A Cross-Platform Editor and Simulator of Magnetic Resonance Imaging Sequences: Design and Implementation

P. Villacorta-Aylagas[1], I. Fernández-Arias[1], C. Castillo-Passi[2], P. Irarrazaval[2], F. Simmross-Wattenberg[1], M. Rodríguez-Cayetano[1], C. Alberola-López[1]

[1] Laboratorio de Procesado de Imagen. ETSI Telecomunicación. Universidad de Valladolid.
[2] Instituto de Ingeniería Biológica y Médica, Pontificia Universidad Católica de Chile.

Abstract

Simulation plays an important role in MRI research and training. Several simulators exist. Nevertheless, they do not have an easy interface to design the MRI sequence to be simulated. In this work, we developed an application that provides a graphical interface for that purpose. We have developed software with a graphical interface in which the user is allowed to build any arbitrary MRI sequence by arranging basic blocks such as gradients and RF pulses as well as to carry out a physics-informed simulation. This editor, implemented in Qt, makes use of KomaMRI.jl, an existing GPU-accelerated MRI simulator. We chose this simulator because it is the fastest. The main feature of our application is that it supports web-based technologies so that it can run either as a desktop application or in a browser. It also includes other functionalities, such as saving/loading sequences or the possibility of visualizing the excited slice in three dimensions. The results of its development have been satisfactory and promising for future work.

1 Introduction

Magnetic resonance imaging (MRI) is a non-invasive medical technique used for the diagnosis of pathologies related to soft tissues [1]. It is a versatile tool, so it has become the most appropriate choice in many clinical situations. However, it is not an easy task to operate with a MRI system and many factors contribute to these difficulties [2]. These obstacles, coupled with the high cost of real MRI systems as well as their high clinical demand, make the task of learning this technology an issue.

This motivation gave rise to MRI simulators, which are a breadth of tools that ease the understanding of all the elements involved in the process, and they also allow practitioners to carry out an unlimited number of uses. Since this idea emerged, several simulators have been described. One of these simulators is *KomaMRI.jl* [3]; this tool, developed in the Julia programming language [4], is meant to simulate general MRI scenarios that could arise in pulse sequence development. It has a GUI that allows for the visualization of the outcomes that arise from the simulation of a predefined sequence (reconstructed image, K-space, raw signal, pulses, etc.) . However, the tool lacks an actual sequence editor as well as a graphical tool to observe the selected slice. This, coupled with the fact that, to the best of our knowledge, no MRI sequence editors are available on web-based technologies, is the main motivation for this paper.

What we propose here is an MRI sequence editor with the following features:

- **Simple to use**, versatile and both education- and research-oriented purposes.
- **Cross-platform**. Hosted on a server, the application can be accessed from any web browser.

This article aims to describe the sequence editor creation process from an engineering software perspective. Albeit in the paper we focus on the desktop-application, the technologies used enable us, with almost no code modification, to embed it in a client-server application; detailed steps on how to accomplish the latter are also provided.

2 Background

Nowadays, there is a wide variety of tools available for MRI simulation. Some of them (like the ones described in [5] and [6]) were created with the aim of teaching theoretical MR concepts such as spin dephasing (Bloch equation) or quantum mechanics, but had no direct medical purposes [2]. Other simulators, for example, Virtual MRI [7], can emulate some of the main elements of the acquisition process in an actual MRI scanner. Torheim [8] simulator has a very similar behaviour to the one we have just mentioned, and shares the didactic purposes [2].

As for the research-oriented simulators, we should talk about JEMRIS [9], an open-source simulator focused, like the one here presented, on sequence development. It considers many properties of interest in MRI, to be as realistic as possible; however, it presents the drawback of only using CPU multi-threading. Other alternatives could be BlochSolver [10], MRISIMUL [11] (both closed-source) or MRILab [12], one of the most recent MRI simulators.

