows. The database creator creates a database that is used in real-time as input for the warped FFT framework for RTProc that has the PNR algorithm enabled. The user evaluates the real-time output of the system and he changes the parameters of the database creator in real-time by utilizing the evaluation. In so doing, a database is created with audio fragments that are interesting for the PNR algorithm. The database consists of a radio play mixed with a defined SNR with various noise sources and the noise sources include for example typing, babble, factory, talking and music. The *gain depth* parameter of PNR is kept constant with this method.

The second evaluation method imitates walking around with the RTProc system on a laptop. This method consists of the following steps. First, an audio database is created with the database creator, to imitate the input audio the user will get if he walks around with the RTProc system on a laptop. This database consists of a radio play mixed with a defined SNR with various noise sources. The noise sources include for example typing, babble, factory, talking and music. Next, the audio database is used as input for the warped FFT framework for RTProc that has the PNR algorithm enabled. The user listens to the real-time output of the system and he switches the *gain depth* parameter between the values -3, -6 and -9. Now, when an interesting event occurs, the user pushes the save data button and the input audio signal fragment of the last ten seconds, the parameter settings and the user feedback are saved. After this, the obtained input audio signal fragments are played with the input fragments player. The user now listens again very carefully to the real-time output of the warped FFT framework for RTProc with PNR enabled and he rates every output audio signal fragment with a QR, for a *gain depth* of -3, -6 and -9. The QR depends on the intelligibility and the distortion of the speech. This method makes it possible to tune the *gain depth* parameter setting of PNR and to analyze the behaviour of PNR for various noise sources.

5.2 Experimental design

The second evaluation method of section 5.1 has been used to perform the evaluation of the PNR algorithm. This evaluation method is described now in more detail. I have been the user for this evaluation.

An audio database of half an hour has been created with the database creator. This database consists of a radio play of *Lord of the Rings* mixed with various noise sources. These noise sources are:

- Noises, namely *airplane*, *ambulance*, *babble*, *car*, *church*, *factory*, *hammer*, *helicopter*, *knock*, *motorbike*, *music_hall*, *phone*, *saw*, *shower*, *speech shaped*, *supermarket*, *traffic* and *typing*.
- Speech, namely *annemarie*, *bbcnews*, *biptips*, *child*, *jenny* and *scholars*.
- Music, namely *baaba maal sidiki*, *bach two violins*, *eagles hotel california*, *eminem lose yourself*, *enigma gregorian chant*, *enya only time*, *pavaroti ave maria*, *philip glass there are some men*, *scooter weekend*, *shapplin spente le stelle* and *vega*.

The audio in the database can have a SNR of -15, -10, -5, 0, 5 and 10. The RMS value is equal to *NaN*, which means that the audio data is scaled so that it is as loud as possible without clipping.

The audio database has been used as input for the warped FFT framework for RTProc that has the PNR algorithm enabled. First, the database is played and PNR has a *gain depth* of -3, next, the database is played and PNR has a *gain depth* of -6 and then, the database is played and PNR has a *gain depth* of -9. One user listens to the real-time output of the system and when an interesting event occurs, he pushes the save data button and the input audio signal fragment of the last ten seconds, the parameter settings and the user feedback are saved.

The obtained input audio signal fragments are played with the input fragments player. The same user now listens again very carefully to the real-time output of the warped FFT framework for RTProc with PNR enabled and he rates every output audio signal fragment with a QR, for a *gain depth* of -3, -6 and -9. The QR depends on the intelligibility and the distortion of the speech. It can have five categorical values, namely 1 for bad, 2 for poor, 3 for fair, 4 for good and 5 for excellent. The final QR for a specific noise type and a specific *gain depth* is the average of the QRs of all input audio signal fragments of that noise type and that *gain depth*.

5.3 Results

It is expected that the PNR algorithm with a *gain depth* of -9 gives the best QR. However, this setting will also result in the highest distortion of the speech. In addition, it is expected that the PNR algorithm will perform better on signal fragments with stationary noise than on signal fragments with other types of noise, such as non-stationary noise or talking.

The results of the evaluation of the PNR algorithm can be seen in table 5.1 and figure 5.1. Table 5.1 displays the QR for signal fragments with various types of noise for a *gain depth* of -3, -6 and -9. The same results can be seen in a graphical representation in figure 5.1. The detailed results of the evaluation of the PNR algorithm can be seen in appendix J. It can be seen that a *gain depth* of -9 gives the best average QR, that a *gain depth* of -6 gives an average QR that is also reasonably good, but that a *gain depth* of -3 gives an average QR that is somewhat worse. However, the QR for a *gain depth* of -9 is worse than the QR for a *gain depth* of -6 for some noise types. This is because the distortion of the speech has become audible for a *gain depth* of -9 for these noise types. It can also be seen that the results of the evaluation of the PNR algorithm can be divided into four categories. The first category is for noise types on which the PNR algorithm has little to no effect, the second category is for noise types on which the PNR algorithm has a positive effect that can already be heard for low values of the *gain depth* and that does not become much larger for higher values of the *gain depth*, the third category is for noise types on which the PNR algorithm has a positive effect that can only be heard for higher values of the *gain depth* and the fourth category is for noise types on which the PNR algorithm has a positive effect that is the largest for a moderate value of the *gain depth*. *Typing*, *shower*, *saw*, *hammer*, *phone*, *male speaker*, *female speaker*, *child speaker* and *female singer* fall into the first category, *motorbike*, *airplane*, *speech shaped* and *pop music* fall into the second category, *church bell*, *music hall* and *classical music* fall into the third category and *babble* and *factory* fall into the fourth category.

