UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

DANIEL COSMO PIZETTA

PyMR: a framework for programming magnetic resonance systems

São Carlos

2018

DANIEL COSMO PIZETTA

PyMR: a framework for programming magnetic resonance systems

Thesis presented to the Graduate Program in Physics at the Instituto de Física de São Carlos, Universidade de São Paulo to obtain the degree of Doctor of Science.

Concentration area: Applied Physics
Option: Computational Physics

Advisor:
Prof. Dr. Alberto Tannús

Corrected Version

(Original version available on the Program Unit)

São Carlos

2018

AUTHORIZE THE REPRODUCTION AND DISSEMINATION OF TOTAL OR PARTIAL COPIES OF THIS DISSERTATION OR THESIS, BY CONVENTIONAL OR ELECTRONIC MEDIA FOR STUDY OR RESEARCH PURPOSE, SINCE IT IS REFERENCED.

Cataloguing data reviewed by the Library and Information Service of the IFSC, information supplied by Author.

Pizetta, Daniel Cosmo

PyMR: a framework for programing magnetic resonance systems / Daniel Cosmo Pizetta; advisor Alberto Tannús - São Carlos, 2018.

132 p.

Thesis (Doctorate - Graduate Program in Applied Physics) - Instituto de Física de São Carlos, Universidade de São Paulo - Brasil, 2018.

1. Magnetic resonance. 2. Pulse sequence. 3. Framework. 4. Python. 5. Spectrometer. I. Tannús, Alberto, advisor.

First, to billions of years of linked events that make me here and now.

The life's miracle; God's touch.

To my mom, Jandira S. Pizetta, for all her sacrifices to allow me to get here.

To my brother, Marcelo A. Pizetta, that helped me to fix an old radio, replacing all those cheerful and striped things, which today I call resistors, by wires. It did not work. Then, he encouraged me to course Radio and TV Electronics from *Instituto Universal Brasileiro*, when I was 10 years old, even though I did not know what it was a cubic root.

To my niece, Angeline A. F. Pizetta, to inspire her to always be dedicated to studying.

To my girlfriend, Geisiane R. da Silva, for the support, the care, and the pets - ALL of them.
Always by my side.

To my friends, for the support, for the help with the code, their company, the coffee or some conversation in a difficult time.

To my professors João G. Vieira and Gilsando C. Prudente for encouraging me, giving support in my desire to enter university.

ACKNOWLEDGEMENTS

To my advisor Prof. Ph.D. Alberto Tannús for all his shared knowledge and enviable enthusiasm that he brings to the projects. Also, to his patience and support in difficult times of my life.

To the technicians Ph.D. Edson L. G. Vidoto and Ph.D. Mateus J. Martins for all the immeasurable contributions to the projects.

All the coworkers that shared their knowledge and time, contributing and collaborating with the PyMR project as follows. Lucas C. R. da Silva for contributing mainly, but not only, with PyMR managers and extensions; Diego Turíbio integrating the PyMR plugin in Spyder; Maurício Falvo, implementing the geometry editor and image visualization; Christian C. Bones for the console and data processing development; Douglas H. T. Shimada in the data acquisition development; Pedro V. B. D de Souza in the peripheral manager and the spectrometer emulator. A special thanks to Maurício and Christian that helped me with figures and revisions.

To the partners and contributors in the MR Sprint free-version project. Victor H. M. Pessoa implementing the most of its functionalities; Prof. Ph.D. Fernando F. Paiva, Ph.D. Edson L. G. Vidoto, Prof. Ph.D. Daniel Papoti, Prof. Ph.D. Alberto Tannús, Camila Cardoso for their contributions.

To revisions and contributions by Paul P. Wang.

To all open-source software community that provides many amazing libraries.

To the Fine Instrument Technology Company, our partner in the hardware development, and its specific people Daniel M. Consalter and Agide G. Marassi for their contributions and provided data.

To the Institute of Physics of São Carlos and the University of São Paulo.

To the financial support by the National Institutes of Health (NIH) identified by U01EB025153 and U01EB025153, Schott Foundation, The Olson Foundation, MLDF, Juvenile Diabetes Research Foundation (JDRF).

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and Financiadora de Inovação e Pesquisa - Brasil (FINEP).

>>> **import this**

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one– and preferably only one –obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea – let's do more of those!

– Tim Peters, *The Zen of Python*

ABSTRACT

PIZETTA, D.C. **PyMR**: A framework for programming magnetic resonance systems. 2018. 132 p. Thesis (Doctor in Science), Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2018.

In recent years, the use of magnetic resonance technology has grown with advances in hardware, delivering accessible and small-size equipment and devices that open a range of new applications. Innovation in this field requires versatility and flexibility of both hardware and software. Despite the technological advances in the magnetic resonance hardware, the software still the most notable problem currently. This stagnation, delays progress that could reduce production costs and deliver faster development. Researchers in this field are unsatisfied with currently available options. In this panorama, we seek the enhancement of our specific framework for programming magnetic resonance systems, employing concepts from the areas of computing, engineering, and physics. This setup allows the software to merge different perceptions, causing it to be flexible and robust. We converged to Python and object-oriented programming to offer the Python Magnetic Resonance framework - PyMR. The PyMR includes graphical interfaces from templates that can be filled with data, requiring no programming. Our framework comprises other programming tools such as our plugin for the Spyder IDE, which creates the perfect environment to create systems and the pulse sequences. Also, a user-friendly magnetic resonance simulator MR SPRINT, derived from the PyMR structure, addresses educational use, exposing the whole experiment construction, setup, and visualization. Including, PyMR has been contributing to new challenging magnetic resonance systems, introducing modern concepts to change the actual scenario the researchers are facing when developing new magnetic resonance systems.

Keywords: Magnetic resonance. Pulse sequence. Framework. Python. Digital spectrometer.