KomaMRI.jl [3] is a simulator intended to be fast, easy-to-use, extensible, open-source, and cross-platform. This simulator has been developed in Julia, a nice language for fast prototyping which includes precompiling and GPU support; the former should make *KomaMRI.jl* as efficient as other simulators written in C/C++ languages and the latter makes fast execution possible.

3 Analysis of requirements

The main functional requirements are:

- A set of basic building blocks for a MRI sequence (essentially, RF pulses and gradients) should be available and ready to use out of the box.
- The application should allow for the creation of arbitrary MRI sequences by arranging these basic blocks through simple drag & drop operations. Every block should be individually configurable and sequences should be passed to the simulator.
- There must be the option to generate groups of blocks. These groups, from the user's perspective, will be seen as blocks in themselves, so all the operations that can be performed on simple blocks (create, move, delete) must also be available on groups.
- The functionality of saving and loading sequences must be

ISBN: 978-84-09-45972-8, pags. 167- 170

included.
- The user must be able to define some global parameters for the simulation (intensity of the main magnetic field B_0, sampling period Δt, maximum gradient $Gmax$, etc.). Other acquisition parameters, such as TE (echo time), TR (repetition time) or ETL (echo train length) are implicitly defined by means of the sequence.
- The user interface should provide a view of the reconstructed image and its corresponding K-space, as well as a temporal diagram of the created sequence.
- A 3D-visualization of the excited slice should also be displayed.
- The simulation core may reside either on the local machine or on a remote one.

Regarding non-functional requirements, these include the development of a desktop application and its subsequent extension to a client-server web version, which shares the same code base, and thus is identical to the desktop version for the end user.

4 Design and implementation

4.1 Selected technologies

Once the requirements have been established, it is necessary to select the technologies to be used. As for the graphical user interface (this is where the user must define all the simulation parameters, including the sequence), Qt [13] is chosen. This C++ framework is oriented to cross-platform (desktop, web or even mobile) application development, so it seems the right option; specifically, Qt Quick, a module for interface development based on the QML language, has been used. The usefulness of QML mainly lies on the definition of the elements that make up the interface.

In addition, Qt permits compilation to WebAssembly [14], a type of low-level code that can be executed in modern browsers.

4.2 Implementation

4.2.1 Interface overview

Here we present the design, both visual and functional, of the interface itself. Based on all requirements previously seen, the final application appearance is shown in figure 1 in which, by means of panels, all the functionalities are included. The interface has been designed to be simple and intuitive.

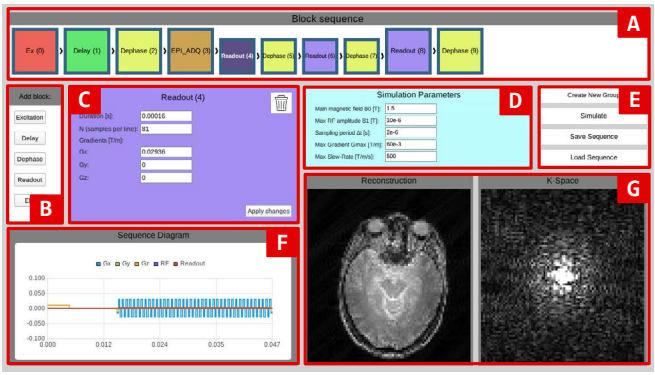

Figure 1. *Graphical User Interface sketch. Each panel is labelled with a letter used in the main text.*

The sequence panel (A) stands out by occupying the entire width of the application window; this is intended as a way to attract the user attention to the sequence, which is the main constituent of the whole process. In addition, it eases its complete visualization, without the need of trimming or drastically reducing the number of visible blocks; these blocks can be freely moved, modified and deleted. Panel (B) provides the library of available blocks, which can be added to the sequence by simply clicking one of the panel buttons. Panels (C) and (D) contain, respectively, the configurable block and the global parameters. As for the former, clicking on a given block will display its configuration menu from which related parameters can be easily set; as for the latter, global parameters are directly tunable. Panel (E) is intended for sequence management; simple operations such as sequence loading and saving are available. In addition, the user is allowed to create a block group; groups are intended for providing repetition capabilities without the need of explicitly writing the repeated blocks. The button to launch the simulation is also allocated here for convenience. Finally, the lowermost panels, namely, panels (F) and (G), are intended for visualization; specifically, panel (F) will display the sequence pulse diagram while panel (G) will show the simulation result, both in image and k spaces.

