International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 5, No.1 (Jan-2016)

http://dx.doi.org/10.12785/ijcds/050102

Non-native Mobile Porting and Multi-platform Benchmarking of Blind Source Separation Algorithms

Marvin Chibuzo Offiah and Markus Borschbach

Competence Center "Optimized Systems", University of Applied Sciences (FHDW), Bergisch Gladbach, Germany

Received 12 May 2015, Revised 20 July 2015, Accepted 2 Oct. 2015, Published 1 Jan. 2016

Abstract: The human daily and the professional life demand a high amount of communication ability, but every fourth adult above 50 is hearing-impaired, a fraction that steadily increases in an aging society. For an autonomous, self-confident and long productive life, a good speech understanding in everyday life situations is necessary to reduce the listening effort. For this purpose, an app-based assistance system is required that makes every day acoustic scenarios more transparent by the opportunity of an interactive focusing on the preferred sound source. The key component of this assistance system is the blind source separation algorithm. Developing such an app in the context of a short-term research project with limited time and limited human time to realize this goal statement raises a lot of challenges. One of the key challenges is the porting of PC-based source separation algorithms to a mobile device without the need for native implementation, and integrating these ported algorithms into the mobile graphical user interface (GUI) app. At the same time, it raises the question about the size of the penalty paid in terms of loss in runtime performance due to such porting. This paper explains the realized porting and integration method and provides a runtime performance benchmark that compares the PC-based algorithms to the ported algorithms in different computing environments. It then draws a conclusion about the practicability of the porting method proposed.

Keywords: signal processing, blind source separation, REPET, PARAFAC, real-time, mobile device, smartphone, Google Nexus, mobile app, graphical user interface, porting, computer architecture, ARM, compiler, programming languages, MATLAB, GNU Octave, operating systems, emulation, Linux, Ubuntu, benchmarking, runtime performance

1. INTRODUCTION

The coping of the everyday working life demands a high level of the acoustic perception caused by a constant sensory overload: Communication plays an important role in the performance-orientated working life in terms of conferences, working in an open-plan office and meetings. A permanent listening effort reduces the working productivity. Furthermore, the listening ability is reduced with aging. The aging society and the longer working life results in an increasing number of hearing-impaired people, which also have difficulties in coping with their daily life. This leads to the consequences of social isolation. Despite using a digital hearing aid, the ability of the selective listening experience is lost and cannot be re-learned. [1] These facts require the development of an assistance system that is able to record acoustic scenarios, to separate the sources and provide an opportunity to listen to the preferred sound source. Such an app-based assistance system as an artificial simulation has been presented as the goal-statement of the SMART-NAVI research project. The key point of this app is the Blind Source Separation (BSS) algorithm. In the ideal case, the BSS algorithm executes on the mobile platform in real-time (online). However, a non-real-time (offline) processing also benefits the user to a large extend.

This requires that a state-of-the-art algorithm that is suited for real-world convolutive audio data and that was originally written for a PC platform is ported to a mobile device and integrated into a running graphical user interface (GUI) app. Due to limited budget and time resources for the research project, this needs to be done without native reimplementation of the algorithm for the mobile platform. A time-saving porting method is required. Furthermore, there is a need to assess the actual online capabilities on the target platform, which creates the need for appropriate experiments.

E-mail address: off@fhdw.de, mbo@fhdw.de

This paper first provides an overview of the state of the art: It introduces the BSS algorithms in question and explains the existing gap between the related PC environments for those algorithms on the one hand and mobile processors running a GUI app on the other, in order to derive the challenges faced. It then explains a solution to the challenges of porting BSS algorithms to the mobile platform and integrating them into a GUI app without the need for manual native reimplementation of the source code. It then provides a runtime performance benchmark of the two algorithms in their offline and online versions running on the PC and on the mobile device in different environments. This is intended to demonstrate the real-time capabilities and runtime losses or gains of the chosen porting and/or integration method. A conclusion is given at the end.

2. STATE OF THE ART

A. *The BSS Algorithms REPET and PARAFAC*

BSS has been of immense interest in the area of digital signal processing during the past decades. Various methods for separating mixed audio signals have been proposed, but the development is still in progress due to the complexity of the problem. In many situations, there is a need to recover original signals from the observed mixture of signals, for example, to recover a single voice in a noisy environment. Regarding the development of the assistance system, an important topic of our research project is the separation of sound signals using BSS. The goal of the separation process is to recover the best estimate (estimated sources) of the original signals (sources) from mixed observations (mixtures) recorded by microphones (sensors) without – or with only a little – information about the mixed signals or the mixing process. A typical example of BSS is the Cocktail-Party Problem [2]. For example, at a party, the person is surrounded by a variety of different sounds like human voices, music or other background noises. If the sounds are recorded with microphones at different positions in the room, every microphone captures a mixture of the sound signals with different weighting. BSS algorithms allow the separation of a sound mixture into its single sources, which enables the user to listen to the preferred single sound source. As a consequence, the selection of a BSS algorithm as a part of a smartphone application for the hearing impaired is in the focus. The BSS enables recovering or separation of unknown signals (sources) from observed mixtures through unknown propagation channels. It is not necessarily the goal to recover the original sources from the mixture, but to recover model sources without disturbing interferences from other sources. [3][4]

