Pulseq: A rapid and hardware-independent pulse sequence prototyping framework

Kelvin J. Layton[1], Stefan Kroboth[1], Feng Jia[1], Sebastian Littin[1], Huijun Yu[1], Jochen Leupold[1], Jon-Fredrik Nielsen[2], Tony Stöcker[3], Maxim Zaitsev[1]

[1] Department of Radiology, Medical Physics, University Medical Center Freiburg, Freiburg, BW, Germany
[2] Department of Biomedical Engineering, University of Michigan, Ann Arbor, MI, USA
[3] German Center for Neurodegenerative Diseases, Bonn, NRW, Germany

Word count: 3568
Running title: Rapid hardware-independent pulse sequence prototyping

Address for correspondence:

Kelvin Layton
Department of Radiology, Medical Physics,
University Medical Center Freiburg, Freiburg, Germany
kelvin.layton@uniklinik-freiburg.de

Abstract

Purpose: Implementing new magnetic resonance experiments, or sequences, often involves extensive programming on vendor-specific platforms, which can be time consuming and costly. This situation is exacerbated when research sequences need to be implemented on several platforms simultaneously, e.g., at different field strengths. This work presents an alternative programming environment that is hardware-independent, open-source and promotes rapid sequence prototyping.

Methods: A novel file format is described to efficiently store the hardware events and timing information required for an MR pulse sequence. Platform-dependent interpreter modules convert the file to appropriate instructions to run the sequence on MR hardware. Sequences can be designed in high-level languages, such as MATLAB, or with a graphical interface. Spin physics simulation tools are incorporated into the framework, allowing for comparison between real and virtual experiments.

Results: Minimal effort is required to implement relatively advanced sequences using the tools provided. Sequences are executed on three different MR platforms, demonstrating the flexibility of the approach.

Conclusion: A high-level, flexible and hardware-independent approach to sequence programming is ideal for the rapid development of new sequences. The framework is currently not suitable for large patient studies or routine scanning although this would be possible with deeper integration into existing workflows.

Keywords: Pulseq, pulse sequence programming, rapid development, platform independent, open-source

Introduction

The rich diversity of magnetic resonance imaging (MRI) applications critically depends on the ability to coordinate various hardware components through a software program known as a pulse sequence. This flexibility coupled with the continued need for improved MRI sequences (increased tissue contrast, shorter scan times, etc.) has lead to a plethora of acquisition techniques using a standard hardware setup. Despite this flexibility, implementation of pulse sequences remains an arduous task for researchers and students. Extensive development environments are provided by MR manufacturers; however, sequence programming typically involves low-level programming with C, C++ or custom programming languages. This often inhibits researchers, whose focus is to quickly test new ideas, demonstrate novel physics or compare different approaches. Furthermore, the environment is extremely vendor-specific, which impedes the translation of research across different institutions.

Some alternative programming environments have attempted to overcome these difficulties. The Object-Oriented Development Interface for NMR (ODIN) provides a platform-independent library for pulse programming (1). Likewise, a modular Java platform for sequence programming is described in (2). However, both frameworks appear overly complicated requiring hundreds of lines of source code or extensive configurations to define a basic sequence, rendering them unsuitable for rapid sequence development.

The open-source SequenceTree package (3) provides a comprehensive graphical interface for platform-independent sequence programming and simulation. Sequences are executed by exporting vendor-dependent C++ code, which must be compiled and installed on the scanner. Although SequenceTree provides an interactive preview of the sequence during development (3), the compilation step increases the time between sequence modification and execution, potentially limiting the approach for rapid development. Furthermore, sequences are currently restricted to trapezoidal gradients and to the best of the author's knowledge can only be executed on one hardware platform.

In contrast to the open-source programming environments intended to run on standard MR systems, some researchers have opted to replace the entire console with custom hardware. For example, Medusa (4), provides a scalable console including RF waveform generation, gradient control and ADC sampling. Another console was developed in the TMX platform (5), which tightly integrates real measurements and simulation. Whilst these approaches provide the maximum flexibility and control, they prevent normal CE/FDA-approved operation of the scanner, for example, in a clinical setting.

Other sequence programming enviroments have emerged to perform accurate

MRI simulations, such as SpinBench (6) and JEMRIS (7). SpinBench can be paired with the RTHawk platform to execute sequences; however, it is not open-source and thus difficult to extend to other platforms. Alternatively, JEMRIS is primarily a simulation tool although the graphical interface is open-source, extensible and platform-independent. Other open-source projects provide a means to customize the image reconstruction pipeline (8–10) although these do not help with data acquisition. R2.1

In this current work, we implement a highly-flexible pulse sequence programming environment named Pulseq, which overcomes some of the limitations of previous approaches. Central to our method is a novel file format to compactly describe the low-level details of a sequence. This approach decouples the sequence design from the hardware implementation, providing a high-degree of flexibility. We provide examples where sequences are defined with MATLAB code or the JEMRIS graphical interface and executed on different platforms. Currently, three vendor platforms are supported: Siemens, GE, and Bruker. No compilation of source code is required so arbitrary sequences can be executed immediately, as desired for rapid sequence prototyping.

Methods

The main components of the Pulseq environment are illustrated in Fig. 1. The high-level sequence can be described directly in MATLAB (The Mathworks, Natick, MA) using functions from a custom toolbox. Alternatively, sequences can be programmed using the graphical interface of the JEMRIS simulation package (7). Regardless of the choice of high-level interface, a sequence file is created containing low-level sequence instructions such as RF pulses, gradients, ADC events and delays. This sequence file can then be executed on various platforms through hardware-dependent interpreter modules. This architecture allows for maximum flexibility of the high-level interface and the target scanner hardware, since the two components are decoupled through the low-level sequence file.

High-level sequence definition

The high-level sequence can be defined in multiple ways without compromising the ability to execute the sequence on various platforms. Fig. 2 presents the MATLAB source code required to define a basic gradient echo sequence. The code uses functions from the `mr` toolbox provided with the Pulseq project to simplify common calculations in sequence programming. The entire sequence is defined with standard MATLAB variables and structures, already familiar to

a vast number of researchers. The `Sequence` object compresses the sequence events and outputs a sequence file in the Pulseq format (described below) suitable for execution. This new toolbox is provided for two reasons. First, it allows researchers already designing trajectories or pulses in MATLAB to execute them immediately from the same environment. Secondly, the toolbox demonstrates that multiple high-level design tools are possible in the proposed architecture.