4.2.2 An overview of KomaMRI.jl

As stated above, KomaMRI.jl will be used for the simulation itself. This simulator presents three input arguments, which will be instances of the following structures:

- `Scanner`: this data structure contains the global parameters for the simulation, such as the main magnetic field strength B_0, the raster time Δt or the maximum slew-rate, among others.
- `Phantom`: it simulates what in a real scanner would be the patient's body (or part of the body), so it contains the data on which the sequence is applied. This data include the spatial location of the spins, T_1, T_2, T_2^* times and proton density ρ information, among others.
- `Sequence`: it contains the MRI sequence information itself. In the simulator, a `Sequence` object consists of:
  - A 2D array of `Grad` elements. One dimension (rows) is space (so that we can apply gradients in x, y and z) and the other (columns) is time. `Grad` structure mainly consists of two real scalars: `A`, corresponding to the gradient amplitude, and `T`, which refers to its time duration.
  - A 2D array of `RF` elements. As before, columns dimension refers to time; as for the other dimension, each row represents a different coil, so various RF pulses may be simultaneously applied. The `RF` structure is composed of these attributes: `A` is the amplitude of the RF pulse complex envelope, `T` corresponds to the pulse duration and Δf, included in latest versions, accounts for the difference in frequency with respect to the Larmor frequency of the RF pulse.
  - A 1D array of `ADC` elements, the dimension of which is time. `ADC` structure stands for data acquisition and contains information about when the readout is accomplished. It contains two values: `T`, which is the duration, and `N`, which accounts for the number of (evenly-spaced) samples to be acquired during time `T`.

4.2.3 Interfacing with KomaMRI.jl

We need functions from KomaMRI.jl to be called from the GUI and their input arguments to be in the correct format. To carry out this task, file `QtSlots.jl` has been added to the source directory of KomaMRI.jl. This file contains the `sim` function, which acts as an "intermediary" between the GUI and the simulator; it is responsible for adapting the data related to the sequence and the global parameters to a suitable format. Our solution has been to provide a `Sequence` element (as described in the previous section) for the former and a `Scanner` object —which is, in essence, a vector— for the latter. The third input argument is a `Phantom` object, also described above. This phantom is obtained, not from the GUI, but from a (`.nii`) file. Figure 2 provides a simple outline of all this.

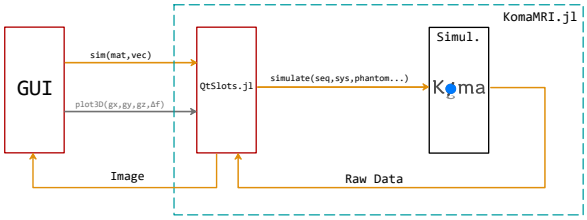

Figure 2. *Desktop-based application operating diagram*

In Qt/QML, sequence data is stored in a `ListModel` structure, which is nothing more than a container of definitions of the `ListElement` structure. Each `ListElement` instance contains information about a sequence block.