Noise type	Gain depth			
	-3	-6	-9	
	QR			
Typing	3	3	3	
Motorbike	2	3	3	
Shower	2	2	2	
Saw	3	3	3	
Hammer	1	1	1	
Babble	2	2	1	
Church bell	2	2	3	
Airplane	1	2	2	
Phone	2	2	2	
Music hall	3	3	4	
Factory	2	3	2	
Speech shaped	1	2	2	
Male speaker	2	2	2	
Female speaker	2	2	2	
Child speaker	3	3	3	
Female singer	3	3	3	
Classical music	3	3	4	
Pop music	2	3	3	
	2.17	2.44	2.50	**Average**

Table 5.1: Results of the evaluation of PNR

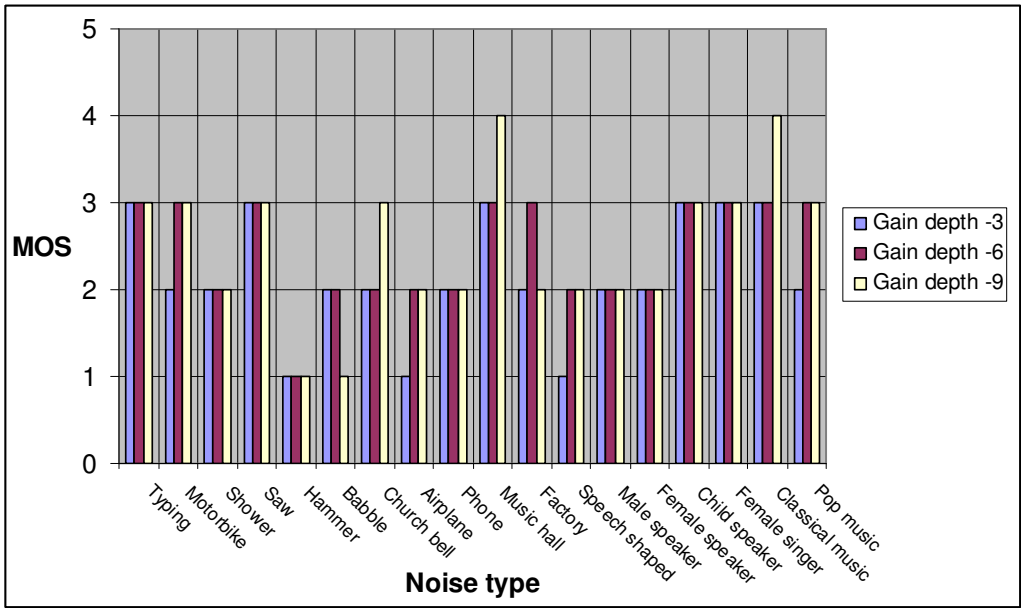

Figure 5.1: Results of the evaluation of PNR

5.4 Conclusions

The expected results of the evaluation of the PNR algorithm match the final results of the evaluation. A *gain depth* of -9 gives the best average QR and a *gain depth* of -6 gives an average QR that is also reasonably good. However, the QR for a *gain depth* of -9 is worse than the QR for a *gain depth* of -6 for some noise types. This is because the distortion of the

speech has become audible for a *gain depth* of -9 for these noise types. In addition, PNR only has an effect on signal fragments with a specific noise type. For non-stationary noise such as *typing*, *hammer* or *talking*, PNR has little to no effect. For noise on the foreground with a time constant spectrum such as *airplane* or *speech shaped*, PNR has a positive effect that can already be heard for low values of the *gain depth* and that does not become much larger for higher values of the *gain depth*. For noise with a time varying spectrum such as *musical hall* or *classical music*, PNR has a positive effect that can only be heard for higher values of the *gain depth*. For noise on the background with a time constant spectrum such as *babble* or *factory*, PNR has a positive effect that is the largest for a moderate value of the *gain depth*. This division of noise types into four categories is very provisional, because much more evaluations have to be done with many users and many signal fragments before a definite division can be given. However, the evaluation shows that it appears to be useful to characterize the noise type in a signal fragment before PNR is applied and then to adjust the *gain depth* according to the characterization of the noise type.

6. Conclusions and recommendations

In this chapter are first some conclusions drawn about the use of the warped FFT framework with GN ReSound algorithms for RTProc for real-time audio processing. Next, some recommendations are given for further developments on RTProc, the warped FFT framework and the GN ReSound algorithms.

6.1 Conclusions

The RTProc system is a real-time simulation platform that uses a PC for real-time audio processing. It is written in C++ and consists of three modules, namely a module that operates the audio device, a module that processes the input signal by applying an algorithm and a module that takes care of the communication between the other two modules. The last module is operated by means of a number of commands in Matlab. These adjustments were made to RTProc to make it suitable for evaluating HA algorithms in real-time:

- The first adjustment is that the warped FFT framework was coded in C++. It is a DLL that can be used by the RTProc system as an algorithm component. It performs a warped FFT on an input audio signal, resulting in spectral coefficients. Spectral log-power coefficients are then calculated from these spectral coefficients and these spectral log-power coefficients are given to an audio processing node in a DLL called GN ReSound algorithms. This audio processing node calculates spectral log-gain factors by applying some specific algorithms designed by GN ReSound. Subsequently, spectral linear-gain factors are calculated from the spectral log-gain factors in the warped FFT framework and an IFFT is performed to transform these spectral linear-gain factors into gain factors in the time domain. Finally, the warped FFT framework applies a warped FIR filter to the input signal, using the gain factors in the time domain. This results in the output audio signal. In addition, the warped FFT framework is capable of saving input audio signal fragments of the last ten seconds.
- The second adjustment is that a GUI called *WarpedFFT* was coded in Matlab. It can operate the RTProc system that uses the warped FFT framework as algorithm component and GN ReSound algorithms as audio processing node. *WarpedFFT* selects the correct audio device and sets its settings and then selects the warped FFT framework as algorithm component and GN ReSound algorithms as audio processing node. Moreover, it is possible with *WarpedFFT* to do the following during runtime. Some settings of the audio device and the warped FFT framework can be adjusted, algorithms in GN ReSound algorithms can be enabled or disabled and their parameters can be adjusted, an internal state of an algorithm in GN ReSound algorithms can be monitored and the input audio signal fragment of the last ten seconds, the parameter settings and the user feedback can be saved.

In addition, the method was documented for adding an algorithm to the warped FFT framework with GN ReSound algorithms and subsequently updating the GUI that operates the framework with the algorithms. This documentation is needed because it has to be very simple to add an algorithm and to update the GUI.