An alternative method to define a sequence is to use the graphical user interface provided by the simulation package JEMRIS (7), shown in Fig. 3. The GUI provides a drag-and-drop interface to define the sequence as a tree structure. Advantages of this approach include instant visualization of the gradient and RF waveforms as the sequence is updated. Further, sequence events can be added and removed with a few mouse clicks. In this work, we modified the JEMRIS C++ code (available in version 2.8) to recurse over the sequence tree and write the hardware events into a Pulseq sequence file. In this way, any sequence defined in JEMRIS can be executed on real MR hardware. An advantage of this approach is that exactly the same sequence can be simulated with the Bloch equations and executed on a scanner. This is similar to the simulation capabilities in other works (3, 5) and provides a useful comparison between simulations and measurements.

Although the high-level sequence definition is vendor-independent, hardware constraints such as maximum gradient amplitude and slew rate are incorporated at this level. Violation of these limits are reported to the user during the sequence design. The constraints are necessary to calculate the precise timing of gradient events prior to export for scanner execution. Likewise, gradient timing is rounded to 10 μs during the sequence calculation while preserving amplitude or area requirements. This is the longest 'raster time' of the three investigated hardware platforms.

In this work, slice localization is performed in a separate graphical interface that integrates with the high-level design tools. A screenshot and description of this interface is provided as Supporting Figure S1 in the online supporting material.

Low-level sequence file

The Pulseq sequence file format was designed to represent MR sequences with the following goals:

1. **Low-level:** The format should be sufficiently low-level. This allows for maximum flexibility of the high-level sequence definition and allows for simple hardware implementation.

2. **Compact:** The file size should be minimised. This is achieved by preventing redundant definitions of pulses and their parameters.
3. **Human-readable:** The basic sequence structure should be easily understood without processing to aid debugging. This necessitates a text-file format.
4. **Easily parsed:** The format should be easy for a computer to parse without the need for external libraries. This precludes existing formats such as XML.
5. **Vendor independent:** The sequence must not contain definitions specific to a particular hardware manufacturer. For example, units such as Tesla and Volts may be required to implement low-level commands but not to define the basic spin operations constituting a sequence. The file format use units of Hertz, meter and second.

The resulting text file is hierarchical and consists of a timing table, which references 'event' objects, which in turn can reference compressed 'shape' objects. Fig. 4 illustrates these basic concepts of the file format. The file contains no loops but a simple list of instructions. This moves much of the logic to the chosen high-level sequence tool and is made possible by the increased memory and performance of the microcontroller hardware used in modern scanners.

The definition of shape objects allow for arbitrary RF and gradient pulses to be executed on scanner hardware. The shapes are stored using a run-length compression scheme on the signal derivative. This scheme highly compresses constant and linear segments of arbitrary shapes (e.g. block pulses or piecewise-linear gradients). Other shapes can also benefit from this compression when linear segments are used to approximate a continuous waveform. A further advantage of this compression is that minimal computation is required for encoding and decoding, unlike more advanced algorithms, e.g. audio compression (11).

The file specification also defines a mechanism for user-specific header information. This allows simple extensions to be implemented with the current format. In this work, for example, slice localization is performed in a separate graphical interface (Supporting Figure S1) and the gradient rotation matrix is passed in the file header. A detailed file specification is available from the Pulseq website (pulseq.github.io).

Interpreter modules

Implementation of the sequence on a real MR scanner inevitably relies on vendor-specific hardware instructions. These instructions are initiated by an

'interpreter module', which translates the sequence file to appropriate hardware commands, as illustrated in Fig. 1. The low-level nature of the Pulseq sequence file makes it relatively simple to implement interpreter modules for different scanner platforms. Precise timing logic and amplitude information is already computed by the high-level sequence tool and stored in the file. The remaining task of the interpreter is to convert each sequence event into an appropriate hardware instruction. There is no guarantee that a single Pulseq file can run on all platforms, due to varying hardware and safety constraints. In this work, sequences were created conservatively such that they satisfy the constraints of all systems. However, if optimization for a specific platform is required, this must be performed at the design stage prior to generating the Pulseq file. The simplicity of this architecture is demonstrated here by successful implementation of interpreter modules for three scanner platforms. R2.3

The setup makes it relatively simple to deal with different vendor software versions, since only the interpreter module needs to be modified while the sequence files remain unchanged. Another advantage of the interpreter architecture, compared to other solutions, is that vendor-specific code does not need to be recompiled prior to executing a new sequence. Thus a sequence can be changed (e.g. by adding gradient pulses) and executed immediately on the scanner. This enables very rapid development and debugging of sequences.

Experiments

Arbitrary RF and gradient shapes

A gradient echo sequence with matrix size 256×256, field-of-view 220mm, flip angle 20°, TE=20 ms, TR=100 ms was defined with the MATLAB source code shown in Fig. 2. This sequence was used to image a cylindrical phantom containing thin Plexiglas tubes on a 3 T system (Siemens Healthcare, Erlangen, Germany) with the proposed file format and interpreter module.

In addition to a simple gradient echo, a 2D spatially selective RF pulse was implemented in order to demonstrate arbitrary gradient and RF pulse shapes. The RF pulse design closely follows (12), modified to excite the superposition of the original target pattern with a shifted version. Specifically, the RF pulse and excitation k-space trajectory have the form,

$$B_1(t) = B_1^{\mathrm{e}}(t)(1 + e^{j2\pi \boldsymbol{x}_0^T \boldsymbol{k}(t)}) \quad [1]$$

$$B_1^{\mathrm{e}}(t) = \alpha e^{-\beta^2(1-t/T)^2} \sqrt{(2\pi n(1-t/T))^2 + 1} \quad [2]$$

$$\boldsymbol{k}(t) = A(1-t/T) \begin{bmatrix} \cos(2\pi n t/T) \\ \sin(2\pi n t/T) \end{bmatrix} \quad [3]$$

where $T = 8\,\text{ms}$ is the pulse duration, $\beta = 2$ corresponding to a Gaussian target excitation region of approximately 3cm, α was set to achieve a 20° flip angle, $n = 8$ is the number of spiral turns and $A = 40\,\text{m}^{-1}$ is the k-space maximum. The sequence timing was TE=20 ms and TR=500 ms. Modulation by the complex exponential in Eq. [1] excites a duplicate pattern at $\boldsymbol{x}_0 = (5\,\text{cm}, 5\,\text{cm})$, chosen to demonstrate pulses with arbitrary phase. Fig. 5 illustrates the final gradient and RF pulses.