Once the `Simulate` button is pressed, the sequence information will be passed from QML to C++ by means of a Qt slot (see *Signals and Slots* in [13]). The next step is to call the Julia function, passing the sequence matrix and the global parameters vector (now, available in C++) as input arguments. The way to do this is by using a C/C++ API that allows embedding Julia (expressions, functions, variables...) in C++. Specifically, the following line of code is in charge of calling the –previously defined in C++– Julia function:

```
jl_array_t * output = (jl_array_t*)jl_call2(jl_sim, seq_arg, sys_arg)
```

Finally, a function called `plot3D` is defined in file `QtSlots.jl`. Its task is to show a 3D representation of the excited slice while displaying the entire 3D model by means of a volume rendering technique (see Figure 5). This functionality provides us with precise information of the slice selected for a choice of selection gradients and it is reactive in real time to the choice of these gradients from the GUI. The visualizer is implemented by using `PlotlyJS`, a Julia package oriented to the construction of graphs and diagrams, albeit with limited support for 3D rendering.

4.2.4 Technical steps for a client-server application

As previously stated, despite the code is reusable from a browser-based perspective, the application is intended to be deployed in an application server so a client-server architecture is mandatory. Hence, additional elements are needed. Figure 3 provides a detailed diagram of what these elements are. Specifically:

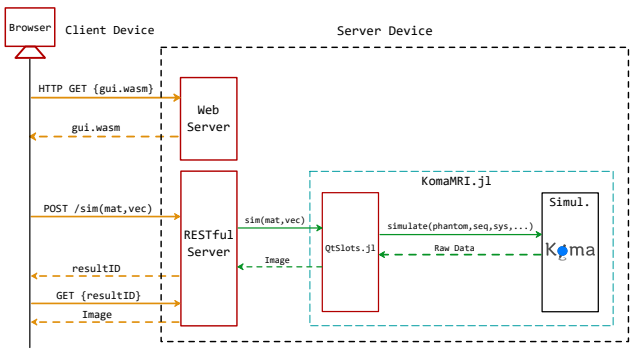

Figure 3. *Web-based application operating diagram. In this case, orange arrows represent commands of the protocol established between the client and the server (HTTP), while the green arrows correspond to Julia function calls.*

- **A web server**, capable of providing the client with the resulting files from the compilation to WebAssembly, which are, mainly, a (`.wasm`), a (`.html`) and a (`.js`) file. These three files will be loaded on the client's browser, so the user will be able to see the GUI and interact with it.
- **A REST-based (RESTful) server**: this server will run remotely-requested commands and will return their results. A RESTful web service works similarly to a web page, so we only need to send requests and receive their corresponding responses. It is created using Flask, a Python framework for creating web applications in a simple way. The RESTful server supports the following functions:
  - `POST/commands`: `POST` executes a new command and and saves its output in a temporary file whose identifier is by default returned in the Location header of the response. A JSON document with the fields `command` and `arguments` must be included in the request body.
  - `GET/commands/{resultID}`: `GET` returns the content of the file with local identifier `resultID` (the one obtained in the Location header of the `POST` response).
  - `DELETE/commands/{resultID}`: deletes the temporary file created as a result of the `POST` command.

5 Results and discussion

In this section we will show the results obtained from the execution of the program. We will also see the functional graphical interface, the 3D visualization tool and the current status of the web version.

5.1 Desktop version

As previously mentioned, Figure 1 showed the final appearance of the editor. In this particular case, an 81x81 pixel image was acquired by means of a GE-EPI sequence. The RF pulse was implemented by adding an `Ex` block; a delay block is then added to control TE. Then a `Dephase` block is added to shift the readout position to the lower-left corner of kspace. Then the `EPI_ADQ` block turns out to be a group, the contents of which (when expanded) are shown as smaller rectangles than regular blocks. The group consists of a two pairs of readout and dephase blocks; the first pair is a rightward readout plus an upward shift in the phase encoding direction; the second pair is a leftward readout plus the upward shift. This group is repeated forty times. The final readout line (to complete the 81 k space line reading) is carried out by the last `Readout` block (index number 8) which is not part of the group. The final `Dephase` block is not mandatory, since it simply returns the readout point to the k space origin.

