WiLMA
a *Wi*reless *L*arge-scale *M*icrophone *A*rray

Christian SCHÖRKHUBER,
Markus ZAUNSCHIRM and
IOhannes m zmölnig
Institute of Electronic Music and Acoustics (IEM)
University of Music and Performing Arts, Graz, Austria
{schoerkhuber, zaunschirm, zmoelnig}@iem.at
http://wilma.iem.at

Abstract

Everyday situations are rich in numerous acoustic events emerging from different origins. Such acoustic scenes may comprise discussions of our fellow human beings, chirping birds, cars, cyclists, and many more. So far, no recording or scene analysis technique for this rich and dynamically changing acoustic environment exists, though it would be needed in order to document or actively shape an acoustic scene. We know customised techniques for recording symphony orchestras with a static cast, but none that automatically readjusts to scenes with varying content. Thus, a new recording technique that analyses the signal content, the position and the activity of all sources in a scene, is required. We present *WiLMA*, a wireless large scale microphone array, a mobile infrastructure that allows for investigating into new recording and analysis techniques.

Keywords

network audio, sensor network, microphone, distributed processing

1 Introduction

Traditionally, the sensor nodes of a wireless sensor network (WSN) that captures sound events, are populated with low quality microphones, amplifiers and analogue to digital converters (ADCs) in order to decrease sensor node size, power consumption and cost.

The Wireless large-scale microphone array (WiLMA) introduces high quality audio processing in wireless sensor networks. Each of the sixteen sensor modules (SM) allows for capturing of up to four high-end microphone signals which in turn enables the use of a 4-channel microphone array (e.g. first order tetrahedral ambisonics microphone) per SM. Thus, the system operates as a large scale microphone array, with a total of 64 audio channels. A single SM and the used microphone array are depicted in fig.1.

The acquired data from all SMs is transmitted (either wireless or wired) to a central unit

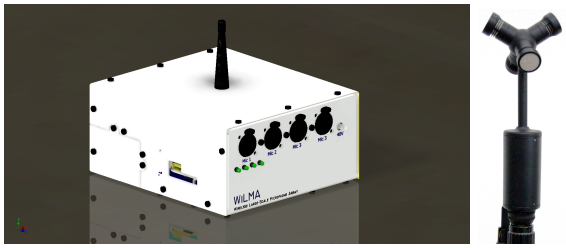

Figure 1: Sensor module and microphone array (*Oktava 4D-ambient*)

(CU) running the host application shown in fig.6 and fig.7. This host application visualises input levels, synchronisation and battery status. Further, it allows the user to individually configure each SM for a specific task.

Each SM is equipped with a local processing unit in order to perform computations on the acquired data. Instead of sending the raw data to the the central unit responsible for the fusion, sensor modules can use their processing abilities to locally carry out simple computations and transmit only the required and partially processed data. This intelligent sensor network approach results in decreased network traffic and higher flexibility of the system.

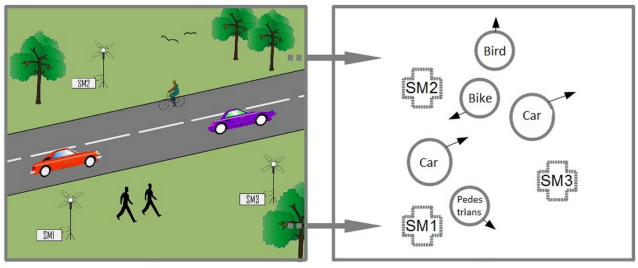

Figure 2: Acoustic scene analysis

An example application using the WiLMA hardware is to separate sources of an acoustic scene and track their movement. Thus, it should be possible to analyse the separated source signals and to assign a specific event to

a specific source. Fig.2 conceptually depicts the process of a spatial transcription. Areas that could benefit from that application include assisted living scenarios, acoustical planning, the surveillance of urban areas, multichannel source separation, event detection, source tracking and so on. Another application is the high audio quality multichannel recording of an acoustic scene with the added benefit of flexible microphone positioning due to wireless operation of the system.

2 Design

The basic design of the sensor network contains a *Central Unit* and a variable number of *Sensor Modules*.