The original gradient echo sequence was modified with approximately 20 additional lines of MATLAB code to define the excitation parameters, complex RF pulse and gradient waveforms and include them in the sequence for execution. This includes conversion of the sequence into a spin echo with a spoiled 180° slice-selective refocusing pulse and additional delays to select a slice through the excited cylinders, as described in (12).

Comparison of simulated and measured data

A spin-echo sequence was created in JEMRIS with the graphical interface and executed on a Siemens scanner. This demonstrates Pulseq integration with an existing high-level sequence design tool and allows for the comparison of simulation data and data acquired on an MRI scanner. The sequence had a matrix size of 64×64, field-of-view of 210 mm, flip angle of 50°, TE of 15 ms and TR of 100 ms.

High-resolution maps of the properties of a phantom (M_0, T_1, T_2, T_2^*) were acquired in order to simulate the sequence in JEMRIS. Parameter maps of a single 3 mm slice at the isocenter were calculated as follows. A multi-echo Carr-Purcell-Meiboom-Gill (CPMG) sequence with 16 echoes spaced 13.2 ms apart was used to fit each voxel to a single exponential function, producing maps of proton density and T_2 (13). Likewise, T_2^* maps were generated by voxel-wise fitting of an exponential to 8 echoes acquired 4 ms apart with a multi-echo gradient echo sequence. Mapping of T_1 was performed with an inversion recovery sequence with inversion times (in ms) of 22, 30, 50, 150, 220, 300, 1000, and 2000. Finally, a B_0 off-resonance map was calculated from the phase of two gradient echo images with echo times spaced 1 ms appart. All data were acquired using standard vendor sequences at 256×256 resolution with a field-of-view of 210 mm. The final maps were interpolated to 512×512 to reduce simulation artifacts caused by approximating a continuous integral (14).

The target sequence was simulated in JEMRIS using the calculated parameter maps and an image was generated with a 2D discrete Fourier transform (DFT). The low resolution of the target sequence allows the simulations to accurately capture intra-voxel dephasing (15, 16).

Platform independent sequences

A gradient echo sequence was designed in JEMRIS to run on three different hardware platforms: 3 T Siemens Trio equipped with a single-channel wrist RF coil (Siemens Healthcare, Erlangen, Germany); 3 T GE Discovery MR750 with an 8 channel head coil (GE Healthcare, Waukesha, WI, USA); and 9.4 T Bruker BioSpec MRI with a single-channel rat coil (Bruker Biospin, Ettlingen, Germany). The scanners were located across two institutions. The sequence had a field-of-view of 80×80 mm to ensure reasonable imaging in both the small-bore 9.4 T and the human 3 T systems. Other parameters were: 256×256 matrix, flip angle=20°, TE=7 ms and TR=100 ms. Data from the GE system were averaged 20 times to account for the loss of SNR due to the increased receive coil size.

All images were reconstructed using a 2D DFT on the raw data and sum-of-square combination was used in the case of multiple RF channels.

Results

Fig. 6 displays images acquired from sequences defined entirely in MATLAB. The gradient-echo image in Fig. 6a represents a slice through the phantom, as expected. Fig. 6b is the result from RF and gradient waveforms designed to achieve 2D selective excitation of two Gaussian profiles. These images demonstrate the correct implementation of the sequence design, low-level sequence file, and interpreter module for arbitrary pulse shapes.

Fig. 7 demonstrates the close match between simulated and measured images. The direct comparison is possible since the same sequence is simulated and also converted to hardware instructions for the MR scanner. Minor contrast differences are visible, possibly due to B_1 inhomogeneity or inaccurate estimation of the phantom parameters.

Fig. 8 presents images acquired from the same sequence file on three different MR platforms. The first two images of a phantom were acquired at the University Medical Center Freiburg on a 3 T Siemens and 9.4 T Bruker scanner, respectively; the third image of an orange was measured at University of Michigan on a 3 T GE scanner. The images in Fig. 8a and 8b differ slightly due to different RF coil characteristics and the increased B_0 and B_1 inhomogeneities at the higher field strength. The variety of platforms demonstrates the flexibility of the proposed sequence interpreter framework.

The compact hierarchical design of the file format results in relatively small sequence files. Table 1 lists the file sizes and compression ratios for the sequences used in the results above. Compression ratios are calculated as a percentage of the size of the uncompressed waveform data. In all cases, the entire sequences

file was sufficiently small to fit in the memory of the hardware control units.

Discussion

Flexibility

A selection of examples was chosen for this publication although the flexibility of the framework is much greater. A range of other sequences can be easily implemented, depending on the given application. Furthermore, any sequence defined in MATLAB could have also been designed in JEMRIS, and vice versa. The choice between MATLAB scripting and JEMRIS is largely left to the developer. For example, some researchers prefer graphical interfaces while others may prefer to output a sequence from the same MATLAB script containing a pulse calculation. The advantage of JEMRIS, however, is the ability to simulate as well as execute sequences.

In addition to state of the art sequences using standard sequence blocks, the inclusion of arbitrary RF and gradient pulse shapes allows for a range of advanced sequences to be implemented as required for cutting-edge MR research. For example, frequency swept adiabatic pulses (17), oscillating gradients for diffusion measurements (18), continuous wave acquisition (19) and acoustic noise reduction (20) can all be implemented without modification of the basic framework.

Openness

The pulse sequence programming environment presented here is open-source to encourage contributions from other researchers. Unlike other open-source projects, such as (1, 3), the focus here is an open format to represent sequences, suitable for execution on any MR platform. It is our opinion that existing projects (both open-source and proprietary) would also benefit from a common sequence file format. In this case, when a programming interface can export sequences to this format, they can automatically be run on various hardware platforms using the interpreter modules provided. This is analogous to other file types, such as images or documents, that have benefited from a common format to share data.

Limitations

The framework presented here is primarily targeted to research and education, where the objective is to rapidly develop and test new sequences. As such, the low-level sequence format was designed for simplicity and portability. Features

such as physiological triggering and multiple slice rotations were deliberately omitted, although the framework could easily be extended to include these. Likewise, the addition of multiple RF transmit channels is also possible. A similar extension to multiple nonlinear encoding fields was used for data acquisition in (21).

Advanced features such as real-time feedback (22) would require some implementation effort and may not be feasible. Another limitation is aggressively time-optimized sequences where, for example, gradient ramps of one block can overlap into another block. The absence of loop structures in the file format leads to an increase in the sequence file size, particularly for long sequences such as 3D or diffusion. In this case, the interpreter modules may need to load the file in sections, during the sequence execution.