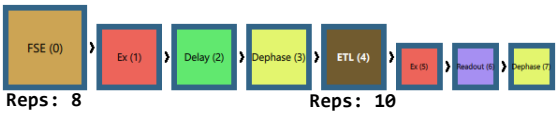

Figure 4. *Block hierarchy for an FSE sequence*

Other sequences are possible and they give us room to illustrate the block hierarchy that the GUI makes use of. Specifically, we will give rise to a Fast Spin-Echo (FSE) with 10 echoes per shot. The sequence has been built by two nested groups (see fig. 4). The outer group, referred to as `FSE`, starts with an excitation block generating the 90º RF pulse, followed by a `Delay` and a `Dephase` that allows the pointer to be placed at the initial position –the lower left corner of K-space, for example–. We set the number of repetitions of this group to 8, i.e., the sequence will consist of eight shots. The inner group referred to as `ETL` controls the readout positioning and the readout process itself; this group consists of an `Excitation` block that generates the 180º refocusing RF pulse followed by a –rightward– `Readout` block and a `Dephase` to move the pointer to the next line. Its number of repetitions —corresponding to the echo train length (ETL)— will be set to 10, so that a total of 80 lines will be read at the end of the simulation. The result is very similar to the one shown in figure 1.

5.2 3D Visualization

When clicking the `View 3D Model` button on the `EX` block configuration menu, a new window emerges. This window shows, simultaneously, the entire volume (on which the simulation is to be performed) and a plane through it, which represents the excited slice. Figure 5 shows the influence of the gradients activated during the excitation period (in our case, gradients in y and z).

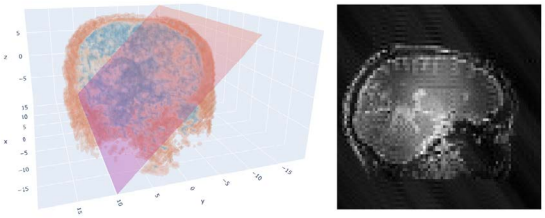

Figure 5. *3D visualization of the excited slice and its corresponding reconstruction*

5.3 Web browser version

Figure 6 shows the browser version of the GUI. In this case, the test was performed in Mozilla Firefox; other popular browsers such as Chrome, Edge or Safari, have also been tested. As previously stated, graphical functionality so far has been implemented with Julia packages. This is a limitation for a web-based application since these packages are not currently supported by WebAssembly. However, well-known graphical tools, such as VTK [15] and, specifically, vtk.js (a rendering Javascript library made for scientific visualization on the web), are off-the-shelf material that can readily circumvent this problem.

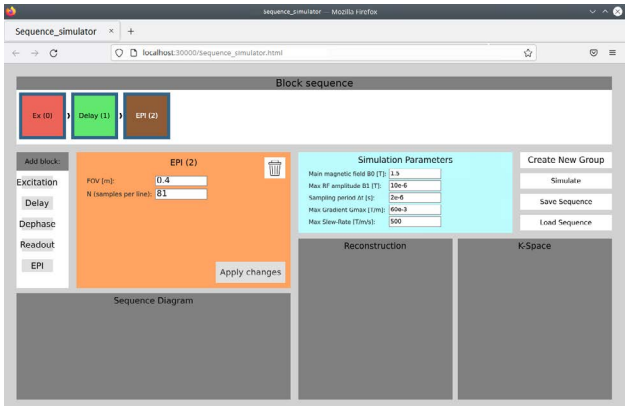

Figure 6. *Application running on a web browser*

6 Conclusions and future work

This paper has presented a MRI sequence editor which, combined with KomaMRI.jl simulator, allows the user to create any arbitrary sequence and to visualize its results. The tool is intended to be cross-platform and simple to use. Its design and implementation (GUI-Julia interaction, client-server architecture operation, 3D visualization...) has also been described.