A lot of BSS algorithms exist and are explained in detail in literature [3][4][5][6]. They have been experimented with in the course of the project to assess the quality performance of these algorithms with different setups. These comparative experiments have come to the result, that the REPET (Repeating Pattern Extraction Technique) [5] and PARAFAC (PARallel FACtor analysis) [6] algorithms perform best in terms of objective and subjective quality metrics. They are therefore to be used for the assistance system. For REPET, the versions of the algorithm implemented in the MATLAB scripts “repet_sim.m” (offline) and “repet_sim_online.m” (online) [7] by Zafar Rafii and Bryan Pardo are used; for PARAFAC, Dimitri Nion’s MATLAB script “sobi_conv_cp3.m” [8]. The algorithms are referred to here in this paper as REPET offline, REPET online and PARAFAC offline. In current work, only the separation of two audio sources from a two-channel microphone input is experimented with. But in future experiments, we intend to consider further cases as well, using more and external microphones.

B. *The Bridge between the MATLAB Environment and ARM Processors and GUI Apps*

The most important challenge in realizing the mobile assistance system is to port an existing online BSS algorithm to the mobile device: Mobile devices typically use a Reduced Instruction Set (RISC) CPU design (mostly ARM [9]) optimized for the mobile context, as opposed to the Complex Instruction Set (CISC) design typically found in PCs (mostly Intel and AMD [10][11]) [12]. ARM processors are the most widely used processors on mobile devices [13]. For these reasons, a source code program written and compiled for a PC cannot necessarily be compiled and executed on a mobile device without major and time-consuming adjustments to it and recompiling the source code itself. This holds for all languages which are not compiled to a virtual machine like Java.

In the context of computer science research in signal processing, source code is often written, exchanged and re-used in a 4th-generation and domain-specific scripting language such as Matlab. The corresponding MATLAB environment by MathWorks is proprietary commercial software and was initially released in 1984. It therefore can look back to a 30-year period of user experience, development and improvement. Ingle and Proakis describe MATLAB as “an interactive, matrix-based system for scientific and engineering numeric computation and visualization”. It is particularly efficient for rapid development, since complex numerical problems can be solved with just a few commands, where other, 3rd-generation, languages such as C would require much more effort. MATLAB is available for all major PC platforms and uses a command line for typing or reading instructions that are then executed immediately. [14][15][16] As many scientific publications in signal processing such as [4][5][6] and [17], and the personal experience of this paper’s authors hint at, Matlab stands out as the most likely language in which to find templates for a given task. Matlab’s important role as a key player in the market of numerical analysis software languages is further substantiated by the TIOBE index as of June

2015, when ignoring all – in this case irrelevant – general-purpose and non-mathematical programming languages listed.

Matlab's source code is executed not directly by the underlying OS, but by the appropriate runtime environment software (i.e. the MATLAB environment). The MATLAB environment exists only for PCs, no equivalent ARM version of MATLAB exists to this day (currently MATLAB 2015a). Without any workaround to this problem, this means that the REPET and PARAFAC scripts have to be completely reimplemented in a native programming language for the smartphone, and using native libraries. This is a very time-consuming task and then needs to be repeated every time the BSS algorithm or the system architecture changes. A workaround, i.e. an efficient way of smartphone porting is necessary, and it needs to be realized and tested in runtime performance, which is the purpose of this paper. Runtime performance tests are critical in particular for the online algorithms like REPET online, since the real-time capability needs to be demonstrably maintained.

Our target operating system platform of choice is an Android 5 due to a higher degree of flexibility in open systems and its widespread use [18]. Android is an open source mobile operating system with a multi-touch GUI by Google based on the Linux kernel. As a mobile operating system, it also runs on an ARM processor. [19] It currently dominates the market of mobile operating systems for smartphones by almost 80%, as evidenced by Gartner [20]. GUI apps in Android are run by the Android Runtime Environment (see figure 1), which consists of a virtual machine (VM) called Dalvik VM and certain core libraries. Programs for the Dalvik VM are written in Java and the compiled Java bytecode is translated to the memory and processor-speed optimized Dalvik bytecode (.dex or .odex files). [19][21] All apps are executed by the Dalvik VM. There is a common basic application framework used by all Android apps (see figure 1): Most app components are objects of custom classes inheriting from – or using – the Activity, Intent, Service or ContentProvider classes, among others. Each of these classes has a particular role to fulfill: E.g., an Activity represents a graphical screen, i.e. a "running app" from the user's perspective and the entry point of the app. An Intent stands for "an intent to do something", for example starting a second Activity, and it is also able to store messages or other kinds of data created by the first Activity in the form of key-value pairs. The second Activity extracts these values from the Intent that started it. Through this and other mechanisms within the application framework, communication between multiple launched and running Android apps is possible. [22][23]

In the assistance system, the GUI app on the Android 5 device performs the recording, playback and graphical display, while the algorithm ported to the device performs the BSS. These two components do not necessarily have to be realized as a single Android process. It depends on the porting method chosen. So besides the porting of the BSS algorithm, integrating it into the GUI app so that both can exchange data and instructions between them, is potentially an additional challenge of its own.