Two database utilities have also been created that can be used with the RTProc system. The first utility is called *Database* and this utility makes it possible to create instantly and audio database for evaluating an algorithm in real-time. The second utility is called *Inputplayer* and this utility makes it possible to play an input audio signal fragment that was saved by the warped FFT framework.

As example of an application of the RTProc system, the PNR algorithm was evaluated in real-time by using the warped FFT framework for RTProc, the database creator and the input fragments player. This evaluation shows that the RTProc system is very useful. Many possible applications of the RTProc system can be thought of, such as the two described in section 5.1. However, there are currently some limitations attached to the use of the RTProc system. These limitations are that the RTProc system is currently not suitable for walking

around with it on a laptop, the method that is used currently for saving the input audio signal fragment of the last ten seconds is not optimal and it disturbs the real-time processing and the user feedback that is saved currently by *WarpedFFT* is limited to the QR.

6.2 Recommendations

A very important future application of the RTProc system is the following. A user walks around with a laptop that runs the warped FFT framework for RTProc, which has one or several GN ReSound algorithms enabled. The GUI that operates the warped FFT framework with GN ReSound algorithms is coded on a PDA. The RTProc system and the GUI on the PDA communicate with each other via Matlab. The user listens to the real-time output of the system and can change the parameters of the enabled algorithms on the PDA. Now, when an interesting event occurs, the user pushes the save data button on the PDA and the input audio signal fragment of the last ten seconds, the parameter settings and the user feedback are saved. Next, the obtained input audio signal fragments are played with the input fragments player, which makes it possible for the user to listen again very carefully to the real-time output of the RTProc system. This real-time output is used then to tune the parameter settings of an algorithm or to redesign the algorithm if it is not functioning properly. However, this application is not possible yet, because the RTProc system is not suitable currently for walking around with it on a laptop.

The following recommendations for further developments are given to make the aforementioned application possible, to add some other features to the RTProc system that are still missing and to improve some weak features of the RTProc system:

- It is necessary that the code of the whole RTProc system is obtained by GN ReSound and not only the code of the algorithm module. This makes the system more flexible and more transparent and because of this, the help from RWTH Aachen is less frequently needed for things like compiling the RTProc system for a different version of Matlab or troubleshooting.
- The QR of the output audio signal that is currently used as user feedback has to be replaced by a number of questions or statements about the quality of the output audio signal, such as:

    - The HA sounds too loud.
    - The HA sounds too soft.
    - The HA sounds too noisy.
    - The HA is whistling.
    - The speech cannot be understood.

    This is because these questions give a better indication about the quality of the output audio signal.
- The method that is currently used for saving the input audio signal fragment of the last ten seconds has to be replaced by a better method, because the method that is currently used disturbs the real-time processing. The new method should not disturb the real-time processing. It has to be possible also to save the output audio signal fragment of the last ten seconds. Due to this, the user does not need to use the RTProc system constantly to obtain the same output audio signal, because this output can now be available offline. In addition, it has to be possible to save the input or output audio signal fragment of the last ten seconds by calling a MEX function in the Matlab host that is dedicated to this action.
- The RTProc system has to be made suitable for walking around with it on a laptop. This means that the GUI that can operate the RTProc system has to be reprogrammed for a PDA. In addition, the PDA has to be able to communicate with Matlab on the laptop. See appendix K for a more detailed description of how the RTProc system can be made suitable for portable use.
- The RTProc system is suitable for sophisticated tuning and machine learning applications. For this purpose, algorithms have to be implemented in the warped FFT framework for RTProc that can make direct adaptations to the parameters based on the current parameter settings, the signal features and the user feedback. Examples

of such algorithms are a noise reduction algorithm or a volume control that can be automatically tuned.

- More applications of the RTProc system have to be developed. These new applications have to be for both using the system on a desktop and walking around with it on a laptop. However, walking around with the RTProc system on a laptop will be the most important application.

7. Bibliography

[1] Berg, van den, A.J. Curvilinear AGCI MSD. Internal report GN ReSound. 11 April 2005.

[2] Kates, James M. Dynamic range compression using digital frequency warping. Internal report AudioLogic/Cirrus Logic. 18 December 2000.

[3] Krüger, Hauke and Peter Vary. RTProc: Integration for MATLAB, Technical Reference. Institute of Communication Systems and Data Processing, RWTH Aachen, Germany. Version 0.0.1, preliminary version.

[4] Pedersen, Brian Dam. Compknee MSD. Internal report GN ReSound. 12 June 2004.

[5] Vries, de, Bert and Almer van den Berg. Perceptual Noise Reduction MSD. Internal report GN ReSound. 14 June 2004.

A. Enumeration types for the RTProc system

Format name	Description
RTP_CHANNELFORMAT_NONE	No specific format, can not be used for any purpose
RTP_CHANNELFORMAT_DOUBLE	`double` format according to C double type.
RTP_CHANNELFORMAT_FLOAT	`float` format according to C float type.
RTP_CHANNELFORMAT_LITTLE_ENDIAN_16BIT	Format for PC signed short data.
RTP_CHANNELFORMAT_LITTLE_ENDIAN_24BIT	24 bit dataformat, unused so far.
RTP_CHANNELFORMAT_LITTLE_ENDIAN_32BIT	Format for PC `signed int` data.
RTP_CHANNELFORMAT_VOID	Extensions available for this type in future applications.

Table A.1: Available data formats for type RTP_CHANNELFORMAT

Error code name	Description
RTP_ERROR_NOERROR	Indication that there is no error pending currently.
RTP_ERROR_INTERNAL	The error is due to a failure in the internal structure of a component. E.g. if a driver technology returns this error, the error was in the driver itself.
RTP_ERROR_NOT_READY	The function call is correct but the component is unexpectedly not yet ready for the call.
RTP_ERROR_NOT_COMPATIBLE	The parameters passed can not be accepted as the underlying system is incompatible to the passed parameters.
RTP_ERROR_WRONGSTATEFORCALL	The function has been called in the wrong state of a component.
RTP_ERROR_IDMISMATCH	When selecting a subcomponent/parameter by specifying an ID, the ID was outside of the range of the entries (end of list).
RTP_ERROR_VOIDDATASIZEMISMATCH	When using the function calls for extended functionality with the open interfaces, the data passed did not have the right size (kind of light error checking)
RTP_ERROR_MISC	Any other error that has none of the specified reasons.