RESUMO

PIZETTA, D.C. **PyMR**: Um framework para programação de sistemas de ressonância magnética 2018. 132 p. Tese (Doutorado em Ciências), Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Nos últimos anos, o uso da tecnologia de ressonância magnética cresceu com os avanços em hardware, fornecendo equipamentos e dispositivos acessíveis e de pequeno porte que abrem uma série de novas aplicações. Inovações neste campo requerem versatilidade e flexibilidade de hardware e software. Apesar dos avanços tecnológicos no hardware de ressonância magnética, o software ainda é um dos maiores problemas atualmente. Essa estagnação atrasa o progresso que poderia reduzir os custos de produção e proporcionar um desenvolvimento mais rápido. Além disso, pesquisadores neste campo estão insatisfeitos com as opções atualmente disponíveis. Com este panorama, buscamos o aprimoramento de nosso *framework* para programação de sistemas de ressonância magnética, empregando conceitos das áreas de computação, engenharia e física. Essa configuração permite que o software mescle visões de diferentes meios, fazendo com que a estrutura seja flexível e robusta. Nós convergimos, então, para a linguagem Python e programação orientada a objetos para oferecer o *framework* Python Magnetic Resonance - PyMR. O PyMR inclui interfaces gráficas a partir de modelos que podem ser preenchidos com dados, sem a necessidade de programação. Nossa estrutura compreende outras ferramentas de programação, como o nosso plugin para o Spyder IDE, que cria o ambiente perfeito para criar novos sistemas e sequências de pulsos. Além disso, um simulador de ressonância magnética de fácil utilização, MR SPRINT, derivado da estrutura PyMR, aborda o uso educacional, expondo toda a construção, configuração e visualização do experimento. O PyMR vem contribuindo para novos e desafiadores sistemas de ressonância magnética, introduzindo conceitos modernos para mudar o cenário atual que os pesquisadores estão enfrentando.

Palavras-chave: Ressonância Magnética. Sequência de pulsos. Framework. Python. Espectrômetro digital.

LIST OF FIGURES

Figure 2.1 - Demonstration of a nucleus precession.	23
Figure 2.2 - Three main stages of an experiment.	24
Figure 2.3 - Magnetic resonance relaxation parameters and its characteristic curves. .	25
Figure 2.4 - Magnetic resonance gradient system.	26
Figure 3.1 - DAMARIS framework in the console context.	30
Figure 3.2 - Pulse editor interface in ODIN.	31
Figure 3.3 - Sequence visualization interface in ODIN.	32
Figure 3.4 - Image visualization interface in ODIN.	33
Figure 3.5 - JEMRIS graphical interfaces.	34
Figure 3.6 - MRI Lab interface.	34
Figure 3.7 - TOPPE files example.	35
Figure 3.8 - Examples of Bruker TopSpin v2 and Paravision v5 software.	37
Figure 3.9 - Example of Philips Medical System Achieva v3 software.	38
Figure 4.1 - Diagram of literature-mapping requirements.	44
Figure 4.2 - Example of the class diagram for the amplifiers class.	45
Figure 4.3 - General M-CV model applied to PyMR's structure.	52
Figure 4.4 - The internal planned structure for PyMR CORE.	53
Figure 4.5 - The internal planned structure of PyMR GUI package.	55
Figure 4.6 - Framework's specialization diagram.	56
Figure 4.7 - PyMR framework and its relationship with other projects.	57
Figure 4.8 - Workflow diagram of the Scrum framework.	60
Figure 5.1 - Current package structure for PyMR framework.	66
Figure 5.2 - Class hierarchy for Coil and CoilSet peripherals.	66
Figure 5.3 - Class hierarchy for Magnet and MagnetSet peripherals.	67
Figure 5.4 - Class hierarchy for Shape and ShapeSet.	67
Figure 5.5 - The GUI for object creator with its view menu.	69
Figure 5.6 - The GUI for the object inspector with its menus.	70
Figure 5.7 - Graph view option for object inspector.	71

Figure 5.8 - The GUI for the object editor and its action menu.	72
Figure 5.9 - The template of a simple structure generated by Qt Design.	74
Figure 5.10 - Graphical template and its generated view.	75
Figure 5.11 - An example of a drag-and-drop edition for the sequence.	75
Figure 5.12 - Graphical interface for the <i>setup</i> vision.	77
Figure 5.13 - Graphical interface for the <i>control</i> vision.	77
Figure 5.14 - Graphical interface for the <i>fast</i> vision.	78
Figure 5.15 - Plot view for 2D data.	78
Figure 5.16 - Plot view for 3D data.	79
Figure 5.17 - Log view with messages for different levels of priority.	80
Figure 5.18 - PyMR Spyder Plugin installed and running in Spyder IDE.	81
Figure 5.19 - Configuration window for PyMR Spyder Plugin.	82
Figure 5.20 - Project menus and dialogs in PyMR Spyder Plugin.	82
Figure 5.21 - Sample context in the MR SPRINT.	84
Figure 5.22 - System context in the MR SPRINT.	84
Figure 5.23 - Sequence context in the MR SPRINT.	85
Figure 5.24 - Simulator context in the MR SPRINT.	87
Figure 5.25 - Processing context in the MR SPRINT.	88
Figure A.1 - Vancouver Symphony Orchestra with Bramwell Tovey.	103
Figure E.1 - Magnetic resonance icon package fundamental aspects.	115

LIST OF ABBREVIATIONS AND ACRONYMS

ADC	Analog To Digital Converter
API	Application Programming Interface
CIERMag	Centro De Imagens E Espectroscopia In Vivo Por Ressonância Magnética
CLI	Command Line Interface
CI	Continuous Integration
DAMARIS	Darmstadt magnetic resonance instrument software
DAC	Digital To Analog Converter
EMBRAPA	Empresa Brasileira De Pesquisa Agropecuária
FPGA	Field-Programmable Gate Array
FIT	Fine Instrument Technology
FID	Free Induction Decay
GAMMA	General approach To magnetic Resonance mathematical analysis
GUI	Graphical User Interface
GPU	Graphics Processor Unit
IDE	Integrated Development Environment
JEMRIS	Jülich extensible mri simulator
MR Sprint	Magnetic Resonance Experiment Simulator And Visualization Tool
MRI	Magnetic Resonance Imaging
MRS	Magnetic Resonance Spectroscopy
MCV	Model Control Vision
NMR	Nuclear Magnetic Resonance
ODIN	Object-Oriented Development Interface For Nmr
OOP	Object-Oriented Programming
PEP	Python Enhancement Proposals
RF	Radiofrequency
ROI	Region Of Interest
SPI	Serial
TOPPE	The end of pulse programming
UI	User Interface
UOP	User-Oriented Programming
VeSPA	Versatile Simulaton, Pulses And Analysis
VCS	Version Control System

CONTENTS

1	INTRODUCTION	21
1.1	Background	21
1.2	Scope	23
1.3	Outline	24
2	CONCEPTS OF MAGNETIC RESONANCE	25
2.1	Phenomenon and systems	25
2.2	Techniques and applications	28
2.2.1	<i>Relaxometry</i>	28
2.2.2	<i>Spectroscopy</i>	29
2.2.3	<i>Imaging</i>	29
2.3	Summary	30
3	REVIEW	31
3.1	Open and free software	31
3.2	Commercial software	38
3.3	Summary	40
4	DESIGN AND METHODS	43
4.1	Principles	43
4.1.1	<i>Programming paradigm</i>	44
4.1.2	<i>Software requirements specification</i>	44
4.1.3	<i>Interfaces</i>	47
4.2	Definitions	48
4.2.1	<i>Programming language</i>	48
4.2.2	<i>Documentation</i>	49
4.2.3	<i>Graphical items</i>	50
4.2.4	<i>File items</i>	51
4.2.5	<i>Integrated development environment</i>	52
4.3	Framework structure	53
4.3.1	<i>Model</i>	53
4.3.2	<i>Internal structure</i>	54