3. METHODS

A. *Excluded Approaches*

There is a number of approaches that look like a solution at first sight, but prove non-feasible on closer inspection:

As already mentioned, there is no MATLAB for ARM CPUs. The MATLAB Mobile app for Android only works as a client to connect from the Android device to an existing MATLAB session on a Desktop Server or on the MathWorks Cloud, but not as a stand-alone smartphone-only solution [24].

The transcoding tool MATLAB Coder only supports a subset of the toolboxes needed and still requires a lot of Matlab code editing (only fixed-size variable and parameter inputs supported) before auto-transcoding to native C/C++ code [25][26][27]. A combination with the Android NDK to compile and integrate the transcoded C/C++ libraries into the Java GUI app [28] is therefore still impractical.

Another tool, the MATLAB Compiler, – although also capable of creating Java (i.e. platform-independent) executables – still requires the PC-only MATLAB Compiler Runtime libraries for the executable to function [29][30].

There has also been an inspection of Open Source alternatives to MATLAB. GNU Octave, together with its toolbox clones from the Octave Forge project, [31], is an Open Source clone of MATLAB that is mostly compatible with MATLAB [32][33] and also exists as an Android app [34]. But some experiments show that running the BSS scripts on the Octave app leads to unpredictable errors (the Android command line freezes without feedback, hence a bug in the app's implementation is to be deduced from that).

B. *Realized Approach*

1) *GNU Octave on a Linux Emulation for Android*

The realized approach is an emulation [35] of a Linux OS for Android ARM devices that runs in parallel with the native GUI app and executes a GNU Octave on that OS (see figure 1). Emulators for other Desktop OS's are either still a work in progress (Wine for Windows [36]) or simply non-existing (Mac OSX) to this date. There is an app called "Complete Linux Installer" that guides the user through the steps of installing ARM Linux distributions on top of the existing Android OS as an emulation and performs the booting of such an emulated OS (root access to the phone is required). In this case, the Ubuntu Linux distribution is used. After installing