Table A.2: Available error codes for type RTP_ERROR

State Name	Description
RTP_P_INIT	State in which the component is at the very beginning. There are in general functions that succeed in each of the possible states that also succeed in this state.
RTP_P_SELECTED	In state selected the user has indicated that he may activate it afterwards. In this state generic runtime messages can be passed in order to modify parameters that are required to be set already during activation.
RTP_P_ACTIVE	An activated component will list up all available functions to return and to edit setup for realtime processing. From here the

	component can be set into the next state.
RTP_P_READY	An active component has been set to this state by preparing for processing. In this state the processing parameters are fixed but the component simply is not yet in processing mode. It is kind of paused state for processing.
RTP_P_PROCESSING	In this state the current component is in the middle of audio processing. Especially the processing main routine may be called in a realtime priority thread.

Table A.3: Available states for type RTP_P_STATE

Name of the capability	Description
RTP_IS_INPUT	The device is only suited for recording (simplex: input).
RTP_IS_OUTPUT	The device is only suited for playback (simplex: output).
RTP_IS_DUPLEX	The device can be used for audio realtime processing.

Table A.4: Available capabilities for type RTP_CAPABILITY

Name for RTP message	Description
RTP_ID_STOPPROCESSING	Function to inform the host to stop processing, initiated by the driver.

Table A.5: Available messages in IhostreffromdriverIND

Name for RTP message	Description
RTP_ID_STOPPROCESSING	Function to inform the host to stop processing, initiated by the algorithm.

Table A.6: Available messages in IhostreffromalgorithmIND

Name of datatype	ID	Description
RTP_GEN_CONF_ELEMENT_DOUBLE	0	Element is a double field (single element)
RTP_GEN_CONF_ELEMENT_DOUBLE_ARRAY	1	Element is a double field with more than one element.
RTP_GEN_CONF_ELEMENT_DOUBLE_ARRAY_EXTERN	2	Element is a double field with an arbitrary number of values. It must be allocated outside the component where it comes from. The pointer to the field should be set to NULL initially.
RTP_GEN_CONF_ELEMENT_INT32	3	Element is an int field (single element)
RTP_GEN_CONF_ELEMENT_INT32_ARRAY	4	Element is an int field with more than one element.
RTP_GEN_CONF_ELEMENT_INT32_ARRAY_EXTERN	5	Element is an int field with an arbitrary number of values. It must be allocated outside the component where it comes from. The pointer to the field should be set to NULL initially.
RTP_GEN_CONF_ELEMENT_STRING_LIST	6	Element is a pointer to a

		field of string elements.
RTP_GEN_CONF_ELEMENT_STRING_LIST_EXTERN	7	Element is a pointer to a field of string elements, field must be allocated externally to component. The pointer to the field should be set to NULL initially.
RTP_GEN_CONF_ELEMENT_SELECTION	8	Pointer to a single field for a selector. There can only be one such field.
RTP_GEN_CONF_ELEMENT_VALUE_IN_RANGE	10	Pointer to a single field for a value in range struct. There can only be one such field.
RTP_GEN_CONF_ELEMENT_CONFELEMENT	12	This element is for recursive datapacking. If in a component another component is available the component returns that field as pointer to type rtpGenericConfigure as well.
RTP_GEN_CONF_ELEMENT_CALLBACK_EXTERN	13	Future type for setting callback pointers. There is another interface for this purpose. As it is not used yet, it also is not documented yet.
RTP_GEN_CONF_ELEMENT_EXTENSION	14	Extension element for future use.
RTP_GEN_CONF_ELEMENT_VOID	15	Non specified datafield for future usage.
RTP_GEN_CONF_ELEMENT_VOID	16	Field to hold an invalid pointer. If ever using NULL pointers for temporary reasons, this field should be set to this type.

Table A.7: Available data types for type RTP_GEN_CONF_TYPE

Name	ID	Description
RTP_GEN_CONF_SET	0	Purpose of this element is to take the value passed in the datafield and integrate internally in component (set the data in component).
RTP_GEN_CONF_GET	1	Purpose of this element is to fill the field passed in the datafield in order to return the component internal value (get the data in component).
RTP_GEN_CONF_IGNORE	2	Ignore the datafield that has this purpose.

Table A.8: Available purposes for type RTP_GEN_CONF_PURPOSE

Name	ID	Description
RTP_GEN_CONF_ELEMENT_SUCCESS	0	Operation to set data has been successfully completed.
RTP_GEN_CONF_ELEMENT_UNKNOWN	1	Operation failed with unknown reason (try to display the error message returned).
RTP_GEN_CONF_TYPE_INVALID	2	Operation failed due to wrong type of passed data.

RTP_GEN_CONF_SIZE_INVALID	3	Operation failed due to wrong size of passed data (esp. for external fields).
RTP_GEN_CONF_WROMG_STATE	4	Operation failed due to wrong state of component.
RTP_GEN_CONF_UNKNOWN	5	Result is unknown, should be on entry only.

Table A.9: Available results for type RTP_GEN_CONF_RESULT

Name	ID	Description
RTP_GEN_CONF_STRING_FILENAME	0	The string passed is a filename. A different dialog can be opened to browse directories etc.
RTP_GEN_CONF_STRING_NUMBER	1	The passed string is a number.
RTP_GEN_CONF_EXCEL_CSV	2	The passed string is in excel format and can be used for any purpose.
RTP_GEN_CONF_STRING_NONE	3	The string has no specific format.