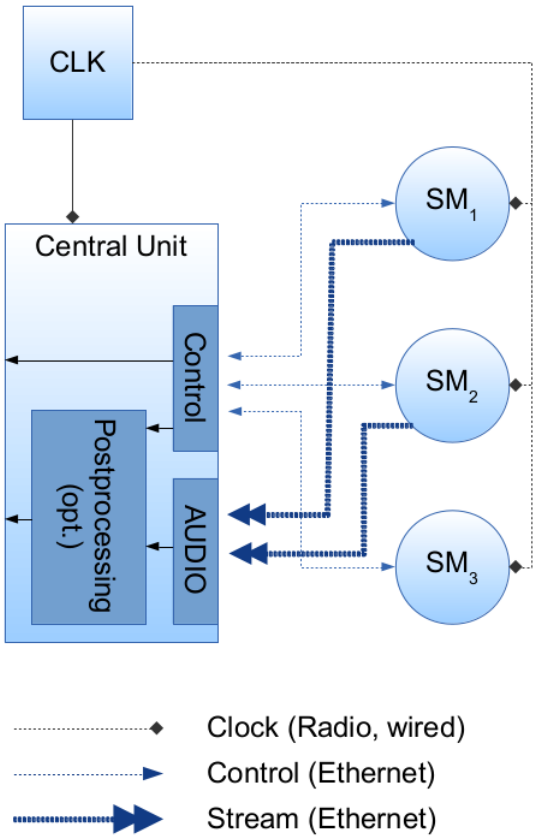

Figure 3: Network diagram of multiple synced Sensor Modules and a Central Unit

The Central Unit controls and monitors the individual modules, The Sensor Modules capture audio autonomously and send their data to the Central Unit, where it can be collected for further processing.

To allow for sample synchronous audio capturing, all SMs are connected to a central master clock.

2.1 Modes of Operation

We can distinguish between three different modes of operation for each sensor unit:

2.1.1 Recording

The simplest operational mode is to *record* the microphone signals locally on the SM. The recording should be time-stamped, so the recording of multiple SMs can be time-aligned later in an offline process.

2.1.2 Streaming

For recording and monitoring purposes, it might often be desirable to not collect the audio data decentralised on the SMs and collect them later, but rather have all audio channels available immediately at the Central Unit, by means of real-time streaming. This allows the sensor network to be used as a de-centralised capture-only multichannel sound card.

2.1.3 Processing

Each SM is also equipped with a local processing unit that can be used to do (simple) analysis of the local signals, parallelising the computational load.

The actual processing algorithm might change depending on the application. It is therefore required to be able to implement algorithms in a reasonable environment and deploy these programs easily on all (or selected) SMs.

The result could be either an enhanced signal, meta-data about the signal or a mixture of both (e.g. using signal identification on the 4 channel recording, it is possible to only stream a mono-version of the signal together with positional meta data).

2.1.4 Mixed

Multiple connected SMs need not operate in the same mode. For instance, some SMs could be streaming audio, whereas other SMs would only do processing and send meta-data to the Central Unit (as depicted in Fig.3).

2.2 Communication

All control communication between the CU and the SMs is based on a bi-directional *OSC*-connection. Typical OSC-applications use UDP as transport protocol, which behaves badly in congested networks. In order to work around reliability issues, the transport layer can be configured to either use UDP or TCP/IP with

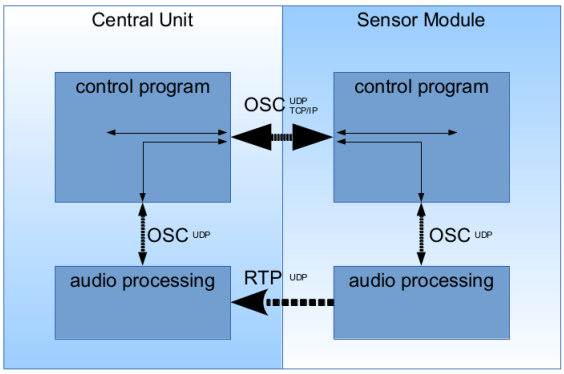

Figure 4: Communication between central unit and sensor modules

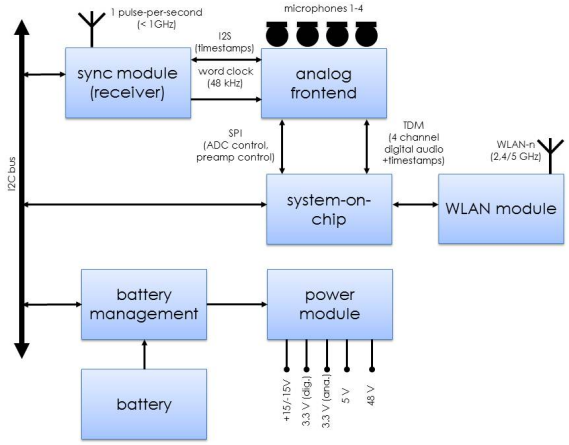

Figure 5: Block diagram of the sensor module

SLIP-based packetizing as suggested by the OSC-1.1 specifications [1].

Besides configuring and activating the various modes of operation, the "control channel" includes basic infrastructure (like sending and receiving heartbeats in order to determine whether the connection is still established (in the case of UDP) and the SM is still responsive) and health information (e.g. CPU load, memory and disk usage, battery status, sync status, microphone levels). It also allows to configure the SM (e.g. setting the gain of the microphone preamplifier) and transports the entire meta-information extracted by any optional *process*ing on the SM.