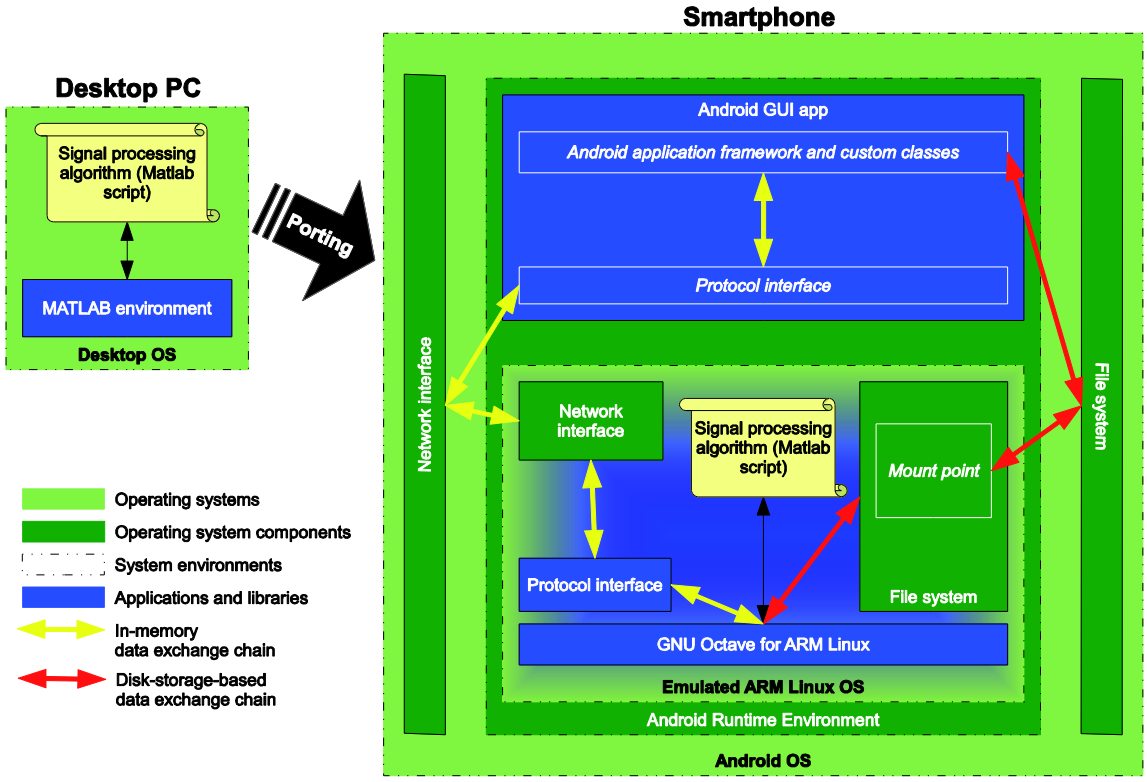


Figure 1. Realized porting method

Ubuntu this way, the emulation is launched and a GNU Octave version packaged for this Linux distribution is installed via Linux command line. The same emulated Linux command line is then used for launching the GNU Octave for ARM Ubuntu. The Matlab BSS scripts are copied to the Android phone's file system. The emulated Linux has its own file system, but is also able to mount parts of the hosting Android Linux's file system onto dedicated mount points of the emulated Linux' file system (see figure 1). Hence the GNU Octave on the Ubuntu is also able to access the Matlab scripts copied to the Android's file system. [37]

2) Data Exchange

Once ported this way, the other challenge, also discussed in [2] is the successful integration of the ported algorithm (Octave-compatible MATLAB script) into the running GUI app: Inter-process communication between the app running natively on the OS and the Octave process running on the emulated Ubuntu on the same device is necessary so that unprocessed and processed audio data frames and instructions can be exchanged between the two systems. In our solution, this data exchange is realized through data files in the commonly accessible file system (see "disk-storage-based data exchange chain" in figure 1): Certain files are polled by both the GUI app and the Octave process and used as control instructions, other files contain the objects of data exchange. So all inter- process communication works through the file system, meaning through secondary storage.

A direct in-memory solution for communication actually requires bridging the gap between the two OS's (i.e. between the hosting Android and the emulated Ubuntu). This gap is bridged most easily through a channel that uses the network interfaces: Note that the emulated Linux – just like any standalone OS – provides a network interface, and that it is mapped to the hosting Android system's network interface. See "in-memory data exchange chain" in figure 1. So what is required is using and/or implementing an appropriate network protocol on each side, i.e. between the Android OS using one port and the emulated Linux OS using another port. Our experiments, however, demonstrate that the data exchange time through secondary storage is still very marginal and thus acceptable. Communication through secondary storage is therefore sufficient for this application.

4. MULTI-PLATFORM BENCHMARK

A. Test Environments

What is tested is the runtime performance of each algorithm version on a particular platform. Testing is performed both on an Intel Windows 7 Desktop PC (Intel Ceon X5650 CPU of 2.66 Ghz and 12.0 GB of RAM) and on a Google Nexus 5 phone (Quad-core 2.3 GHz Krait 400 CPU, 2 GB RAM). These are the two base platforms or devices. Each of these base platforms is subdivided into more specific ones, when including the execution environments (MATLAB, Octave, GUI app): On the PC platform, all algorithms are executed in MATLAB and in Octave. On the Nexus phone, all algorithms are basically executed on the Octave that runs on the emulated Ubuntu system. But for REPET online, in one case the test is run with the GUI app running in parallel, and in the other case without the GUI app running. This is done to better analyse existing runtime volatilities during execution: Tracing big differences in execution times either to the porting architecture or to multiprocessing with the GUI app.

Each algorithm is tested for 8000 Hz and 16000 Hz data to represent the minimum audio quality still tolerable for basic speech intelligibility (as in telephone conferences) and a common frequency slightly above that [38], in order to show where the real-time threshold (processing time must be below the length of the data frame processed) is broken.

So each test case is defined by a particular combination of the algorithm version (REPET offline, REPET online, PARAFAC offline), the audio sample rate (8000 Hz or 16000 Hz) and the platform (PC with Octave, PC with MATLAB, Nexus with Octave and GUI, Nexus with Octave and without GUI). Not all combinations are feasible (e.g. REPET offline with 8000 Hz on the Nexus with Octave and GUI), but the selectable ones are depicted in figures 2 to 6.

Each test case – with the exception of REPET online on the smartphone (only 5 test runs) – is executed 10 times (i.e. 10 test runs per test case), and each test run processes 60 seconds of stereo audio data to identify possible volatilities in runtime performance that occur particularly on the smartphone. The offline algorithms are tested by handing over the entire block of 60 seconds of unseparated audio data to the appropriate Matlab function in one single call. Online tests without the GUI app are performed by separating the 60 seconds offline audio file into 1 second (= 1000 milliseconds) audio data frames, each of which is handed over to the appropriate online function separately, as they would be handed over live by the GUI app. This is to be short enough for sufficient real-time experience, but also long enough to provide the online algorithm with sufficient data. Shorter data frames for the online algorithm are of course intended in the future, as the development and sophistication of the algorithm increases. It is not possible to use the same 1000 ms frame size also for offline algorithms, because this leads to disproportionately longer runtimes: Processing 60 frames of 1000 ms with PARAFAC offline one-by-one took much longer than processing just one 60 seconds frame with PARAFAC offline in one step. Hence the offline algorithms are generally not suited for vast amounts of extremely short audio data frames. But to nevertheless allow for a better comparison between the runtime performance of offline and online algorithms, the offline runtime for the 60 seconds audio data is divided by 60 before averaging across the test runs. A per-second-average is therefore depicted in the offline plots.