Table A.10: Available string types for type RTP_GEN_CONF_STRING_TYPE

B. Enumeration types for the Matlab host

System Command Name	ID	Description
RTP_MX_SYSTEM_GET_STATE_SYSTEM	0	Command to return the current state of the system.
RTP_MX_SYSTEM_GET_STATE_HW	1	Command to return the current state of the subsystem Hardware (Technology and associated Device).
RTP_MX_SYSTEM_GET_STATE_SW	2	Command to return the current state of the subsystem Software (Algorithm Component)
RTP_MX_SYSTEM_INITIALIZE	3	Command to initialize the host. When first calling the matlabHost interface function the host is in uninitialized state.
RTP_MX_SYSTEM_ADD_RTP_PATH	4	Command to pass a directory where to search for technology and algorithm components. If more than one path shall be added, use this command for each path.
RTP_MX_SYSTEM_CLEAR_RTP_PATH	5	Command to clear all paths from the list of paths to search for components at.
RTP_MX_SYSTEM_LOAD_COMPONENTS	6	Command to seach the specified directories for components. All technologies and algorithms will be stored internally afterwards and the user can select among all available components.
RTP_MX_SYSTEM_SET_SAMPLERATE_OFFLINE	7	Command to set a samplerate for offline processing (feed algorithm with data manually). The samplerate must be specified as no hardware can be used to specify samplerate.
RTP_MX_SYSTEM_GET_SAMPLERATE_OFFLINE	8	Return the samplerate specified for offline processing previously. Initially this value is specified to be 0.
RTP_MX_SYSTEM_SET_BUFFERSIZE_OFFLINE	9	Command to set a buffersize for offline processing (feed algorithm with data manually). The buffersize must be specified as no hardware can be used to specify buffersize.
RTP_MX_SYSTEM_GET_BUFFERSIZE_OFFLINE	10	Return the buffersize specified for offline processing previously. Initially this value is specified to be 0

RTP_MX_SYSTEM_SET_FORMAT_PROCESSING_ OFFLINE	11	Command to set a processing format for offline processing (feed algorithm with data manually). The format must be specified as no hardware can be used to specify processing format.
RTP_MX_SYSTEM_GET_FORMAT_PROCESSING_ ACTIVE	12	Return the processing format previously specified for offline processing. This value is initialized to process format RTP_CHANNEL_FORMAT_ DOUBLE.
RTP_MX_SYSTEM_SET_INPUT_CHANNELS_ PROCESSING_OFFLINE	13	Set the number of input channels for offline processing. As there is no instance to set the name for the channels we simply can specify a number.
RTP_MX_SYSTEM_GET_INPUT_CHANNELS_ PROCESSING_OFFLINE	14	Return the number of input channels previously specified for offline processing. Initial value for number of channels is 0.
RTP_MX_SYSTEM_SET_OUTPUT_CHANNELS_ PROCESSING_OFFLINE	15	Set the number of output channels for offline processing. As there is no instance to set the name for the channels we simply can specify a number.
RTP_MX_SYSTEM_GET_OUTPUT_CHANNELS_ PROCESSING_OFFLINE	16	Return the number of output channels previously specified for offline processing. Initial value for number of channels is 0.
RTP_MX_SYSTEM_SET_OPERATION_MODE	17	Set the operation mode for processing. We can set values for offline, duplex or talkthrough processing
RTP_MX_SYSTEM_GET_OPERATION_MODE	18	Return the operation mode for processing previously set. The initial value for processing mode is duplex.
RTP_MX_SYSTEM_PREPARE_PROCESSING	19	Command to prepare for processing. According to the operation mode the device and/or the algorithm will be prepared for processing.
RTP_MX_SYSTEM_START_PROCESSING	20	Command to start processing.
RTP_MX_SYSTEM_STOP_PROCESSING	21	Command to stop processing.
'RTP_MX_SYSTEM_POST_PROCESSING	22	Command to set from state prepared to active. Opposite of prepare Processing functionality.
RTP_MX_SYSTEM_PROCESS_BUFFERS_OFFLINE	23	If in offline mode the algorithm must be driven manually. With this function a set of audio buffers (audio[N,M]; N:number channels input, M:buffersize audio) can be passed for processing.

RTP_MX_SYSTEM_UNLOAD_COMPONENTS	24	Opposite of the load function. If components are loaded they are unloaded.
RTP_MX_SYSTEM_TERMINATE	25	Terminate the system: Set into state INITIALIZED again.
RTP_MX_HW_GET_NUMBER_TECHNOLOGIES	64	Command to list all available technologies.
RTP_MX_HW_GET_ID_TECHNOLOGY__ID	65	Command to return the ID for a technology. The ID is according to audioProcessingTypedefs.h.
RTP_MX_HW_GET_NAME_TECHNOLOGY__ID	66	Return the name of one of the technologies-
RTP_MX_HW_GET_DESCRIPTION_TECHNOLOGY__ID	67	Command to get the description for a specific technology.
RTP_MX_HW_CATEGORY_TECHNOLOGY__ID	68	Return the category associated with a technology (see audioProcessingTypedefs.h)
RTP_MX_HW_GET_STATE_TECHNOLOGY__ID	69	Return the state of a technology component, can be either INITIALIZED, ACTIVE or SELECTED.
RTP_MX_HW_SELECT_TECHNOLOGY__ID	70	Select one of the technologies available.
RTP_MX_HW_ACTIVATE_TECHNOLOGY	71	Activate the selected technology.
RTP_MX_HW_GET_NUMBER_DEVICES	72	Return the number of the devices available for an active technology.
RTP_MX_HW_GET_NAME_DEVICE__ID	73	Return the name of a specific device.
RTP_MX_HW_GET_DESCRIPTION_DEVICE__ID	74	Return the description of a specific device.
RTP_MX_HW_GET_CAPABILITY_DEVICE__ID	75	Command to return the category for a device: INPUT OUTPUT or DUPLEX.
RTP_MX_HW_IS_AVAILABLE_DEVICE__ID	76	Return the indication whether a device is available (for future)
RTP_MX_HW_GET_STATE_DEVICE__ID	77	Return the current state of a device (INITIALIZED, SELECTED, ACTIVE, PREPARED or PROCESSING)
RTP_MX_HW_SELECT_DEVICE__ID	78	Select one among the available devices.
RTP_MX_HW_ACTIVATE_DEVICE	79	Activate the selected device.
RTP_MX_HW_DEACTIVATE_TECHNOLOGY	80	Deactivate the currently active technology.
RTP_MX_HW_UNSELECT_TECHNOLOGY	81	Unselect an available technology.
RTP_MX_HW_GET_SAMPLERATES_DEVICE	82	Return the samplerates available for an active device.
RTP_MX_HW_SET_SAMPLERATE_REALTIME	83	Set the samplerate currently active with a specified device (stored as part of the device)
'RTP_MX_HW_GET_SAMPLERATE_REALTIME	84	Return the samplerate that the device provides. The device comes along with ist own specific initial value for this.
RTP_MX_HW_GET_BUFFERSIZES_DEVICE	85	Return the available buffersizes for a device.