Safety

There are no inherent safety concerns, e.g. peripheral nerve stimulation (PNS) or specific absorption rate (SAR), using this method of sequence programming. The platforms for which interpreter modules were implemented so far perform safety checks at a hardware level further along the chain than the environment of the interpreter modules. This is similar to how custom sequences, implemented in vendor-specific programming environments, will not run if they do not pass the safety tests. Therefore sequences designed with Pulseq are applicable in vivo under the IRB approval conditions similar to other research sequences.

To provide feedback to the sequence programmer prior to scanner execution, some basic checks are performed by the high-level design tool, such as maximum gradient and slew rate. Additional safety constraints such as PNS or SAR are either performed at run-time or left to the interpreter module, depending on the platform. When safety violations are reported by the vendor interfaces, the user must adjust the sequence and export a new Pulseq file. This iterative approach is suitable for prototyping but suboptimal in a clinical setting. In future, more complex checks could be implemented in the high-level design tool to consider the safety constraints with respect to the dependencies between sequence blocks. R2.2

Future directions

It is hoped that this publication will inspire other researchers to create interpreter modules for additional hardware platforms. For example, the ability to design and execute sequences in a simple manner, makes the proposed environment ideal for existing MR hardware projects targeted towards education such as (23–25). The translation of high-level sequence logic to human-readable hardware events, combined with the corresponding measurements, provides new

opportunities for teaching the principles of MR. The framework is also suitable for short Master's or summer projects, since the favorable learning curve means novel data can be obtained quickly.

Advantages of the proposed framework are vendor-independence and the minimal time between design and acquisition. Nonetheless, deeper integration into the existing vendor interfaces would bring several advantages. For example, SequenceTree (3) tightly integrates with an existing vendor interface to allow for interactive slice prescription and parameter adjustment at scan time. It may be possible to incorporate similar ideas into the Pulseq framework.

In addition to the format extensions discussed above, an extensive library of different sequences is required to promote adoption of the programming environment. These sequences should first be created in a high-level design tool (e.g. JEMRIS or MATLAB), to allow the operator to easily change parameters prior to export to the low-level Pulseq format. New sequences are continually being added to the project as the use increases across our various institutions. Furthermore, the open-source nature of the project is expected to encourage other users to contribute their own sequences.

Source code and availability

The source code for sequence design and file operations is available from the project website http://pulseq.github.io or via the ISMRM site *MRI Unbound*. The source code is released under the MIT license and the file format is released under the Creative Commons Attribution 4.0 license. The interpreter modules cannot be openly published due to the use of proprietary sequence programming code; however, they are available on request.

Conclusion

The Pulseq project is a flexible framework to create MR sequences and immediately execute them on real hardware, making it ideal for rapid sequence development. Central to the approach is a novel sequence file format describing all low-level events of a sequence, including arbitrary gradient and RF pulse shapes. A standardized file format promotes a variety of high-level design tools and supports implementation on different scanner platforms. Sequence simulation can also be integrated into the framework, which provides useful insights into sequence design, MR physics and signal modeling.

Acknowledgements

This work was in part supported by European Research Council (ERC) grant 282345 'RANGEmri'. The authors thank Dr Denis Kokorin for useful discussions regarding 2D RF pulses.

References

1. Jochimsen TH and von Mengershausen M. ODIN – Object-oriented Development Interface for NMR. Journal of Magnetic Resonance 2004;170:67.
2. Debbins J, Gould K, Halleppanavar V, Polzin J, Radick M, Sat G, Thomas D, Trevino S, and Haworth R. Novel software architecture for rapid development of magnetic resonance applications. Concepts in Magnetic Resonance 2002;15:216–237.
3. Magland JF, Li C, Langham MC, and Wehrli FW. Pulse sequence programming in a dynamic visual environment: SequenceTree. Magnetic Resonance in Medicine 2015:(Early view) doi:10.1002/mrm.25640.
4. Stang PP, Conolly SM, Santos JM, Pauly JM, and Scott GC. Medusa: A scalable MR console using USB. IEEE Transactions on Medical Imaging 2012;31:370–379.
5. Sharp JC, Yin D, Bernhardt RH, Deng Q, Procca AE, Tyson RL, Lo K, and Tomanek B. The integration of real and virtual magnetic resonance imaging experiments in a single instrument. Review of Scientific Instruments 2009;80.
6. Overall WR and Pauly JM. An Extensible, Graphical Environment for Pulse Sequence Design and Simulation. In Proceedings of the ISMRM 15th Annual Meeting. 2007. p. 1652.
7. Stöcker T, Vahedipour K, Pflugfelder D, and Shah NJ. High-performance computing MRI simulations. Magnetic Resonance in Medicine 2010;64:186–193.
8. Hansen MS and Sørensen TS. Gadgetron: an open source framework for medical image reconstruction. Magnetic Resonance in Medicine 2013;69:1768–1776.
9. Han F, Zhou Z, Sung K, Finn JP, and Hu P. A low-cost flexible nonlinear parallelized MR image reconstruction system. In Proceedings of the ISMRM 23rd Scientific Meeting and Exhibition. 2015. p. 2489.

10. Uecker M, Ong F, Tamir JI, Bahri D, Virtue P, Cheng JY, Zhang T, and Lustig M. Berkeley Advanced Reconstruction Toolbox. In Proceedings of the ISMRM 23rd Scientific Meeting and Exhibition. 2015. p. 2484.

11. Xiph.Org Foundation. 2014. FLAC: Free Lossless Audio Codec. URL: `xiph.org/flac/index.html`.

12. Pauly J. A k-space analysis of small-tip-angle excitation. Journal of Magnetic Resonance 1969 1989;81:43–56.

13. Layton KJ, Morelande M, Wright D, Farrell PM, Moran B, and Johnston LA. Modelling and Estimation of Multicomponent T2 Distributions. IEEE Transactions on Medical Imaging 2013;32:1423–1434.

14. Sharp J, Yin D, Tyson R, Lo K, and Tomanek B. An Integrated MR Console / MR Physics Simulation System. Proceedings 14th Scientific Meeting, International Society for Magnetic Resonance in Medicine 2006;2402:1351.

15. Latta P, Gruwel MLH, Jellúš V, and Tomanek B. Bloch simulations with intra-voxel spin dephasing. Journal of Magnetic Resonance 2010;203:44–51.