The GUI provides practitioners with easy access to KomaMRI.jl that would otherwise require programming skills and profound knowledge of the Julia language. Despite this language is similar to other popular prototyping languages, such as Matlab and R, it has some specificities (such as the a rigorous control of the variables types and its consequences in related operations, such as operator overloading, to give an example) that makes the learning curve an issue. Hence, our GUI hides this complexity to the user to that they can concentrate on their interests, whether educational or research-oriented. In addition, the aim is to offer this tool as free software for the scientific community.

Despite in its current state the client-server version is in working progress, we have provided clear technical steps on how this will be accomplished and facilitating technologies —both in terms of communications and visualization— have also been identified. In addition, interfacing with OpenCL is also a short-term goal, so that a parallelized multidevice version of KomaMRI.jl can also be provided.

7 Acknowledgements

This work has been funded in part by grant PID2020-115339RB-I00, Science and Innovation Ministry, Spain. The Social Council of the University of Valladolid should also be acknowledged for promoting student initiation in research activities.

References

[1] Z.-P. Liang y P. C. Lauterbur, *Principles of magnetic resonance imaging: a signal processing perspective*. "The" Institute of Electrical and Electronics Engineers Press, 2000.

[2] D. Treceño-Fernández, J. Calabia-Del-Campo, M. L. Bote-Lorenzo, E. Gómez-Sánchez, R. de Luis-García, y C. Alberola-López, "A web-based educational magnetic resonance simulator: Design, implementation and testing," *Journal of medical systems*, vol. 44, no. 1, 12 2019.

[3] C. Castillo-Passi y P. Irarrazaval, "Komamri.jl," sep 2021. Available: https://doi.org/10.5281/zenodo.6627503

[4] J. Bezanson, A. Edelman, S. Karpinski, y V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[5] L. G. Hanson, "A graphical simulator for teaching basic and advanced mr imaging techniques," *RadioGraphics*, vol. 27, no. 6, pp. e27–e27, 2007.

[6] S. B. McKagan, K. K. Perkins, M. Dubson, C. Malley, S. Reid, R. LeMaster, y C. E. Wieman, "Developing and researching phet simulations for teaching quantum mechanics," *American Journal of Physics*, vol. 76, no. 4, pp. 406–417, 2008.

[7] T. Hackländer y H. Mertens, "Virtual mri: A pc-based simulation of a clinical mr scanner," *Academic radiology*, vol. 12, no. 1, pp. 85–96, 2005.

[8] G. Torheim, P. A. Rinck, R. A. Jones, y J. Kvaerness, "A simulator for teaching mr image contrast behavior," *Magnetic Resonance Materials in Physics, Biology and Medicine*, vol. 2, no. 4, pp. 515–522, 1994.

[9] T. Stöcker, K. Vahedipour, D. Pflugfelder, y N. J. Shah, "High-performance computing mri simulations," *Magnetic resonance in medicine*, vol. 64, no. 1, pp. 186–193, 2010.

[10] R. Kose y K. Kose, "Blochsolver: A gpu-optimized fast 3d mri simulator for experimentally compatible pulse sequences," *Journal of Magnetic Resonance*, vol. 281, pp. 51–65, 2017.

[11] C. G. Xanthis, I. E. Venetis, A. Chalkias, y A. H. Aletras, "Mrisimul: a gpu-based parallel approach to mri simulations," *IEEE Transactions on Medical Imaging*, vol. 33, no. 3, pp. 607–617, 2013.

[12] F. Liu, R. Kijowski, y W. F. Block, "Mrilab: performing fast 3d parallel mri numerical simulation on a simple pc," en *ISMRM Scientific Meeting & Exhibition*, vol. 2072, 2013.

[13] T. Q. Company, "Qt documentation," 2022. Available: https://doc.qt.io

[14] A. Rossberg, *WebAssembly Specification*. WebAssembly Community Group, 2022. Available: https://webassembly.github.io/spec/core

[15] W. Schroeder, K. Martin, L. Avila, y C. Law, *The VTK user's guide*. Kitware, 2010.