4.3.3	<i>External structure</i>	57
4.3.4	<i>Project relationships</i>	59
4.4	Auxiliary tools and techniques	60
4.4.1	<i>Logging and exceptions</i>	61
4.4.2	<i>Work management</i>	61
4.4.3	<i>Version control system</i>	62
4.4.4	<i>Integration system</i>	62
4.4.5	<i>Code metrics</i>	63
4.4.6	<i>Versioning and distribution</i>	63
4.5	Summary	64
5	IMPLEMENTATION AND RESULTS	65
5.1	General characteristics	65
5.2	PyMR framework	66
5.2.1	<i>Structure</i>	67
5.2.2	<i>Object manager</i>	70
5.2.3	<i>Vision manager</i>	75
5.2.4	<i>Data manager</i>	78
5.2.5	<i>Auxiliary views and dialogs</i>	80
5.3	PyMR Spyder Plugin	82
5.4	MR SPRINT	85
5.4.1	<i>Sample context</i>	85
5.4.2	<i>System context</i>	85
5.4.3	<i>Sequence context</i>	85
5.4.4	<i>Simulator context</i>	89
5.4.5	<i>Processing context</i>	89
6	CONCLUSION	91
6.1	Contributions	92
6.2	Perspectives	92
	APPENDIX A - ANALOGIES OF MAGNETIC RESONANCE SYSTEMS	105
	APPENDIX B - ACCESS CONTROL, INTEGRITY	109
	APPENDIX C - EXAMPLES OF DOCUMENTATION	111

APPENDIX D - EXAMPLE OF TUTORIAL	115
APPENDIX E - ICONS FOR MAGNETIC RESONANCE CONTEXT	117
APPENDIX F - DOCUMENTING PHYSICAL QUANTITIES	123
APPENDIX G - PROPOSAL FOR A NEW VARIABLE TYPE	127
APPENDIX H - PRODUCTION AND PROJECTS	129
APPENDIX I - OTHER CONTRIBUTIONS BY THE AUTHOR	131

1 INTRODUCTION

Please, allow us to start with a simple parallel example. Imagine buying a modern computer that can perform a high number of operations; it comes with a package to deal with the calculus, including a graphical interface. Up to now, we found that the computer is excellent in what it proposes to do, i.e., it is practical for ordinary usage. Then, instead of using the package that comes with it, we decide to develop a new algorithm, creating new code and executing the program inside that computer. At this moment, we realized it is impossible, or there is not a simple way to do that. Also, no documentation is available. Going ahead, imagine that we are creating a new revolutionary array processor and we want to test it in that machine, again, we cannot.

The example above elucidates the current problems the scientists and professionals in the magnetic resonance field are facing. Ordinarily, no system or tool available allows deep development, either in software or hardware. Their primary aim is to provide specific applications; those systems are not supposed to or do not support changes. As a consequence, discoveries, new applications, and the implementation of new technologies are troublesome and delayed. Thus, in the next sections, we will introduce the background of magnetic resonance field in retrospective, our project scope and the outline of this work.

1.1 Background

The nuclear magnetic resonance phenomenon, discovered in 40's,¹⁻³ describes magnetic properties of nuclei that in the presence of an external magnetic field, tend to align to it. When excited, applying radio waves, those nuclei are moved from the equilibrium. Then, returning to their equilibrium state, they oscillate also emitting radio frequency that induces an eletromotriz force in a perpendicular coil to the main field.^{1,3,4}

This signal is then acquired and processed, and the information about the nuclei and their environment can be extracted. The magnetic resonance technique main advantage is the fact that it is noninvasive and does not produce any harmful radiation over the object of study.⁵⁻⁷ Also, it can infer different information such as chemical structure, environment, mobility, and quantification depending on the experiment executed.^{5,8} Magnetic resonance techniques started to tread its path when first used for chemical analysis in the 50's.⁵ Then, in 80's with advances in imaging techniques, it started to be applied as a diagnostic tool in medicine for soft tissues.^{5,9-13}

From 2000's, with more improvements,¹⁴ new areas started employing and merging them such as neuroscience,⁵ functional studies,^{8,15,16} flux measures and heart imaging^{17,18} demanding high spatial resolution and fast measurements.¹⁹

However, the systems' high cost, between hundred thousand to a million of dollars, imposes limits to span their coverage, being impracticable in many clinical, laboratory and industrial facilities. The leading causes for the high costs are the magnet that, in general, use cryogenic fluids within the superconductor like helium, high-power peripherals, and spatial and structural requirements (about hundred of square meters in space for the equipment and tons of mass). Additionally, a significant number of suppliers discontinued many products because of their high development cost. Solutions for the previous hardware problems have been created and are still improving. We can mention superconducting materials at room temperature^{20,21} or not such low-temperature,²⁰ low power systems or systems that do not require strong magnetic fields.^{20,22} All those improvements decreased costs of production and operation to thousands of dollars, the size to square meter, and weight to hundred kilos.^{20,23}

As a consequence, those miniaturized systems introduced new applications, allowing magnetic resonance equipment to be used right in the field, instead of being an on-site modality.^{20,24} This approach delivered a highly skilled and versatile tool, which started to operate as a prototype in new areas such as petroleum,²⁵ agriculture,^{23,26,27} and even in the industry.²⁶ That evolution made many other companies increase their investments on a new generation of magnetic resonance systems with new applications in this marketing. Although all those advances, the commercial software, and the control system seems at low development rate or even stagnant and not boosted equally, keeping scientists limited by the current options. Difficulties when customizing, reusing and setting new requirements cause significant effort disposal - when it is possible to conduct such changes.²⁸⁻³¹ Furthermore, commercial systems target the final application that, in general, uses predefined methods and peripherals in such a black-box model.²⁸⁻³⁰

Those facts bring us back to the initial example in this chapter, in which we have a machine, but we cannot make changes on it. Alternative control software,^{29,30,32-36} simulators and processors,³⁷⁻⁴⁴ and educational tools^{36,43,45} from the literature have presented similar problems, turning the learning, development, and usage into an arduous process.

All exposed issues catch our attention and the attention of the whole magnetic resonance community to the research side.^{29,30} Modern hardware demand flexible alternatives to aggregate them to the systems,^{29,30,46} and many new magnetic resonance methods are

still waiting for a system capable of performing them.^{47,48} We can realize that new studies are required to improve the system's flexibility and usability, solutions that decrease the costs when developing new systems, promoting state-of-art techniques to contribute to a faster and straightforward evolution in this field.