16. Layton K, Kroboth S, Jia F, Littin S, Yu H, and Zaitsev M. Improved reconstruction of nonlinear spatial encoding techniques with explicit intra-voxel dephasing. In Proceedings of the ISMRM 23rd Scientific Meeting and Exhibition. 2015. p. 98.

17. Tannus A and Garwood M. Adiabatic pulses. NMR in Biomedicine 1997;10:423–434.

18. Schachter M, Does MD, Anderson aW, and Gore JC. Measurements of restricted diffusion using an oscillating gradient spin-echo sequence. Journal of magnetic resonance 2000;147:232–237.

19. Idiyatullin D, Corum C, Park JY, and Garwood M. Fast and quiet MRI using a swept radiofrequency. Journal of Magnetic Resonance 2006;181:342–349.

20. Hennel F, Girard F, and Loenneker T. 'Silent' MRI with soft gradient pulses. Magnetic Resonance in Medicine 1999;42:6–10.

21. Layton KJ, Kroboth S, Jia F, Littin S, Yu H, and Zaitsev M. Trajectory optimization based on the signal-to-noise ratio for spatial encoding with nonlinear encoding fields. Magnetic Resonance in Medicine 2015;(in press).

22. Maclaren J, Herbst M, Speck O, and Zaitsev M. Prospective motion correction in brain imaging: A review. Magnetic Resonance in Medicine 2013;69:621–636.

23. Wright SM, Brown DG, Porter JR, Spence DC, Esparza E, Cole DC, and Huson FR. A desktop magnetic resonance imaging system. Magnetic Resonance Materials in Physics Biology and Medicine 2002;13:177–185.

24. Halse ME, Coy A, Dykstra R, Eccles C, Hunter M, Ward R, and Callaghan PT. A practical and flexible implementation of 3D MRI in the Earth's magnetic field. Journal of Magnetic Resonance 2006;182:75–83.

25. Cooley CZ, Stockmann JP, LaPierre C, et al. Implementation of low-cost, instructional tabletop MRI scanners. In Proceedings of the ISMRM 22nd Scientific Meeting and Exhibition. 2014. p. 4819.

List of Tables

Table 1: The sequence duration, file size and compression ratio of the Pulseq sequence files used in this work. Compression ratios are calculated as a percentage of the size of the uncompressed waveform data. Sequences are designed in MATLAB or JEMRIS before exporting to the scanner.

Sequence	Design	Duration	Size	Compression
GRE	MATLAB	25.6 s	80 KB	0.017%
SE 2D RF	MATLAB	128.0 s	193 KB	0.008%
SE	JEMRIS	6.4 s	39 KB	0.033%
GRE	JEMRIS	25.6 s	65 KB	0.014%

List of Figures

Figure 1: Overview of the Pulseq environment. Sequences are described in a high-level design tool, e.g. a MATLAB script or using a graphical user interface (left). A hardware-independent sequence format is output (middle) and executed using a hardware-dependent interpreter module (right). Simulation data may also be generated from the Bloch equation solver JEMRIS.

Figure 2: A gradient-echo sequence defined in MATLAB with Pulseq toolbox functions. The resulting low-level sequence file is suitable to execute on any MR hardware platform equipped with an interpreter module.

Figure 3: A screenshot of the modified JEMRIS graphical interface for sequence design. Sequences can be exported to a Pulseq low-level sequence file for execution on the scanner.

Figure 4: Main elements of the Pulseq hierarchical data format describing a simple FID. At the top level, the sequence consists of blocks, which contain integer IDs of sequence events. Sequence events may contain IDs of arbitrary shape objects to describe, for example, an RF pulse shape.

Figure 5: Gradient and RF pulse shapes to achieve 2D selective excitation. The (top) RF magnitude and (middle) RF phase combined with (bottom) a spiral trajectory excites two Gaussian cylinders as described in the text. Arbitrary pulse shapes are inherently supported in the proposed sequence format and interpreter modules.

Figure 6: Image acquired directly from MATLAB for (a) the gradient-echo sequence shown in Fig. 2 and (b) a spin-echo sequence with 2D selective RF excitation. The Pulseq framework converts the MATLAB source code to hardware-dependent instructions to control the scanner.

Figure 7: An axial image of a phantom (a) simulated with the Bloch equations and (b) acquired on a scanner. The same spin echo sequence file was used for simulation and measurement.

Figure 8: Images from the same sequence file executed on (a) Siemens, (b) Bruker and (c) GE hardware platforms. The sequence is stored in a novel platform-independent file and converted to hardware instructions by platform-specific interpreter modules.

Figure S1: A screenshot of the custom slice geometry interface. The interface obtains localiser images through an in-house reconstruction pipeline. Geometry information is passed via a MATLAB structure to the high-level design tools of the Pulseq environment, where appropriate sequence parameters are set (including frequency and phase offsets). The gradient rotation matrix is passed to the vendor-specific interpreter modules through the Pulseq file header. Although the interface tightly integrates with the Pulseq environment, it is not currently provided as open-source due to vendor-dependencies.