B. REPET and PARAFAC Offline

As seen in figure 2 for REPET offline, there is an enormous, almost 5-fold increase in execution time (independent of sample rate) within the PC-based platform alone, that is, only because of switching from MATLAB to Octave. This already demonstrates the high runtime penalty paid for using an Open Source clone of the mathematical computing environment. The penalty is almost equally high with a 4-fold increase when moving from Octave on the PC to Octave on the smartphone. This hints at a high penalty paid in general for using the attempted porting method as a workaround to avoid native reimplementation. Judging by the expectations raised by this offline figure alone, the ratio between runtime performance in MATLAB and an audio frame length for online processing is not allowed to be greater than 1:20. In other words: Already if it takes just 50 ms for an online algorithm to process one second of audio data in MATLAB, it is expected to take 20 times as long, i.e. a full second, on the smartphone. So only the most highly efficient online algorithms are expected to stand a chance on the smartphone with this porting method.

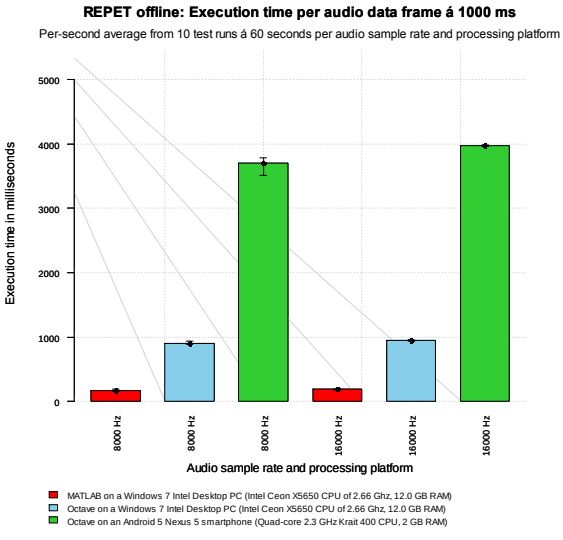


Figure 2. REPET offline

Figure 3 adds not much to what is already expressed by the REPET offline values of figure 2, except that the same conclusions are drawn for PARAFAC offline.

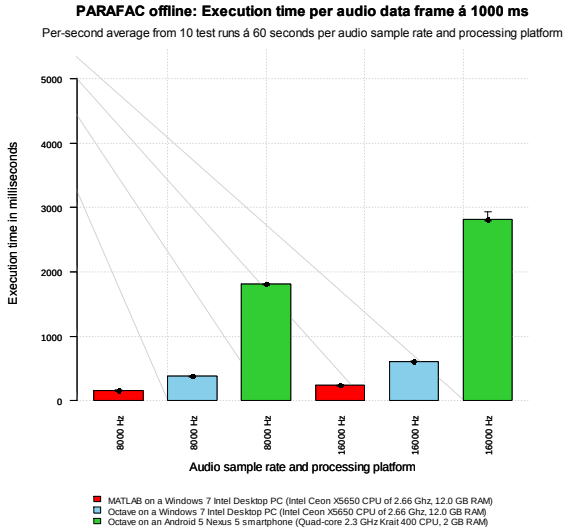


Figure 3. PARAFAC offline

C. *REPET Online*

The expectations raised by the offline performance is clearly met for the online version of REPET, when comparing only the average or median values in the boxplots of figures 4 and 5 (the star symbols indicate the averages). At the same time, figure 4 confirms the real-time capabilities of REPET online promised by the authors when executing on a PC. This also holds for Octave as a PC-based environment.

Figure 5 illustrates how volatile the execution times are on the mobile platform: With a wide interval of processing time differences at roughly a third of a second (8000 Hz) or at a fifth (16000 Hz) for half of the values in each case, and with an average surpassing of the 1000 ms threshold – all when running with the GUI – the runtime performance is not only quite unpredictable but also makes the porting method inadequate for REPET (see first and fourth plot). A less volatile picture is drawn when running without the GUI app (see third and sixth plot): There are very insignificant inter-quartile-ranges independent of sample rate, and at 8000 Hz the runtime is clearly below the real-time threshold for most frames. But what is still observable is the large number of outliers above 1100 ms for both mobile platforms at 16000 Hz (fourth and sixth plot), and frequent data exchange time outliers between the app and Octave (second and fifth plot) that range up to a quarter of a second (although otherwise in most cases in a very narrow inter-quartile range somewhere between 0 and 50 ms).