Audio streaming from the SM towards the CU is not done via OSC (as suggested e.g. by [2]), but instead uses the more widespread *RTP* protocol [3] on top of UDP. The RTP-timestamps are synchronised, in order to be able to re-align the audio signals of multiple SMs.

3 Sensor Module

The Sensor Module (see Fig.1) consists of a custom hardware design running Linux.

3.1 Audio

The 4 channel analogue front end is equipped with THAT1570 low noise, differential microphone preamplifiers which are digitally controlled via SPI using THAT5173 controller ICs. Analogue to digital conversion is performed by an AD1974, a 4 channel, 24 bit ADC with integrated phase-locked loop (PLL).

3.2 Synchronisation

The internal sampling clock of the AD1974 is derived from the word clock provided by the synchronisation module. Wireless synchronisation within the WiLMA system is established via a 1 pulse-per-second timestamp signal that is broadcasted by the master module on a sub-GHz ISM band. The synchronisation module is populated with a voltage controlled oscillator (VCXO) that is disciplined by a frequency locked loop (FLL) and a subsequent frequency divider to obtain the 48 kHz word clock for the ADC. The sample accurate timestamps generated by the synchronisation module is multiplexed with the output data of the ADC into a 8-channel/32 bit time-division multiplexing (TDM) stream.

3.3 System On Chip

The heart of each sensor module is a *Beaglebone A6* equipped with an *ARM Cortex A8* based processor running Linux. The TDM audio stream is read by an ALSA driver that sets up the ADC, controls the microphone preamplifiers and accesses the Multichannel Audio Serial Port (McASP) via the *DaVinci ASoC* driver.

3.4 Power Supply

The power module generates supply voltages for the different modules from the wall plug supply or the battery, respectively. It also generates an optional 48V supply voltage for microphones requiring phantom power. The *LiPo* battery pack is connected to a battery management system which is responsible for controlling charge voltage and charge current, switching between power sources and providing information about the battery status via I2C bus. In case of battery undervoltage the battery management system autonomously disconnects the load from

the battery to keep the battery in a safe state.

3.5 Software

Each SM is running on *Ubuntu-11.10 (Oneiric Ocelot)*, using the standard *armel* architecture packages, with the notable exception of the kernel, which is a customised build of *linux-3.2.30* due to the required ALSA drivers of the custom sound card.

When the system starts up, a control program – the *WILMAsm* daemon – is started. This daemon monitors the various health states of the system and runs an OSC-server for communication with the CU. The service is announced via ZeroConf/Avahi [4], using the type specifier `_wilma-sm._udp` (resp. `_wilma-sm._tcp`).

Since the daemon is implemented in *Python*, a more appropriate sub-system for running the audio-related tasks is needed. This subsystem has been implemented using *Pure Data*, as it is a well known environment and allows to deploy algorithm implementations in a text-based form (thus reducing the need to cross-compile binaries for the target *ARM* platform).

In order to integrate nicely with the framework, any *process*ing unit needs to adhere a simple standard, which defines inlets/outlets of the *Pd*-patch and the filesystem layout.

The used implementation of *Pd* is a slightly modified version of `Pd-0.44-2`. The main modification has been a customisation towards the special audio layout of the SM, which provides an eight channel audio interface, where only the first four channels contain actual audio data (as sampled from the microphones), and the remaining four channels contain a 32bit timestamp synchronised on all SMs.[1]

Pd is running as a sub-process of the control-daemon, which monitors the audio process and restarts it in the unlikely event of a crash. The control daemon and the audio process communicate via a bi-directional OSC connection on top of UDP. (No TCP/IP option is given here, as the connection is only running on `localhost`).

4 Central Unit

The Central Unit is an off-the-shelf Linux system eventually equipped with a MADI audio interface (in order to play back the independent audio streams from 16 SMs), and is running the audio stream aggregator and control application *WILMix*.

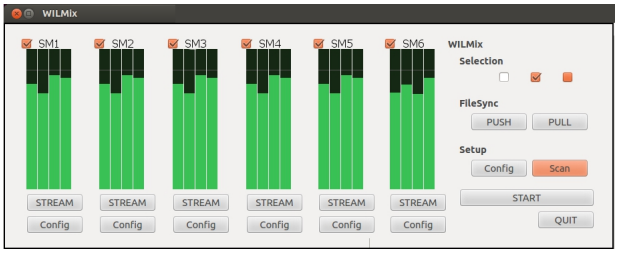

Figure 6: *WILMix* overview over available SMs

The control application provides a user-interface for controlling and monitoring the various aspects of the SMs, like starting audio streaming, distributing *process*-patches or collecting recordings.