RTP_MX_HW_SET_BUFFERSIZE_REALTIME	86	Set the buffersize currently active with a specified device (stored as part of the device)
RTP_MX_HW_GET_BUFFERSIZE_REALTIME	87	Return the buffersize that the device provides. The device comes along with ist own specific initial value for this.
RTP_MX_HW_GET_PROCESSING_FORMATS_ DEVICE	88	Return the available processing formats for a device.
RTP_MX_HW_SET_PROCESSING_FORMAT_ REALTIME	89	Set the active processing format for the active device.
RTP_MX_HW_GET_PROCESSING_FORMAT_ REALTIME	90	Return the currently active processing format for a device.
RTP_MX_HW_GET_INPUT_CHANNELS_DEVICE	91	Command to return a struct with all available input channels including the current channel state (active/inactive)
RTP_MX_HW_GET_OUTPUT_CHANNELS_DEVICE	92	Command to return a struct with all available output channels including the current channel state (active/inactive)
RTP_MX_HW_ACTIVATE_INPUT_CHANNEL_ DEVICE	93	Set a specific channel for input to active state.
RTP_MX_HW_DEACTIVATE_INPUT_CHANNEL_ DEVICE	94	Set a specific channel for input to inactive state.
RTP_MX_HW_ACTIVATE_OUTPUT_CHANNEL_ DEVICE	95	Set a specific channel for output to active state.
RTP_MX_HW_DEACTIVATE_OUTPUT_CHANNEL_ DEVICE	96	Set a specific channel for output to inactive state.
RTP_MX_HW_GET_BITS_INTERN_DEVICE	97	Command to return the resolution of the hardware internally.
RTP_MX_HW_DEACTIVATE_DEVICE	98	Deactivate the active device.
RTP_MX_HW_UNSELECT_DEVICE	99	Unselect the selected device.
RTP_MX_HW_GET_LATENCY_INPUT_DEVICE	100	Command to return the system latency for input for talkthrough (no additional software latency assumed)
RTP_MX_HW_GET_LATENCY_OUTPUT_DEVICE	101	Command to return the system latency for output for talkthrough (no additional software latency assumed)
RTP_MX_SW_GET_NUMBER_ALGORITHMS	128	Command to return the number of available algorithm components.
RTP_MX_SW_GET_NAME_ALGORITHM__ID	129	Function to return the name of one of the available algorithm components.
RTP_MX_SW_GET_DESCRIPTION_ALGORITHM__ ID	130	Command to return the description for a specified algorithm component.
RTP_MX_SW_SELECT_ALGORITHM__ID	131	Command to select one of the algorithm components available.
RTP_MX_SW_ACTIVATE_ALGORITHM	132	Activate the selected algorithm component.
RTP_MX_SW_GET_SETUPS_ALGORITHM	133	Return a struct containing all setups available for an algorithm.

RTP_MX_SW_GET_PROCESSING_FORMATS_ ALGORITHM	134	Return all processing formats for an active algorithm.
RTP_MX_SW_GET_LATENCY_ALGORITHM	135	Return the software latency for the algorithm.
RTP_MX_SW_DEACTIVATE_ALGORITHM	136	Deactivate the active algorithm.
RTP_MX_SW_UNSELECT_ALGORITHM	137	Unselect a selected algorithm.
RTP_MX_GENERIC_GET_INTERNAL_DATA_ ALGORITHM	256	Get the internal data for generic runtime configuration on the selected algorithm.
RTP_MX_GENERIC_SET_INTERNAL_DATA_ ALGORITHM	257	Set the internal data for generic runtime configuration on the selected algorithm.
RTP_MX_GENERIC_GET_INTERNAL_DATA_ TECHNOLOGY	258	Get the internal data for generic runtime configuration on the selected technology.
RTP_MX_GENERIC_SET_INTERNAL_DATA_ TECHNOLOGY	259	Set the internal data for generic runtime configuration on the selected technology.
RTP_MX_GENERIC_GET_INTERNAL_DATA_ DEVICE	260	Get the internal data for generic runtime configuration on the selected device.
RTP_MX_GENERIC_SET_INTERNAL_DATA_ DEVICE	261	Set the internal data for generic runtime configuration on the selected device.

Table B.1: Available MEX functions to control the RTProc system

Name	Description
RTP_MX_SYSTEM_UNINITIALIZED	The state in in uninitialized state, that is system is loaded but no action has been undertaken yet.
RTP_MX_SYSTEM_INITIALIZED	The system is in initialized state. From here the system components can be loaded.
RTP_MX_SYSTEM_COMPONENTS_SCANNED	The system components can be used from now on. The subsystems must be addressed from now on to prepare hardware and software before starting realtime processing
RTP_MX_SYSTEM_PREPARED	System is prepared for processing. That is the hardware and software components are ready for processing, depending on the operation mode.
RTP_MX_SYSTEM_PROCESSING	The system is in state processing.