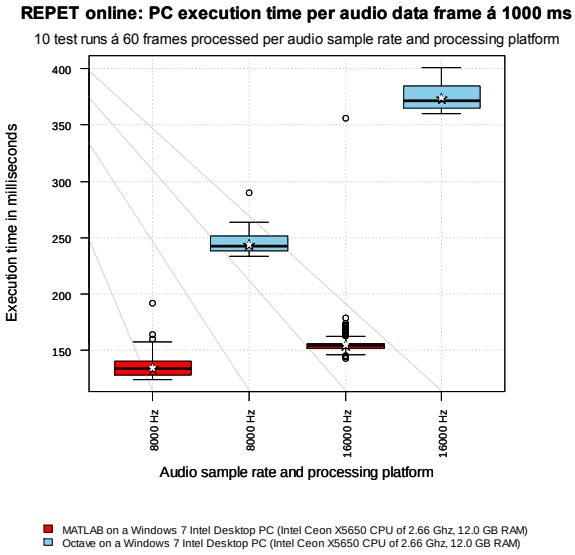


Figure 4. REPET online on a PC platform

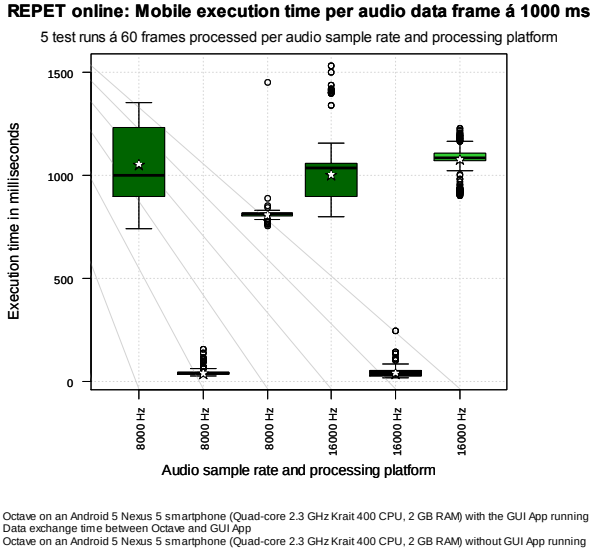


Figure 5. REPET online on a mobile platform

Figure 6 further demonstrates the volatility of the system by plotting the average amount of milliseconds it took to process a data frame of 1000 ms on the phone over time at an audio sample rate of 8000 Hz. This is plotted both for the algorithm running together with the GUI and without the GUI. Dashed lines below and above the graphs indicate the minimum and maximum execution times among the test runs. First of all, the plot indicates an extremity in the first frame of each graph, which is potentially due to initialization problems. But what is seen for the entire interval after the first second,

is that there is a general tendency of the execution time to in-crease, the longer the algorithm runs together with the GUI, while without the GUI an overall stable behavior (no trend) is observed both for the average and the minimum/maximum graphs. On average, the divergence between execution with GUI and without GUI already begins to develop after just 10 seconds into execution. Already at less than 20 seconds into execution, the sum of execution plus data exchange time surpasses the 1000 ms threshold, destroying the real-time effect. Shortly after just half a minute, it is already 200 ms above the real-time threshold and significantly worse than without the GUI. Even more importantly, looking at the sum of the maximum graphs alone, the threshold is surpassed 8 seconds earlier than for the sum of average graphs. Only one second after that, there is a very sudden leap above 1200 ms, which is in just half the time as the sum of average graphs.

Without the GUI, average execution time is predominantly below the threshold. There is hardly any difference between the average and minimum or maximum. All values fluctuate around 800 ms. But this being without the GUI app running, are only performance values of the algorithm running stand-alone for measurement, but without any benefit for the user.

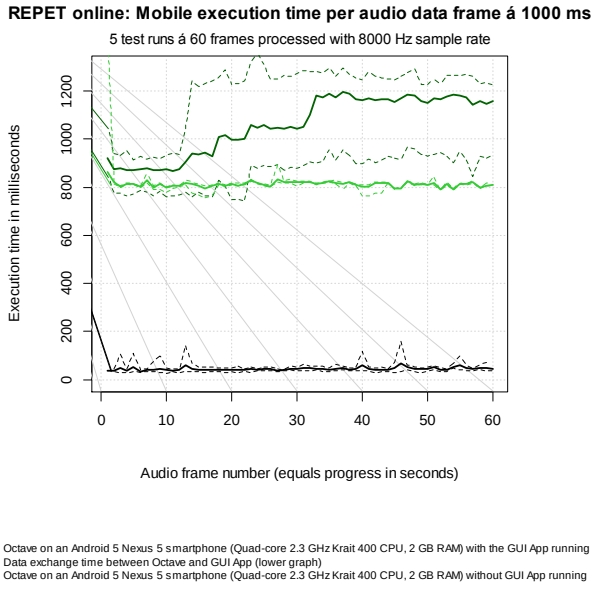


Figure 6. REPET online on a mobile platform in detail

The GUI vs non-GUI execution times indicate the negative impact that running the algorithm together with the GUI (which after all records from the microphone and performs playback) has on the runtime performance of the actual separation algorithm: It causes a worsening performance over time and makes real-time processing – even in telephone quality – impossible for anything longer than a few seconds. This is in spite of the fact that the algorithm – when run by itself – still fulfills the real-time requirements.