1.2 Scope

The CIERMag (*Centro de Imagens e Espectroscopia in vivo por Ressonância Magnética*) group has been developing local technologies^{12,13} that aim to fill the gaps highlighted in the background section, some of them were already met. CIERMag previous work provides the basis for the new developments in this area when inserting the digital spectrometer, based on FPGA (Field-Programmable Gate Array) boards, allowing a flexible and high configurable hardware to deal with the magnetic resonance environment.^{46,49} On the software side, we provided the proof of concept when including an IDE (integrated development environment), a software that comprehends all necessary tools to deal with magnetic resonance, for both system, i.e. hardware, and pulse sequence development.^{28,50–52} Within this concept, all the items that were part of magnetic resonance software are handled as software projects, in which a first structure was proposed and tested.²⁸

Now, we will answer the question of how and what we can provide to offer a trustfull and easily reusable system, based on well-established techniques and pre-existent tools that are supportive and flexible to conceive the development of new technologies in the magnetic resonance field. We aim less-time-consuming, low-cost, and a resourceful system that spreads the magnetic resonance technology to new areas. Our goal is to create a dynamic magnetic resonance framework, which is a set of predefined and structured code capable of creating, configuring and controlling brand new systems. It accomplishes technologies primarily developed by CIERMag group and its partners. It must take into account the entire hierarchy of people involved at different levels such as programmers, researchers, students, and even ordinary users by fulfilling the requirements for each one. It culminates in the interaction between various areas of knowledge, such as computer science, physics, and engineering, that should benefit and feed our framework and its derived tools along with their structure, documentation, functionality, and usability. According, in the next section we provide an outline to guide the reader through this thesis.

1.3 Outline

As mentioned in the previous section, this thesis spans the concepts of computer science, electrical engineering, and physics. As a consequence, the chapters will not provide an in-depth discussion about each introduced topic. However, it will be noticed that many times, the concepts of each one will be introduced and then merged for a common goal, which mostly has never been used before, promoting a broad vision. We will start with a brief theory of magnetic resonance phenomena, the main techniques and system aspects that will be important to the future structure understanding. At this moment, concepts of physics and engineering will be introduced. Then, in the review, the focus will be on similar or related software tools in the magnetic resonance area. It helps us to achieve and fulfill all requirements for the proposed framework. Also, we will emphasize the pros and cons of each discussed tool, which includes sources from literature, educational and commercial kind. The methods chapter will rely on the choices, structure, and mechanisms that are in framework planning, implementation and some that should be adopted along with the framework usage. If any specific techniques are used, they will be explained. In the results chapter, we will focus on the framework implementation and what we make available to the users. Then, examples and measures will be exposed. Finally, we will discuss and compare our tools with those pre existents, presented on the review. Included to that, we will underline our contributions and future perspectives. As a complementary material, this thesis includes some analogies, diagrams, examples, proposals, production and contributions lists. All of them can clarify, illustrate or make more explicit the development and contributions we have made.

2 CONCEPTS OF MAGNETIC RESONANCE

The theory chapter will cover the basics of magnetic resonance phenomena with a short presentation about the main derived techniques and their applications: relaxometry, spectroscopy, and imaging. Then, the focus will be on the magnetic resonance system composition, operation, and processes for each of the previous techniques. The content will allow the reader to follow the ideas that will be discussed in the subsequent chapters.

2.1 Phenomenon and systems

The magnetic resonance phenomenon is based on intrinsic properties of the atom nuclei that interact with electromagnetic fields: the spin or intrinsic angular momentum, I , and the permanent magnetic moment, μ , which we can consider a little magnet. The relation between both in the nucleus is given by $\mu = \gamma I$, where γ is called gyromagnetic ratio. The gyromagnetic ratio for the 1H nucleus is expressed as $\approx 42.578 MHz/T$. As the frequency is around megahertz, we call that range of radiofrequency (RF), so we can relate it further with RF pulses. In a presence of a external magnetic field B_0 it would be produced a torque which will make it precess around B_0 direction. The precession frequency is the Larmor frequency, given by $\omega = \gamma B_0$, which depends on the nucleus characteristic value of γ and the magnetic field intensity in its place. This first depiction is given by the Figure 2.1.

Figure 2.1 – Demonstration of a nucleus precession around B_0 . The magnetic moment μ , in the presence of an external magnetic field B_0 creates a torque that continuously changes the direction of the magnetic moment, generating the precession.

Source: By the author.

In a macroscopic vision, a classical vision, a large number of nuclei are contributing to what is called net magnetization M_0 , initially is pointing to the direction of B_0 , which is Z in this example. In this first stage, the total magnetization M has only a contribution of longitudinal magnetization, named M_L . If we excite the nuclei with a oscillating magnetic field

with its frequency matching the Larmor frequency, the nuclei will resonate, and then we can change the direction of the total magnetization. If the net magnetization goes to the plane XY , this is called a 90-degree pulse. The magnetization in the XY plane is called transversal magnetization, denoted by M_T . Then, letting the system return to equilibrium, i.e., turning off the RF excitation, the net magnetization will precess back to the initial position.^{1,53–55} Those steps are elucidated by the Figure 2.2.

Figure 2.2 – Three main stages of an experiment. Starting at equilibrium, then the excitation is made using an RF pulse that rotates the magnetization to the XY plane, and then the relaxation processes take place. Greyish lines indicates the laboratory frame of reference view.

Source: By the author.

As the magnetization can be considered a small magnet, if a coil is placed perpendicular to the XY plane normal, will be induced an emf (electromotive force) that will decrease as the magnetization returns to equilibrium and it will represent a signal in the system. This signal is called free induction decay (FID). In the perfect sample, the amplitude of the signal is proportional to the number of nuclei (density), among other factors, and the frequency is the Larmor frequency.^{1,53–55} Applying the Fourier formalism, we can get the information of the right frequency the nuclei are precessing. In this process we can observe different resonant frequencies that is related to different chemical environments the nuclei experience.^{1,53–55} That is the basis for the spectroscopy technique. More factors are processed in spectroscopy, but it will not be discussed because it is out of our scope.