Table B.2: Available states for type RTP_MX_SYSTEM

Name	Description
RTP_MX_HW_UNINITIALIZED	The hardware is in state uninitialized. No technology has been loaded.
RTP_MX_HW_INITIALIZED	Subsystem HW is initialized. The directories have been scanned, there may be technology components but there can also be none.
RTP_MX_HW_TECHNOLOGY_SELECTED	A technology has been selected. The selected technology can be activated in order to proceed to next state.
RTP_MX_HW_TECHNOLOGY_ACTIVE	Now the technology is active. In this state all devices can be listed.
RTP_MX_HW_DEVICE_SELECTED	One device has been selected. The

	selected device can be activated to proceed to next state.
RTP_MX_HW_DEVICE_ACTIVE	The device is now active. Hardware capabilities can be listed now and the device can be edited to specify a setup for realtime processing.
RTP_MX_HW_PREPARED	The hardware is prepared for processing. This state can only be reached in the context of a system call.
RTP_MX_HW_PROCESSING	The hardware is in processing state. This state can only be reached in the context of a system call.

Table B.3: Available states for type RTP_MX_HW

Name	Description
RTP_MX_SW_UNINITIALIZED	The software is in state uninitialized. No algorithm has been loaded.
RTP_MX_SW_INITIALIZED	Subsystem SW is initialized. The directories have been scanned, there may be algorithms components but there can also be none.
RTP_MX_SW_ALGORITHM_SELECTED	An algorithm has been selected. The selected algorithm can be activated in order to proceed to next state.
RTP_MX_SW_ALGORITHM_ACTIVE	Now the algorithm is active. In this state all devices can be listed.
RTP_MX_SW_PREPARED	The software is prepared for processing. This state can only be reached in the context of a system call.
RTP_MX_SW_PROCESSING	The software is in processing state. This state can only be reached in the context of a system call.

Table B.4: Available states for type RTP_MX_SW

Name	ID	Description
RTP_MX_SYSTEM_OPERATION_OFFLINE	0	For processing offline data. The `matlabHost` has a function ID to feed in signal vectors.
RTP_MX_SYSTEM_OPERATION_TALKTHROUGH	1	Talkthrough mode to bypass the algorithm.
RTP_MX_SYSTEM_OPERATION_DUPLEX	2	Full duplex operation.

Table B.5: Available operation modes for type RTP_MX_SYSTEM_OPERATION

Name	Description
RTP_ERROR_CODE_WRONG_COMMAND	The command sent to the MATLAB host did not have a valid ID.
RTP_ERROR_CODE_WRONG_TYPE	The parameter passed to the MATLAB host did not have the correct format. This is in general due to the second input argument.
RTP_ERROR_CODE_NUMBER_PARAMETER	The number of parameters is not correct. If passing only one argument for a function that expects two will result in an error.
RTP_ERROR_CODE_WRONG_STATE	The current function was called

	in the wrong state.
RTP_ERROR_CODE_PATH_NOT_EXISTS	The path specified does not exist.
RTP_ERROR_CODE_ID_OUT_OF_RANGE	The ID specified to access one element among others was out of range.
RTP_ERROR_UNEXPECTED_SITUATION	The error was due to an unexpected error.
RTP_ERROR_CODE_NOT_READY	The host was not ready for the requested operation.
RTP_ERROR_CODE_NOT_COMPATIBLE	The host is not compatible to current function call.
RTP_ERROR_CODE_INTERNAL	The MATLAB host has routed a call to a component. The component however failed to proceed.
RTP_ERROR_CODE_FATAL_EXCEPTION	A fatal exception has occurred. This is mostly due to reaching part of the code that should never be reached.

Table B.6: Available error codes for type RTP_ERROR_CODE

C. Source code of the warped FFT framework

See the internal GN ReSound version of this report for this appendix.

D. Source code of GN ReSound algorithms

See the internal GN ReSound version of this report for this appendix.

E. Steps for adding an algorithm to GN ReSound algorithms

See the internal GN ReSound version of this report for this appendix.

F. Source code of the Matlab GUI for the warped FFT framework

See the internal GN ReSound version of this report for this appendix.

G. Steps for updating the Matlab GUI for the warped FFT framework

See the internal GN ReSound version of this report for this appendix.

H. Source code of the database creator

See the internal GN ReSound version of this report for this appendix.

I. Source code of the input fragments player

See the internal GN ReSound version of this report for this appendix.

J. Detailed results of the evaluation of PNR

Noise type	Fragment	Gain depth		
		-3	-6	-9
		QR		
typing	audio_input_2005_6_7_11_11_18.937	3	3	3
motorbike	audio_input_2005_6_7_11_12_21.468	2	3	3
shower	audio_input_2005_6_7_11_13_41.421	2	2	2
shower	audio_input_2005_6_7_11_13_58.953	2	2	2
saw	audio_input_2005_6_7_11_14_46.531	3	3	3
saw	audio_input_2005_6_7_11_15_7.765	2	2	2
hammer	audio_input_2005_6_7_11_15_27.375	1	1	1
babble	audio_input_2005_6_7_11_16_38.906	2	3	2
babble	audio_input_2005_6_7_11_16_59.406	2	2	1
babble	audio_input_2005_6_7_11_17_40.671	3	3	2
female speaker	audio_input_2005_6_7_11_18_38.671	2	2	2
child speaker	audio_input_2005_6_7_11_19_3.734	3	3	3
male speaker	audio_input_2005_6_7_11_19_40.796	3	3	3
classical music	audio_input_2005_6_7_11_21_20.89	3	3	4
pop music	audio_input_2005_6_7_11_23_27.093	2	2	3
female singer	audio_input_2005_6_7_11_23_46.859	3	3	3
pop music	audio_input_2005_6_7_11_24_32.796	3	3	4
church bell	audio_input_2005_6_7_11_25_52.859	2	2	2
airplane	audio_input_2005_6_7_11_26_49.921	1	2	2
phone	audio_input_2005_6_7_11_27_13.89	2	2	2
music hall	audio_input_2005_6_7_11_28_33.906	2	2	3
factory	audio_input_2005_6_7_11_30_54.234	2	3	2
speech shaped	audio_input_2005_6_7_11_31_20.812	1	2	2
female speaker	audio_input_2005_6_7_11_33_36.109	3	3	3
male speaker	audio_input_2005_6_7_11_34_10.39	2	2	2
pop music	audio_input_2005_6_7_11_34_38.968	2	2	2
classical music	audio_input_2005_6_7_11_35_51	3	3	3
typing	audio_input_2005_6_7_13_36_28.937	3	3	3
motorbike	audio_input_2005_6_7_13_37_38.75	2	2	3
shower	audio_input_2005_6_7_13_38_45.921	2	2	2
shower	audio_input_2005_6_7_13_39_10.171	1	1	1
saw	audio_input_2005_6_7_13_39_43.156	3	3	3
hammer	audio_input_2005_6_7_13_40_28.703	1	1	1
babble	audio_input_2005_6_7_13_42_1.281	1	2	1
female speaker	audio_input_2005_6_7_13_43_44.25	2	2	2
child speaker	audio_input_2005_6_7_13_44_23.125	3	3	3
male speaker	audio_input_2005_6_7_13_44_45.828	2	2	2
female singer	audio_input_2005_6_7_13_48_52.078	3	3	3