D. Summary of the Experiments

The results of the runtime performance benchmark on multiple platform all-together lead to one conclusion: The opportunity of porting state-of-the-art BSS algorithms to a mobile platform without native reimplementation is demonstrably achieved. The same holds for the feasibility of integrating them into a mobile GUI app. What still forms a major problem is the significant loss of run-time performance of the ported algorithms, which is highly critical in the case of real-time algorithms. This is primarily due to the non-native porting method. Furthermore, there are huge problems in achieving and maintaining a stable runtime performance with real-time capabilities when multi-threading the GUI (recording, display and playback) with the algorithm. The latter is, of course, only so far demonstrated for the quite simple online algorithm tested (REPET). This already demonstrates the limits of the porting method at this early stage.

5. Conclusion and Outlook

For an autonomous, self-confident and long productive life, a good speech understanding in everyday life situations is necessary to reduce the listening effort. In this paper, necessary considerations in developing an app-based assistance system are presented that makes every day acoustic scenarios more transparent to hearing-impaired people by the opportunity of an interactive focusing on the preferred sound source. For a quick development of such a system using limited resources, an algorithm porting and app integration method is presented and tested. The runtime performance tests show the inability of the porting method to enable real-time capabilities for the app. As an outlook, outsourcing the signal processing to a MATLAB PC platform is under consideration, for which a significantly higher runtime performance with real-time potential is partly already demonstrated by the tests. Challenges lie predominantly in real-time data exchange over the network.

Acknowledgment

This work as a part of the SMARTNAVI-project is funded by the German Federal Ministry of Education and Research (BMBF) under the registration number 16SV6368. The financial project organization is directed by the VDI/VDE Berlin.

References

[1] H. Zaretsky, S. Flanagan, and A. Moroz, Medical Aspects of Disability, Fourth Edition: A Handbook for the Rehabilitation Professional, ser. Medical Aspects of Disability: A Handbook for the Rehabilitation Professional. Springer Publishing Company, 2010.

[2] E. C. Cherry, "Some experiments on the recognition of speech, with one and with two ears," The Journal of the Acoustical Society of America, vol. 25, no. 5, pp. 975–979, 1953.

[3] M. S. Pedersen, J. Larsen, U. Kjems, and L. C. Parra, "A survey of convolutive blind source separation methods," in Springer Handbook of Speech Processing. Springer Press, 2007.

[4] M. Pedersen, D. Wang, J. Larsen, and U. Kjems, "Two-microphone separation of speech mixtures," Neural Networks, IEEE Transactions on, vol. 19, no. 3, pp. 475–492, March 2008.

[5] Z. Rafii and B. Pardo, "REpeating pattern extraction technique (REPET): A simple method for music/voice separation," IEEE Transactions on Audio, Speech & Language Processing, vol. 21, no. 1, pp. 71–82, 2013.

[6] D. Nion, K. N. Mokios, N. D. Sidiropoulos, and A. Potamianos, "Batch and adaptive PARAFAC-based blind separation of convolutive speech mixtures," Trans. Audio, Speech and Lang. Proc., vol. 18, no. 6, pp. 1193–1207, 2010.

[7] Z. Rafii, "REPET – codes," 2015. [Online]. Available: http://www.zafarrafii.com/repet.html#Codes

[8] D. Nion, "Dr. Dimitri Nion, signal processing for digital communication – Matlab codes," 2015. [Online]. Available: http://dimitri.nion.free.fr/

[9] ARM, "ARM – the architecture for the digital world," 2015. [Online]. Available: http://http://www.arm.com/

[10] PassMark Software, "PassMark Software – CPU benchmarks – CPU popularity in the last 90 days," 2015. [Online]. Available: http://www.cpubenchmark.net/share30.html

[11] CPUBoss, "CPUBoss – best CPUs – all the CPUs ranked in real-time!" 2015. [Online]. Available: http://cpuboss.com/

[12] D. A. Patterson and D. R. Ditzel, "The case for the reduced instruction set computer," SIGARCH Comput. Archit. News, vol. 8, no. 6, pp. 25–33, 1980.

[13] J. Fitzpatrick, "An interview with Steve Furber," Commun. ACM, vol. 54, no. 5, pp. 34–39, 2011.

[14] MathWorks, "MATLAB – the language of technical computing," 2014. [Online]. Available: http://www.mathworks.com/products/matlab/

[15] B. Hunt, R. Lipsman, and J. Rosenberg, A Guide to MATLAB: For Beginners and Experienced Users. Cambridge University Press, 2014.

[16] V. Ingle and J. Proakis, Digital Signal Processing Using MATLAB. Cengage Learning, 2011.

[17] Z. Koldovsky and P. Tichavsky, "Time-domain blind sep- aration of audio sources on the basis of a complete ICA decomposition of an observation space," Audio, Speech, and Language Processing, IEEE Transactions on, vol. 19, no. 2, pp. 406–416, Feb 2011.