Up to now we already get the signal. Now we will introduce the mechanisms that make the magnetization return to the equilibrium, decaying, what is called relaxation. The longitudinal relaxation is related to the time parameter T_1 , describing a time variable that is related to the energy exchange, or thermal energy.^{1,53–55} The transversal relaxation is related to the random phase changes in the local, losing phase coherence (little arrows in the figure), and it has associated the time parameter T_2 , being irreversible. Small inhomogeneities in the local static magnetic field will accelerate the process of dephasing, and the time parameter associated with it is named T_2^* , in which part of this process is reversible, due to its conditions. The Figure 2.3 represents all those three values, also giving the formula for each magnitude. Note that in

this case is not represented the oscillation, because the coordinates are in the rotating frame of reference. Thus, we only see the decay. The reader should also note the highly difference between the time values for T_1 and T_2 . Each time parameter can bring information about the substance in analysis; it leads us to the relaxometry technique.^{1,53-55}

Figure 2.3 – Magnetic resonance relaxation parameters and its characteristic curves. Note the value for T_1 be higher than both T_2 and T_2^* . Greysh lines indicates the laboratory frame of reference view.

Source: By the author.

As we have already seen, the precession frequency is proportional to the magnetic field, if we produce a controlled linear magnetic field variation along with one direction, the excitation frequency should be different to different spatial positions. Thus, this is the function of the magnetic field gradient, called ordinarily just gradient. In the Figure 2.4 we show in the left side, a gradient in the Z direction. This gradient is called selection gradient because using it with the match selective RF pulse one can select a slice in the sample in which all the disposal nuclei will be with the same phase in the transversal magnetization. Now, if we apply another gradient, but along with the Y direction for an instant of time, it will make each line of that slice change its phase, so this gradient is called phase encoding. Finally, inserting one more, in the X direction, called the readout gradient, we make that each column of that slice could contribute to the signal with different frequencies, then we call the frequency encoding gradient. As we can encode each position of the slice, we can then decode and know the original position. Applying the 2D Fourier formalism, we can decode that space, called K-space.

Figure 2.4 – Magnetic resonance gradient system. Each component of the gradient encoding one position direction space into K space. Then, the information can be recovered.

Source: By the author.

2.2 Techniques and applications

The nuclear magnetic resonance[†] has three main distinct techniques: relaxometry, spectroscopy, and imaging. As seen in the *Phenomenon and systems*, all of them shares the same principles and characteristics, but aiming at different objectives. The system components also change along with the technique, requiring more or fewer items, quality, technology, power, and so on. Basically, for all of them, when increasing the resolution will increase all the values cited, which make the systems increase their price too.

2.2.1 Relaxometry

Relaxometry is based on relaxation parameters to infer physical and chemicals properties for the material of study. Relaxation parameters T_1 and T_2 , respectively, transversal and longitudinal relaxations, refers to the signal decay over time.⁴ T_1 reflects the nucleus' natural relaxation in the system, while T_2 relates to the net or spin-spin relaxation. Both are the keys in relaxometry and intrinsic from the system, but they are not the only ones. Other parameters, even being fundamentally derived from the previous ones, take more effects into account, such as T_2^* that depends on the local magnetic field inhomogeneities. Relaxometry methods are suitable for ubiquitous scientific fields and industrial applications. Simple examples are to infer about the quantity of a specific component in the sample, e.g., amount of sugar in the

[†] The term Nuclear was removed from the imaging technique to avoid frightening people addressing harmful radiation.

fruit or oil in the seed.^{23,26,27} Those applications are practically new in the industry, because of the development of new low-cost magnets and fast pulse sequences. Relaxometry methods can assist other techniques such as magnetic resonance imaging and spectroscopy. Several medical applications use relaxometry for complementary information, such as iron quantification in the liver, diagnoses for multiple sclerosis and tumor identification.⁵⁶

2.2.2 Spectroscopy

Nuclear magnetic resonance spectroscopy (NMR) may be the most used magnetic resonance technique until some decades ago.⁴ Chemistry and biochemistry are the leading areas employing NMR to chemical analysis. In the NMR methods, frequency and phase that compose the signal are handled to extract structural information from sample components, not relying only on relaxation constants.⁴ Ordinarily, Fourier formalism is employed to process data that results in the NMR spectra, which names the method. MRS requires high quality and precise equipment to deliver good spectral resolution, which, of course, is expensive, compared to the relaxometry dedicated ones.⁴ Lately, as well as quoted in relaxometry, magnetic resonance imaging techniques were merged with NMR methods, which led us to magnetic resonance spectroscopy localized (MRS).⁵⁷ This fusion allows selecting a voxel in the acquired image, in which an MRS method will be executed. In many cases, the MRS can give us information about proteins, nucleic acids, and carbohydrates. Those are just some examples of MRS usage that are invaluable for extensive and accurate chemical analysis in a large number of applications such as in medical and chemical industry mainly.

2.2.3 Imaging

The most well-known application for magnetic resonance imaging techniques is the medical diagnosis of soft tissues that has more hydrogen (water) nuclei available.⁵ In the MRI technique, the images can be created from the 2D or 3D acquired signals, which after a Fourier Transform they represent the spatial density of nuclei contrasted by T_1 and T_2 . The gradients are used in the system to select what part will be analyzed.^{9,10} Although MRI is extremely useful in the medical field, many other areas have contributions from it. Material sciences use it to acquire images inside porous media. Flux and its dynamics can also be studied using MRI. As the previous techniques, MRI can be combined with other ones to extract more information.^{5,8} Spectroscopy and Relaxometry can be employed,⁵⁷ generating methods known as voxel spectroscopy and magnetic resonance fingerprint⁵⁶ to analyze complex changes in the tissue.^{5,9,58,59}

2.3 Summary

In this chapter, we provided a short basis for the understanding of the magnetic resonance phenomena and the central techniques in this field, such as relaxometry, spectroscopy and imaging. All magnetic resonance techniques have been highly valuable in the scientific research, chemical and medical applications and since then, employed to new areas of study such as oil, agriculture, material engineering, and so forth. Additionally, gave an overview about the structure, characteristics, and relationships between the system components and their processes. All addressed concerns will guide the reader through the revised literature, methods and implementation of our system. If any doubt about the system composition and functionality still affront the reader, this thesis includes some illustrative analogies, presented in *Appendix A - Analogies of magnetic resonance systems*.

3 REVIEW

Due to this thesis multidisciplinary character, the review chapter will concern the magnetic resonance field, pointing software-related issues and their description. Items described here will give the reader an overview of what is currently available, including literature and commercial sources. Also, beneficial characteristics will be highlighted as they can contribute to our implementation.

3.1 Open and free software

In this section, we will start talking about the main tools and simulators available in the magnetic resonance field, as publications and public source code are easily available than commercial options in the question of depth of information. Focused on spectroscopy, VeSPA (Versatile Simulation, Pulses, and Analysis)⁶⁰ is an application suite specializing in RF pulses, simulation and spectrum analysis. Written in Python, VeSPA is a set of tools for pulse design, spectral simulation, simulated data and data processing, and analysis. It works with GAMMA,^{38,61} used in simulation and mathematical analysis in spectroscopy systems. Written in C++ it has a wrapper providing almost all the existing functionality within Python interface. This library provides data commonly used in the formulation of magnetic resonance problems such as spins systems, RF and gradients pulses. Both have a high potential to be integrated into our code in the future. Noticing the use of Python in both is essential.