pop music	audio_input_2005_6_7_13_49_20.859	3	3	3	
pop music	audio_input_2005_6_7_13_49_59.578	2	2	3	
church bell	audio_input_2005_6_7_13_50_56.953	2	2	3	
airplane	audio_input_2005_6_7_13_51_54.281	1	2	2	
phone	audio_input_2005_6_7_13_52_18.578	2	2	2	
music hall	audio_input_2005_6_7_13_53_35.578	3	4	4	
factory	audio_input_2005_6_7_13_55_57.812	2	3	2	
speech shaped	audio_input_2005_6_7_13_56_21.468	1	1	2	
female speaker	audio_input_2005_6_7_13_58_53.625	2	2	2	
male speaker	audio_input_2005_6_7_13_59_16.078	2	2	2	
pop music	audio_input_2005_6_7_13_59_46.437	2	2	2	
classical music	audio_input_2005_6_7_14_1_15.187	3	3	4	
pop music	audio_input_2005_6_7_14_1_58.093	3	3	4	
typing	audio_input_2005_6_7_14_32_29.75	2	2	2	
factory	audio_input_2005_6_7_14_34_4.062	2	2	3	
shower	audio_input_2005_6_7_14_35_1.421	2	2	3	
saw	audio_input_2005_6_7_14_36_7.828	2	2	2	
hammer	audio_input_2005_6_7_14_36_35.234	2	2	2	
babble	audio_input_2005_6_7_14_38_2.296	1	2	1	
female speaker	audio_input_2005_6_7_14_39_49.062	2	2	2	
child speaker	audio_input_2005_6_7_14_40_25.296	2	2	2	
male speaker	audio_input_2005_6_7_14_40_50.203	2	2	2	
female singer	audio_input_2005_6_7_14_44_56.39	3	3	3	
pop music	audio_input_2005_6_7_14_45_36.171	3	3	3	
pop music	audio_input_2005_6_7_14_46_3.484	2	3	3	
airplane	audio_input_2005_6_7_14_47_55.281	1	1	2	
phone	audio_input_2005_6_7_14_48_21.171	2	2	2	
factory	audio_input_2005_6_7_14_52_3.125	2	3	2	
speech shaped	audio_input_2005_6_7_14_52_40.781	1	2	2	
female speaker	audio_input_2005_6_7_14_54_40.968	3	3	3	
female speaker	audio_input_2005_6_7_14_55_6.593	2	2	2	
male speaker	audio_input_2005_6_7_14_55_23.468	2	2	2	
pop music	audio_input_2005_6_7_14_55_38.718	2	2	2	
		2.14	2.32	2.39	**Average**

Table J.1: Detailed results of the evaluation of PNR

K. Suggestions for portable use of the RTProc system

RTProc can be made suitable for walking around with it on a laptop. This means that the GUI that can operate the RTProc system has to be reprogrammed for a PDA. Several programming languages can be used for this purpose, for example html or java. The PDA has to be able to communicate with Matlab on the laptop. This can be done with a serial connection, an USB connection or a wireless connection, for example bluetooth or WIFI.

A serial connection is the simplest solution, because Matlab has a serial port interface. This interface provides direct access to peripheral devices that are connected to the computer's serial port. The interface is established through a serial port object. The serial port object supports functions and properties that allow the user to configure serial port communications, use serial port control pins, write and read data, use events and callbacks and record information to disk. A Matlab program that communicates with a serial device via the serial port object uses this five-step pattern:

1. Create the serial port object with the function *serial*.
2. Set up the device and communication channel properties with the functions *set* and *get*.
3. Connect to the serial port with the function *fopen*.
4. Send and receive data and commands with the functions *fread*, *fwrite*, etc.
5. Disconnect from the device and free resources with the functions *fclose* and *delete*.

The device on the serial port works like a special kind of file, as a result of which file I/O functions can be used to send commands to and read data from the serial device. Serial devices communicate with each other via protocols or sequences of commands and responses. These protocols are either synchronous or asynchronous. The serial port object supports both types of protocols.

An USB connection or a wireless connection is a more complex solution, because Matlab does not have an interface for these types of communication. There are two possibilities to interact with a device via an USB or wireless connection. The first possibility is to write a MEX file in C or Fortran code that can communicate with a generic USB driver or a generic wireless driver, for example a generic bluetooth driver or a generic WIFI driver. With a MEX file, C or Fortran subroutines can be called from Matlab as if they are built-in functions. The second possibility is to code a DLL with a C interface that can communicate with a generic USB driver or a generic wireless driver, for example a generic bluetooth driver or a generic WIFI driver. Matlab's interface to generic DLLs can be used then to interact with the functions in the DLL.

The portable use scenario is then as follows. The user walks around with a PDA in his hands that runs the GUI that can operate the RTProc system that uses the warped FFT framework as algorithm component and GN ReSound algorithms as audio processing node. In addition, the user has a laptop in his backpack that runs the warped FFT framework for RTProc. The user controls the RTProc system with the GUI on the PDA. This GUI performs all calculations and sends the MEX functions for the RTProc system that are generated to Matlab on the laptop. Matlab registers these MEX functions and sends them through to the RTProc system. It then sends the results of the MEX functions from the RTProc system back to the PDA. The GUI on the PDA registers these results and uses them to update the GUI or to perform new calculations.