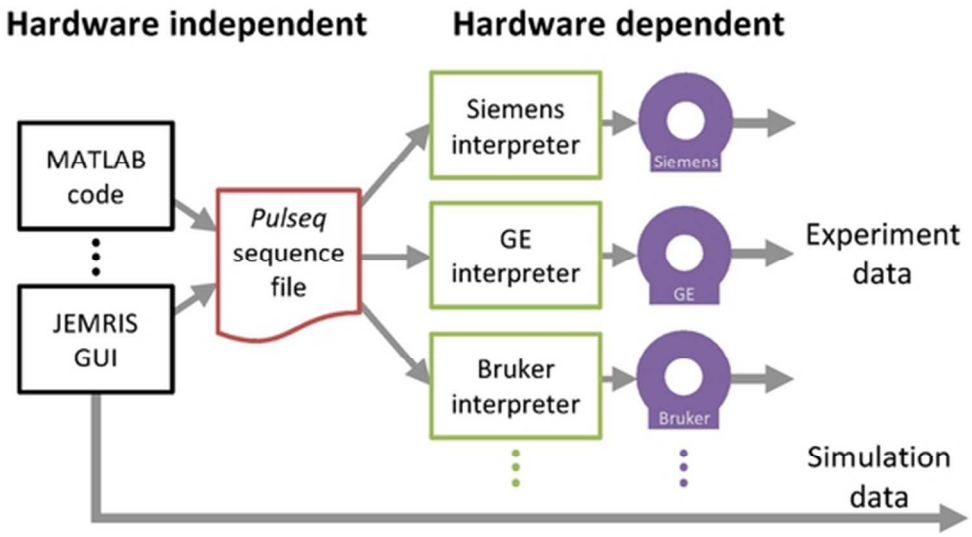

Figure 1: Overview of the Pulseq environment. Sequences are described in a high-level design tool, e.g. a MATLAB script or using a graphical user interface (left). A hardware-independent sequence format is output (middle) and executed using a hardware-dependent interpreter module (right). Simulation data may also be generated from the Bloch equation solver JEMRIS.
49x29mm (300 x 300 DPI)

```
seq=mr.Sequence();              % Create a new sequence object
fov=220e-3; Nx=256; Ny=256;     % Define FOV and resolution

% Create 20 degree slice selection pulse and gradient
[rf, gz] = mr.makeSincPulse(20*pi/180,'Duration',4e-3,...
    'SliceThickness',5e-3,'apodization',0.5,'timeBwProduct',4);

% Define other gradients and ADC events
deltak=1/fov;
gx = mr.makeTrapezoid('x','FlatArea',Nx*deltak,'FlatTime',6.4e-3);
adc = mr.makeAdc(Nx,'Duration',gx.flatTime,'Delay',gx.riseTime);
gxPre = mr.makeTrapezoid('x','Area',-gx.area/2,'Duration',2e-3);
gzReph = mr.makeTrapezoid('z','Area',-gz.area/2,'Duration',2e-3);
phaseAreas = ((0:Ny-1)-Ny/2)*deltak;

% Calculate timing
delayTE=20e-3 - mr.calcDuration(gxPre) - mr.calcDuration(rf)/2 ...
    - mr.calcDuration(gx)/2;
delayTR=100e-3 - mr.calcDuration(gxPre) - mr.calcDuration(rf) ...
    - mr.calcDuration(gx) - delayTE;

% Loop over phase encodes and define sequence blocks
for i=1:Ny
    seq.addBlock(rf,gz);
    gyPre = mr.makeTrapezoid('y','Area',phaseAreas(i),'Duration',2e-3);
    seq.addBlock(gxPre,gyPre,gzReph);
    seq.addBlock(mr.makeDelay(delayTE));
    seq.addBlock(gx,adc);
    seq.addBlock(mr.makeDelay(delayTR))
end

seq.write('gre.seq')       % Write to pulseq file
```

Figure 2: A gradient-echo sequence defined in MATLAB with Pulseq toolbox functions. The resulting low-level sequence file is suitable to execute on any MR hardware platform equipped with an interpreter module.
110x93mm (300 x 300 DPI)

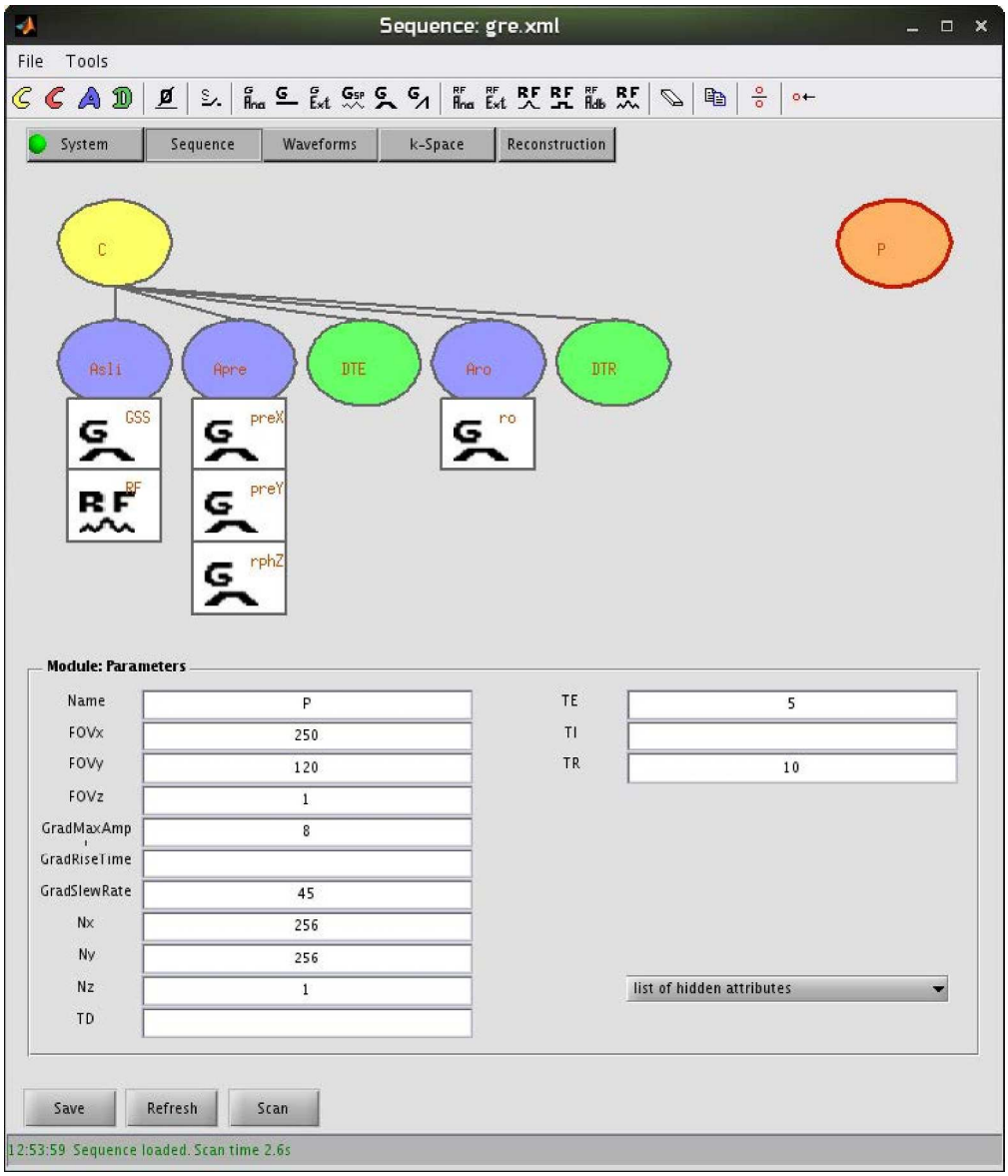

Figure 3: A screenshot of the modified JEMRIS graphical interface for sequence design. Sequences can be exported to a Pulseq low-level sequence file for execution on the scanner.
299x349mm (300 x 300 DPI)