Another tool for processing is NMRPipe,⁶² based on the UNIX pipe. The pipe system allows programs or scripts to share information through streaming data, which can be applied in the distributed format. Although interesting, the application already restricts the use to Linux machines, and or for those who do not have the familiarity with a shell script.⁶²

As a way to simplify the use of the UNIX pipe format we found NMRGlue^{39,63} that also includes other tools. Among other functions, it is possible to convert data between different types of commercial vendor format using NMRGlue. In comparison to the previous one, NMRGlue provides a simplification in programming by using the Python language, as well as keeping the API in a more user-friendly format, including excellent documentation. Although it has several good points, it is based on functions and almost all their names are nomenclature with jargon in the area. This naming template is deprecated for readability purposes.

Entering in a more complex system that also provides graphical interfaces, there is a tool called DAMARIS^{30,64}, which is a framework for controlling generalized magnetic resonance equipment, as shown in Figure 3.1 with its main window opened. It does not deal with imaging techniques nor image visualization, remaining the use for spectroscopy techniques.^{30,64} It supports pulse sequences written in Python language, what is an exciting feature as we will see later in the next chapters. DAMARIS provides a simple console to set up, execute and control the experiment from the computer. Then, the acquired data can be processed and visualized. The graphical user interface is not as sophisticated as demanded in most applications.

Figure 3.1 – DAMARIS framework in the console context. A minimalist interface to set up the experiment, process and visualize data.

Source: From DAMARIS website [64].

DAMARIS' most important aspect is the separation between front-end and back-end.^{30,64} The first provides a generic API that allows the user program and reuse the code to another equipment. Then, the back-end is what communicates directly with the hardware or other dedicate software.^{30,64} This separation is crucial in simplifying problems and generalizing items offered by the API, being desirable to be implemented in our system. DAMARIS proposal for its back-end is yet more deep by translating the front-end's program to allow the use of commercial equipment. Despite being an attractive feature, it involves massive, time-consuming development and legal issues, which scapes our short-term goals.

Focused into the development of new sequences and simulation study, ODIN^{29,65} uses object-oriented paradigm in order to allow simple code reuse and writing. However, adopting the C++ language as the base programming language, it makes learning for beginners very

difficult.^{29,32} Also, it uses short naming, what was a tradition in old programming techniques, but nowadays the readability is most valuable. The advantage is that C++ can be easily integrated with Python enabling the generation of wrappers and integration with our system in future development. ODIN has a graphical interface in which it is possible to create shapes of RF pulses in Figure 3.2, trajectories in K space (gradient pulses), visualization of MR sequences as shown as Figure 3.3 and magnetization simulation as in Figure 3.4. All of those characteristics make it a complete tool. However, even with several functionalities, its graphical interface is precarious and inflexible,^{29,65} easily noticed by the figures cited. Such contexts in the figures should be detachable to better positioning or maximization and graphics view. Group boxes are overused in the whole interface, for not the purpose of grouping objects and their titles has mixed styles (acronyms, underscores, title case). The graphs scale can be rounded to improve the use the space and also not mixed with controls. Also, sliders are not recommended in mouse interaction interfaces, they are better for touch screens, so it can be changed to spin boxes. However, we can consider that the controls have units, what helps the user.

Figure 3.2 – Pulse editor interface in ODIN. This is the interface for the pulse editor (RF and gradients) which accomplishes the sample and the magnetization setup and preview.

Source: ODIN website [65].

Another interesting tool is named JEMRIS,^{40,41,66} Figure 3.5, which is a multipurpose

Figure 3.3 – Sequence visualization interface in ODIN. Buttons have mixed styles, icons and text, follow uncommon order (print, save and close in the middle of toolbox instead of at begin). Graph scales can be improved and colors can be used, at least as an option.

Source: ODIN website [65].

simulator, also used for didactic purposes in advanced courses. This simulator can handle arbitrary RF pulses as well as hardware items such as multi-channel receiver and transmitter coils. Other essential items in modern techniques such as non-linear gradients and field inhomogeneity are covered. It uses parallel data processing in MPI and it is written in C++, with the graphical interface in MatLab,⁶⁷ Its core capabilities are widely and bring many perspectives to our system. Although the intention of using a MatLab graphical interface is to have a smooth integration with that one, it is not practical for other purposes because of graphical limitations of this toolkit.

Recently released, MRI Lab,^{43,68} is a fantastic complete tool that allows the user to create the whole experiment^{43,68} for simulation purposes. It has samples, phantoms, selection of magnetic field map, different types of coils.^{43,68} MRI Lab project has a similar internal structure of our framework, which bring hope for future integration. Some negative points are the particular complex interface, and the fact that it is based on Matlab, as we already said, limit others applications, Figure 3.6. MRI Lab comprehends most of the features we were trying to provide with MR Sprint (will be discussed in results), that at this point is being integrated into our framework. Although MRI Lab is a complete and functional MRI simulator, it aims practical

Figure 3.4 – Image visualization interface in ODIN. Problems with graph scales, oversused groupbox and positioning are seen again, such as the previous figures.

Source: ODIN website [65].

operation, which includes interfaces of typical commercial systems - not for basic students. We will discuss the differences better in *Implementation and results*.

With a perspective of being used to program the sequence in the magnetic resonance context, many recently tools were published. TOPPE^{35,69}, SequenceTree³² and Pulseseq^{33,34} are some examples. We show the TOPPE main files in Figure 3.7. Those are some cases that creating a new format to describe a sequence could facilitate the work of the sequence programmer, but at the same time, it introduces difficulties in debugging, and to connect to the main other services. They use description files and graphics in substitution to the programming language, which may work for simple sequences. They were planned to be integrated into the current commercial equipment, along with the production of a converter or a type of compiler, which is not promptly available for all vendors. Their layers, and ideas in the files share some principles with our structure, like the division presented in the chapter *Concepts of magnetic resonance*. We can exploit concepts and ideas provided because they provide a vision from the current sequence programmer.

In the simulation area, SIMRI^{37,70,71} is specialized in effects of T_2^* , which is generated

Figure 3.5 – JEMRIS graphical interfaces. a) Coil edition and simulation. b) Graphical sequence design. JEMRIS has a more didactic sequence plot, in which we can see each module with its functionality in the system. The lack of units in the interface is the biggest problem. The positioning and use of groupboxes are better if compared to ODIN interface, but still a precarious interface.

Source: Adapted from JEMRIS website [66].