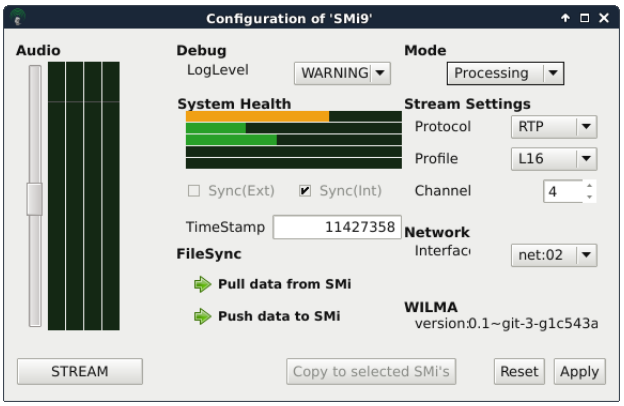

Figure 7: *WILMix* controlling a specific SM

The application uses ZeroConf to detect all available SMs in the local network, and constructs a mixer application for the given number of channels.

The audio stream aggregator receives the RTP-streams from the various SMs, and realigns them in time, so that they can be played back sample synchronously.

As is with the SMs, the control part of the application is implemented in *Python*, whereas the audio processing part is written in *Pd*, both communicating via OSC over UDP.

5 Discussion

While the current software implementation works as a proof of concept, there are certainly things to improve.

For one thing, the use of *Pure Data* on an *ARM Cortex A8* is suboptimal, as the processor

[1] Obviously this makes the timestamp encoded in a highly redundant way. The main reason for this redundancy is that the AD1974 allows to easily copy a single 32bit auxiliary digital data word into four channels at once. Since the channels 5 to 8 are unused anyhow, no immediate drawback arises from this redundant data handling.

lacks an FPU, whereas *Pd* does all processing on floating point samples.

Implementations using alternative frameworks that would allow for fix-point arithmetic (such as *GStreamer*[5]) were initially planned but were soon discarded in order to avoid cross-compilation environments altogether. (A major issue when the potential algorithm implementers are matlab-spoilt, C-agnostic students).

Even with *Pd* as the audio engine, it might be advisable to use it's library incarnation *libpd*[6] rather than a full-fledged *Pd*, as it would greatly simply the communication between the control application and the audio engine. Using *libpd*, it should even be possible to get rid of the modifications currently needed to obtain the 32bit timestamps from the audio channels[2].

6 Availability

The source code for the WiLMA-Application (running on both the SMs and the CU) has been released under the *GNU GPL*, and is available for download from *github*[3].

The hardware has beed designed in-house at the *Institute of Electronic Music and Acoustics*. However, the schematics have not yet been published under an open license.

7 Conclusions

The WiLMA hardware introduces high quality audio processing in wireless sensor networks. The overall system comprises 16 sensor modules that allow for recording up to 64 audio channels. Audio signals in the frequency range between 20Hz and 20kHz are converted with a high quality ADC (24bit). The information of each sensing module is collected by a central unit, that combines the individual data to a final outcome. Data transmission between the SMs and a central unit can either be wireless (WLAN) or wired (Ethernet). The capsules of the used microphone arrays (*Oktava 4D*) obey a linear frequency response (no sound colouration) and a minimal gain mismatch between capsules. Furthermore, the system offers a runtime of up to 8 hours in battery-powered mode. Thus, its mobile and flexible use is ensured.

In order to allow for the application of algorithms of the acoustic field theory, the audio streams of different SMs are synchronised with an accuracy of one sample ($\approx 20\mu$s).

[2]The timestamps cannot be read directly in patchspace, as *Pd* does not provide a 32bit integer type – all numbers are equal... and they are (single precision!) floats.

[3]`https://github.com/iem-projects/WILMAmix/`

8 Acknowledgements

This project was partly funded by the *MINT/Masse* program of the Austrian Federal Ministry of Science and Research.

References

[1] A. Freed and A. Schmeder, "Features and future of open sound control version 1.1 for nime," in *NIME'09: Proceedings of the 9th Conference on New Interfaces for Musical Expression*, 2009.

[2] W. Jäger, "Audio over internet using OSC," Institute of Electronic Music and Acoustics, Tech. Rep., 2010.

[3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," *IETF RFC3550*, 2003.

[4] S. Cheshire, "Zero configuration networking (Zeroconf)," in *http://www.zeroconf.org/, accessed 2014-02-02.*

[5] GStreamer Team, "GStreamer: open source multimedia framework," in *http://gstreamer.freedesktop.org/, accessed 2014-02-02.*

[6] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth, and H.-C. Steiner, "Embedding pure data with libpd," in *Proceedings of the Pure Data Convention*, 2011.