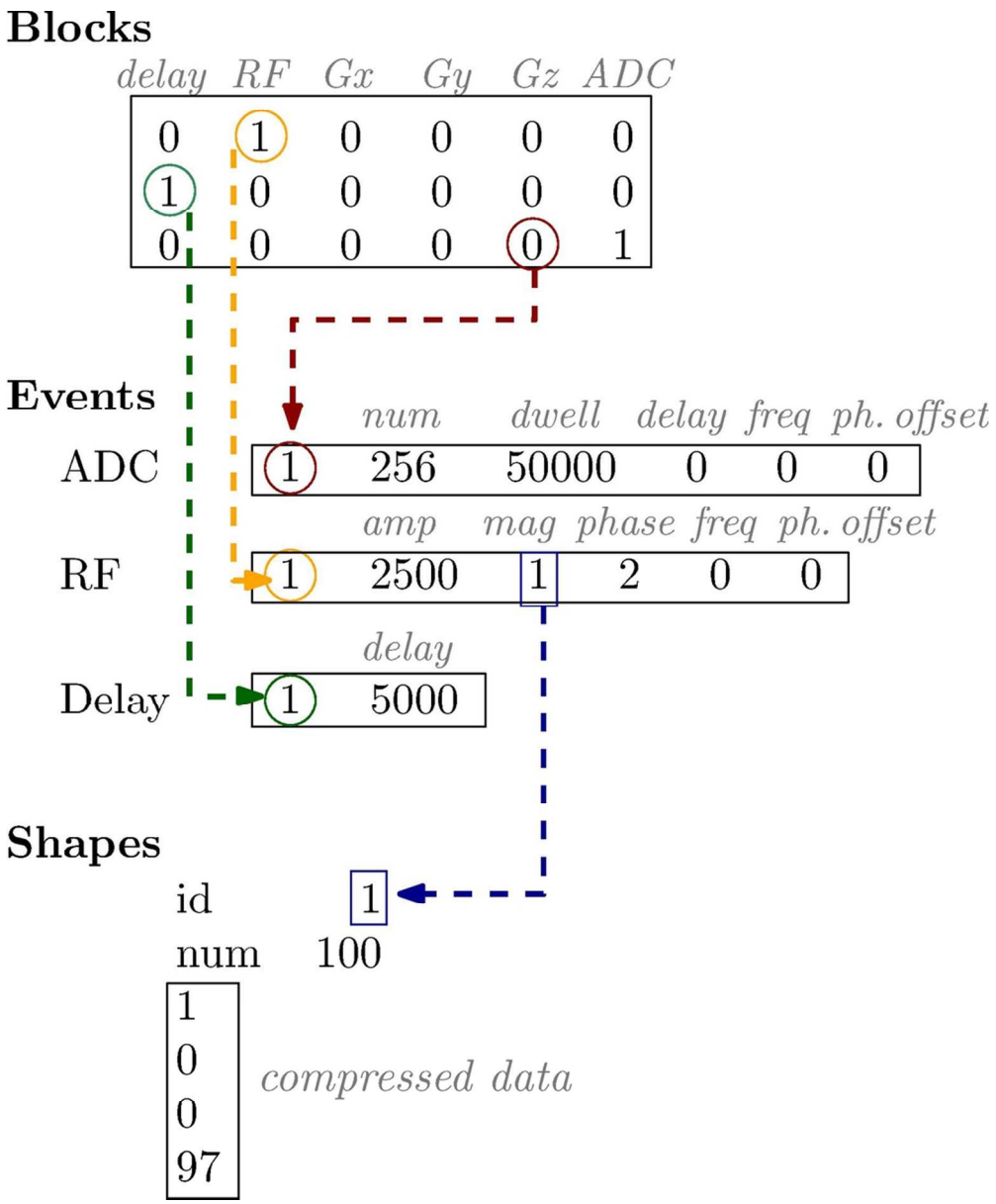

Figure 4: Main elements of the Pulseq hierarchical data format describing a simple FID. At the top level, the sequence consists of blocks, which contain integer IDs of sequence events. Sequence events may contain IDs of arbitrary shape objects to describe, for example, an RF pulse shape.
93x113mm (300 x 300 DPI)