Figure 3.6 – MRI Lab interface. a) Parameter setting window. b) Sequence setting window. It is a complete simulator tool for MRI that can simulate a complete experiment. However, the interface not accomplishes the units for values, it uses acronym the no deatchable windows also.

Source: Adapted from MRI Lab website [68].

Figure 3.7 – TOPPE files example. TOPPE is a sequence programmer based on files, in which each file represents a module that will be executed. It is difficult to say how to program or interact with it.

Source: From the TOPPE website [69].

by magnetic field inhomogeneities, aiming imaging applications. It is written in C programming language, but it already has a Python wrapper that simplifies its use. It also includes a graphics system for visualization and simulation of spins in 1D for educational purposes, again, assisting T_2^* artifacts. Continuing in the simulation goal, the Virtual MRI,⁴² show a simplified vision of a simulator for basic simulation of magnetic resonance. It is also similar to our proposal for MR Sprint and the previous MRI Lab,^{43,68} which will be discussed later. Its main language is Java; thus the integration should not be smooth with Python. On the other hand, it can be easily integrated into browser systems. For more advanced applications in magnetic resonance data simulating, Bloch Solver⁴⁴ brings the concept of the use of Graphical Processor Unit (GPU) to process data faster.

The majority amount of tools mentioned here, bring us ideas of structure, interfaces, pros and cons that match with our current project. Also, they contribute to a vast number of visions from different areas to our framework interface - programming or graphical. Besides, most of them are C, C++, and Python-based, which we will see in *Design and methods* they interface well with Python. Odin, DAMARIS and JEMRIS are our closest application at this point. Additionally, we can see that there are no tools to assist in developing and the documentation, in general, is not enough, code structure is hard to learn. The inclusion of jargons and abbreviated words are not recommended. We call the attention to the fact that none of them deal in the interface with the physical unit, which will be our concern to avoid errors and also facilitating the readability for the user. Also, the sequence description based in entirely new formats does not help when using tools such as text editor with code highlight and wizards, as they must be implemented and distributed together, also generating time-consuming maintenance.

3.2 Commercial software

In this section, we will discuss two examples of commercial software for controlling magnetic resonance machines. As noticed in survey applied in 2013,²⁸ commercial software are practical for their purpose in most cases, but they are very restricted to modifications, even for scientific purposes. Furthermore, the target of commercial systems is the final application that, in general, uses predefined methods in the equipment in a black-box model.²⁸ An advantage is that practical applications extensively use those systems,²⁸ which should guide us with standard practices. Additionally, current marketing supplies seek the prerogative of sailing services, culminating almost by the end of development by the scientific community using commercial systems.³⁰ In general, none of them provide a highly assistive tool for development of new pulse sequences or systems. The closest solution to our proposal is available for internal use only as described by Debbins.⁷²

Our first example of a commercial tool is the Bruker (TopSpin and Paravision) system. As seen in Figure 3.8, those interfaces shows a highly condensed and challenging navigating interface. Parameters that represent physical quantities do not have the unit in most cases.[†] The icons and the graphical items disposal are hard to interact even for technicians. Making advanced changes in that system require the user to change text files for configurations, many of them for instance, without any support.[†] However, they are told to be the most flexible for making changes.

Our second example is a Philips system, Figure 3.9, which have a slightly more modern graphical interface. However, they are declared[†] one of most hard-coded sequence design programming, raising many objections when programming them.²⁸ This case shows us the medical-application like interface, which has less advanced attributes in the interface.

The reader can notice that all the tools exposed so far have similar characteristics whether their structure, the graphical interface or the way each one interacts with the user. Thus, the problems are still the same for both open and commercial software, being the last one more closed and unchangeable.

[†] Recurring information given by technical users.

(a)

(c)

(b)

(d)

Figure 3.8 – Examples of Bruker TopSpin v2 and Paravision v5 software. a) Spectrometer control tool. b) System control logging. c) Scan control. d) TopSpin control interface. Those images show how complex and challenging is to navigate and to configure such a machine. Advanced modifications, when possible, need to be done right in the configuration files. This system is more developer alike.

Source: Screenshot by the author.

Figure 3.9 – Example of Philips Medical System Achieva v3 software. a) Image view interface. b) Parameter tab setup for geometry. c) Parameters tab setup for contrast. This system presents a better and more simplified graphical interface. However, it is much more difficult to implement any changes, as they are targeted to medical uses.

Source: Screenshot by the author.

3.3 Summary

In our review, it was shown that magnetic resonance dedicated applications had not been continuously updated, exposing many issues in the interfaces. Techniques that could have improved the software structure, usage, appearance, and interaction were not adequately employed, not followed or they are faulty in those systems. The majority of available tools are implemented by non-programmers which can produce a new and vital point-of-view, but it also inserts conceptual programming issues.

The open-source and non-commercial software gives us several examples of great tools, which are extremely valuable for researchers. Whereas, simulators and data processors, are usually presented isolated and focused on particular issues. Also, problems in the graphical interface, code structure, and restrictions when installing and using them are numerous. In this case, allowing our proposal to be future integrated with them should benefit both sides. Educational implementations have similar issues, causing misleading interpretation of what is occurring in the system.

On the commercial side, applications are, most of the time, black boxes that do not allow external communication either changes. They should only be modified within the companies or by authorized people. Thus, magnetic resonance researchers are unsupported,

and development tools are unavailable for them. It turns that both, hardware and software design, slow down the advances in the magnetic resonance technologies and methods.

Finally, the absence or the precarious and outdated documentation constitute an obstacle for both, commercial and non-commercial systems and tools. Also, a user-friendly and functional graphical user interface that fulfills the user needs is practically unnoticed in all of them. As a result, the reasons above reinforces the importance of our proposal, offering a programming-enhanced, flexible and scalable system that should comprehend all types of users.

4 DESIGN AND METHODS

As a consequence of the extensive scope of the work, this section focus will be to provide an overview of the principles, the necessary techniques, explaining their significance, and present a rationale for why specific design choices were made. Also, they will be related to what was cited in the review chapter. This chapter describes our proposed innovative framework called Python Magnetic Resonance framework (PyMR). The PyMR seeks to fulfill the gap presented in the literature: a flexible and reusable magnetic resonance system, based on well-established techniques and easy to use interface. Also, the PyMR aims to be a less-time-consuming, low-cost, and resourceful system allowing the user to set, create, maintain and use magnetic resonance systems and pulse sequences easily.