[18] Gartner, "Gartner: Worldwide smartphone sales to end users by operating system in 2013," feb 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2665715

[19] J. Drake, Z. Lanier, C. Mulliner, P. Fora, S. Ridley, and G. Wicherski, Android Hacker's Handbook. Wiley, 2014.

[20] Gartner, "Gartner: Worldwide traditional PC, tablet, ultramobile and mobile phone shipments on pace to grow 7.6 percent in 2014," jan 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2645115

[21] K. Yaghmour, Embedded Android: Porting, Extending, and Customizing, ser. Oreilly and Associate Series. O'Reilly Media, Incorporated, 2013.

[22] M. Gargenta and M. Nakamura, Learning Android: Develop Mobile Apps Using Java and Eclipse. O'Reilly Media, 2014.

[23] Android Developers, "Starting another Activity," 2014. [Online]. Available: http://developer.android.com/training/basics/firstapp/starting-activity.html

[24] Google, "Matlab Mobile – Android apps on Google Play," 2014. [Online]. Available: https://play.google.com/store/apps/details?id=com.mathworks.matlabmobile&hl=en

[25] MathWorks, "MATLAB Coder," 2014. [Online]. Available: http://www.mathworks.com/products/matlab-coder/

[26] MathWorks, "Functions and objects supported for C and C++ code generation categorical list," 2014. [Online]. Available: http://www.mathworks.com/help/coder/ ug/functions-supported-for-code-generation--categorical-list. html

[27] MathWorks, "Features – MATLAB Coder," 2014. [Online]. Available: http://www.mathworks.com/products/matlab-coder

[28] Google, "Android NDK," 2014. [Online]. Available: https://developer.android.com/tools/sdk/ndk/index.html

[29] S. Chapman, MATLAB Programming for Engineers. Cengage Learning, 2007.

[30] MathWorks, "MATLAB Compiler," 2014. Online]. Available: http://www.mathworks.com/products/compiler/

[31] Sourceforge.net, "Octave-Forge – extra packages for GNU Octave," 2014. [Online]. Available: http://octave.sourceforge.net/

[32] A. Quarteroni, F. Saleri, and P. Gervasio, Scientific Computing with MATLAB and Octave, ser. Texts in Computational Science and Engineering. Springer, 2010.

[33] J. Hansen, GNU Octave: Beginner's Guide: Become a Proficient Octave User by Learning this High-level Scientific Numerical Tool from the Ground Up, ser. Learn by doing : less theory, more results. Packt Publ., 2011.

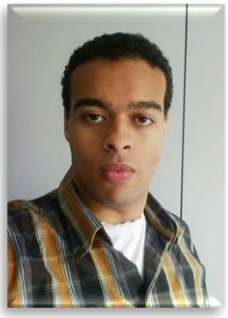
[34] Google, "Octave – Android apps on Google Play," 2014. [Online]. Available: https://play.google.com/store/apps/details?id=com.octave

[35] Y. Yu and S. U. of New York at Stony Brook, OS-level Virtualization and Its Applications. State University of New York at Stony Brook, 2007.

[36] S. Smith, MATLAB: Advanced GUI Development. Dog Ear Pub., 2006.

[37] Google, "Linux Installer – Android apps on Google Play," 2014. [Online]. Available: https://play.google.com/store/apps/details?id=com.zpwebsites.linuxonandroid

[38] P. Aksoy and L. DeNardis, Information Technology in Theory, ser. Information Technology Concepts Series. Thomson Course Technology, 2007. [Online].



Marvin Chibuzo Offiah Since 2011, Marvin C. Offiah is a member of the research group "Optimized Systems" which focusses on research in BMBF-founded and industrial projects in the area of acoustic and digital signal processing as well as evolutionary optimization in various areas, for instance Chemoinformatics. Since more than four years, he researches in the field of Digital Signal Processing, and especially in the area of Applied Computer Science. Over the years, he published 5 articles and journals on these topics. He received his diploma degree in Geoinformatics from the Institute for Geoinformatics at the University of Münster in 2011.

Markus Borschbach Since 2009, Markus Borschbach is an Associated Professor at the University of Applied Sciences in Bergisch Gladbach (Germany). Here, he founded the Competence Centre and the research group "Optimized Systems", which focusses on research in BMBF-founded and industrial projects in the area of acoustic and digital signal processing as well as evolutionary optimization in various areas, for instance Chemoinformatics. Since more than fifteen years, he researches in the field of artificial intelligence and especially in the area of Applied Computer Science. Over the years, he published more than 60 articles and journals on these topics. He earned his PhD from the University of Münster, department of mathematics and computer science in 2002, where he stayed for a further five years as post-doctoral researcher and team leader. He took a duty as a visiting professor at the department for computer networks and distributed systems at the University of Chemnitz (Germany) from 2006 to 2007. He received his diploma degree of engineering (Dipl.-Ing.) in Technical Computer Science from the department of Electrical Engineering and Computer Science at the University of Siegen in 1996 and a call as a fulltime Professor at the faculty of Computer Science in Bergisch Gladbach in 2008.