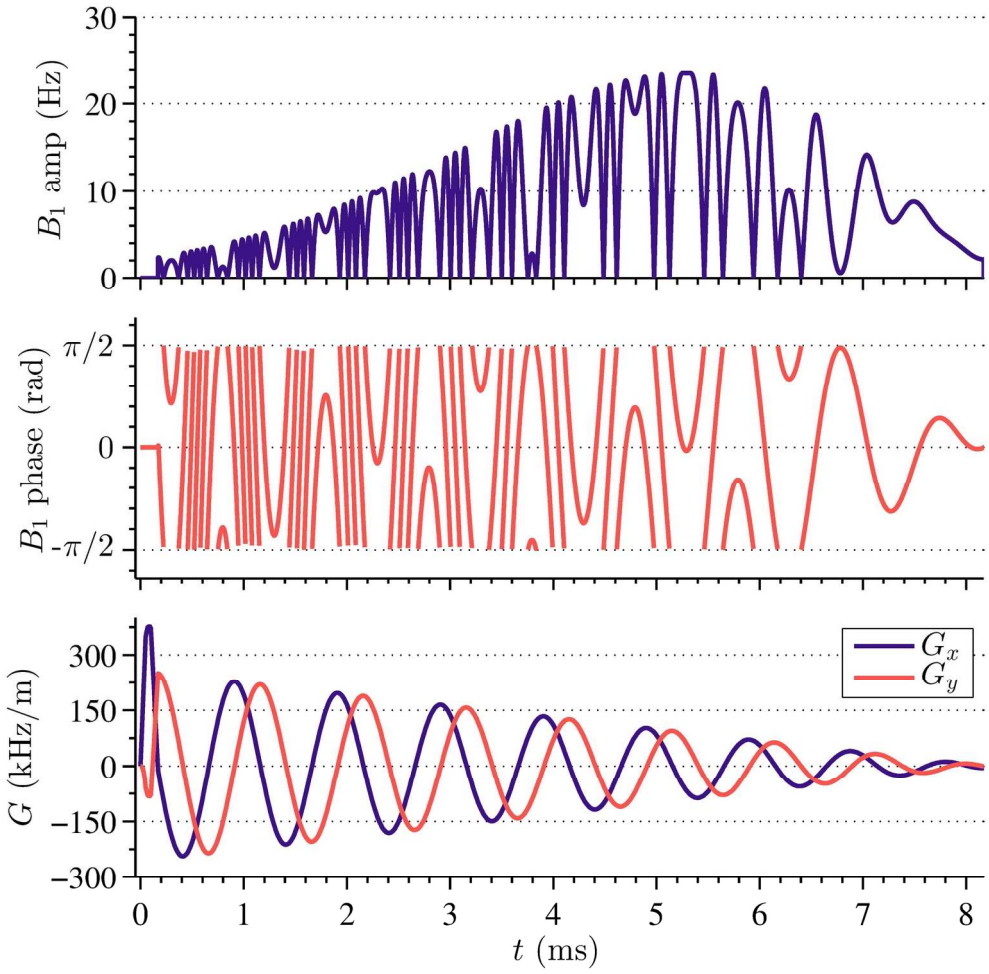

Figure 5: Gradient and RF pulse shapes to achieve 2D selective excitation. The (top) RF magnitude and (middle) RF phase combined with (bottom) a spiral trajectory excites two Gaussian cylinders as described in the text. Arbitrary pulse shapes are inherently supported in the proposed sequence format and interpreter modules.
161x158mm (300 x 300 DPI)

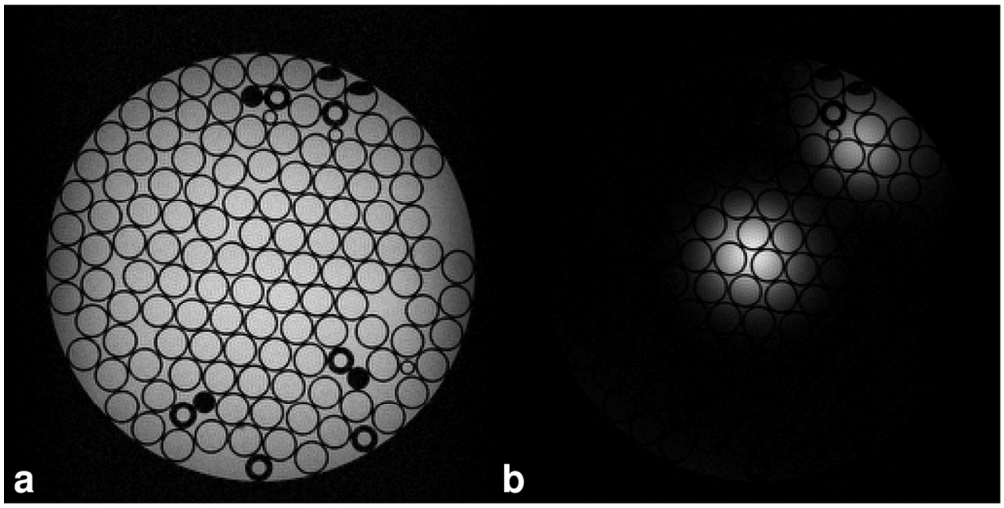

Figure 6: Image acquired directly from MATLAB for (a) the gradient-echo sequence shown in Fig. 2 and (b) a spin-echo sequence with 2D selective RF excitation. The Pulseq framework converts the MATLAB source code to hardware-dependent instructions to control the scanner.
111x56mm (300 x 300 DPI)

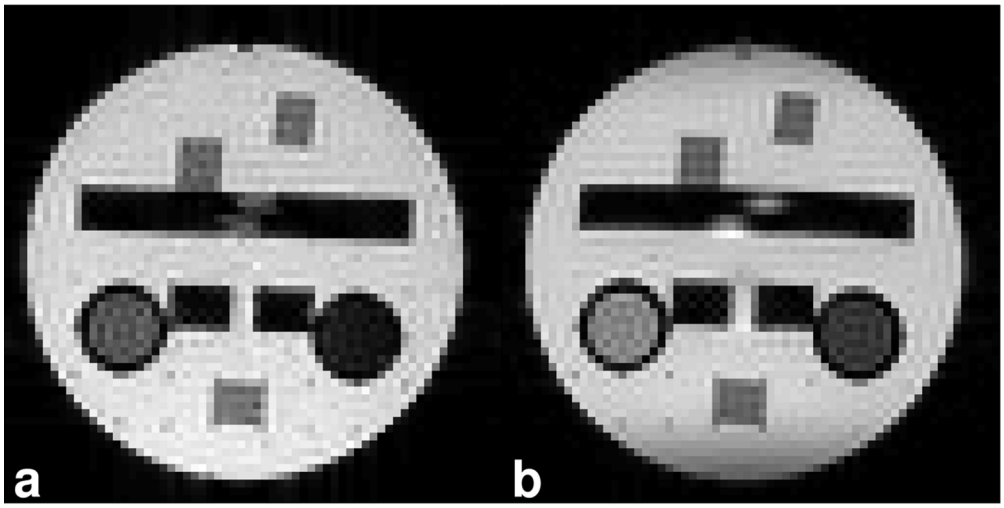

Figure 7: An axial image of a phantom (a) simulated with the Bloch equations and (b) acquired on a scanner. The same spin echo sequence file was used for simulation and measurement.
112x56mm (300 x 300 DPI)

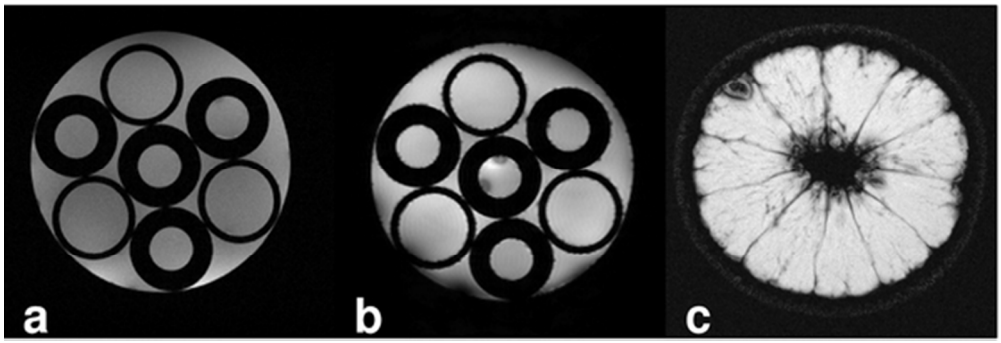

Figure 8: Images from the same sequence file executed on (a) Siemens, (b) Bruker and (c) GE hardware platforms. The sequence is stored in a novel platform-independent file and converted to hardware instructions by platform-specific interpreter modules.
50x17mm (300 x 300 DPI)