4.1 Principles

This section starts by describing the principles applied during project development, selection of programming paradigm, and the establishment of design requirements. In a first perspective, creating our description format, language or a whole new and isolated tool, instead of using pre-existent and already well developed, we can fail.⁷⁵ Commonly found in the literature, this approach generates limited solutions that sometimes are sufficient only for specific cases. For general cases, they ordinarily are faulty by the lack of support and its community size, for example. Also, when integrated with other tools, such as simulators or processors, the relationship and the usability are ordinarily poor. Many of them were exposed in the *Review* chapter.

Changing our strategy to another point-of-view, we can see that it is inevitable not to introduce programming techniques or some form of interaction with a programmable machine. In all cases, avoiding programming at all provide solutions that are not scalable. However, introducing deep concepts of programming, high advanced or exotic languages can withdraw the user. How much we can extract information, it is how much we will need to learn before starting. Thus, it has a balance between both solutions that can benefit all types of users.⁷⁵ Using the tools available in the literature and reviewing all the work already done, we start to propose our solution to this case by inserting some principles the meet our goals.

4.1.1 Programming paradigm

Typically, the structure of a program and its execution model define the programming paradigm although the existence of other classifications, they are beyond the scope of this discussion. The use of uncommon paradigms should, in general, be avoided, because they are more difficult for users to learn and use.^{30,32} As an example, the structure that employs the user-oriented paradigm (UOP)⁷² may not be the best choice, even being the more indicated one.⁷⁶

Finding the ideal paradigm is a difficult task. Furthermore, even after a specific paradigm is chosen, concepts and execution models from other paradigms are often found to be useful and thus are included as well.⁷⁶ We have chosen object-oriented programming (OOP) as the paradigm for PYMR framework. We arrived at this choice because of two features that make OOP intuitive to use: encapsulation of classes and their methods; and the direct relation between instances of a class and real-world objects.⁷⁶ In OOP classes represent objects, methods define behavior, and attributes describe states and relationships. Even though the states and relationships may be unclear since they could be executed/related in a different order, it is easy to keep modularity and abstraction in this structure. The majority of analyzed tools cited in the first chapters already embraces the object-oriented paradigm, relating this choice to the structure's simplification and understanding.

Although all our structure is object-oriented, part of the code uses execution models that may not reflect an object-oriented kind. Examples of those cases are in signal processing that is based on the procedural execution model - focused on the variable state changes, and in the graphical user interface that uses inherent event-oriented characteristics - based on event calls.

4.1.2 Software requirements specification

Determining software requirements specification is the first step in the software development.⁷⁶⁻⁷⁹ Questionnaires, interviews and other techniques from human-computer interaction provide requirements to improve user interaction and experience.^{77,78} The following items represent the results of the author's master's thesis²⁸ to date. They are also merged with items from API design references,⁸⁰⁻⁸² good practices of programming,⁷⁹ and our current *Review* exposed problems. Each requirement will be linked to specific features of the program, presented within the results.

- R1. Uniformity and readability in system and sequence programming (simple syntax in system and sequence definition);
- R2. Simplified, standardized, consistent and mnemonic naming (no jargons, nor abbreviations, but patterns);^{29,30}
- R3. High level of documentation, integrated with source code (facilitate learning and to date information);
- R4. Different visions compatible with all groups of users (from developer to operator);
- R5. Flexibility, mobility and easy to introduce new features;³⁰
- R6. Easy construction of new applications (GUI and CORE);³⁰
- R7. Simplified graphical user interface structure (user-friendly);^{77,78}
- R8. Simplification of the programming processes involved (modularization);
- R9. Standardization of inputs and outputs at different levels of programming (data pattern and formats);^{77,78}
- R10. Two correlated programming interfaces (CLI and GUI);
- R11. Cross-platform compatibility (Windows, Linux, OS X; including the possible mobile[†] and embedded systems[‡]).
- R12. Learn once, apply everywhere (use the same logic construction);
- R13. Implement once, use everywhere (use same items for different applications);
- R14. Allow future integration with pre-existing tools (Odin, JEMRIS).

In order to create a robust framework fulfilling state-of-art demands, other sources of information were included. The diagram in Figure 4.1 describes the literature-mapping process, which has two primary inputs. From the top, literature sources provide the fundamental prerequisites for the formulation of packages, modules, and classes. Those sources ensure up-to-date information about correlated fields, reflecting the generic interface. From the bottom of the diagram, that is, starting from the most specific point of the system, the manuals and datasheets provide the hardware-oriented specification. The information extracted is then set to

[†] Refers to the operation control and visualization, not massive computational operations.

[‡] Refers to a proposal of using FPGA Python interpreter or ARM processors, i.g Raspberry Pi.

the correct abstraction layer, by implementing the meta or base classes and their subclasses until the specialized classes, as object-oriented was chosen as the programming paradigm. There is a known subjective interference in this procedure when setting the layer level. However, it stills preserve the fundamental characteristics that will converge the structure to the best system's representation. Also, the framework's structure will be refined through its usage in new systems. This in-depth search for requirements, considering all levels of users, is the part of our system advantages.

Figure 4.1 – Diagram of literature-mapping requirements. Literature sources provide up-to-date information about what is necessary for the system, manifesting the generic API. Hardware-oriented materials such as manuals and data sheets provide the structure specialization.

Source: By the author.

For example, in our system, have used that mapping to generate the base and specialized classes for the RF amplifiers. Thus, PYMR has a `PyMRObject` class that represents a general item in the framework. From it, comes the `Peripheral` class, representing an ordinary peripheral. Then, there is a `Amplifier` class, for generic amplifiers from which `RFAmplifier` class is derived. This is depicted in Figure 4.2. All the classes mentioned, except the last one, are meta-classes, i.e., they are not meant to be instantiated directly; they must first be reimplemented. *Set* classes are containers for their related ones, having special methods and attributes to deal with their contents. For example, the container class should know how to calculate the total power for the `RFAmplifierSet`, or turn on all `RFAmplifiers` at the same time. All those previous class layers still inside the generic-framework layer. More specialized definitions will populate the framework-specialization layer and this will be discussed in details later in the section *Framework structure*.

Figure 4.2 – Example of the class diagram for the amplifiers class. The last class in the hierarchy is the instantiable class as indicated by the dashed line. *Set* classes are containers that have specific methods and attributes to deal with their contents.

Source: By the author.

4.1.3 Interfaces

From the beginning of the project, the requirement of being able to use both the command line and the graphical interface was made, as noted in the previous subsection. Although the graphical interface is more user-friendly, the command line interface is far more powerful, allowing the user a finer control over the system^{||}. This interface is related more closely to the programming experts or when high advanced modifications are required.^{29,72} Of course, this is not to say that the graphical interface is unimportant. The most practical for using magnetic resonance systems, which deal with lots of graphics data, is the graphical interface. More than that, magnetic resonance specialists and operators should use this interface as the main.^{32–35} To control everything in the system, both are fundamentally necessary.

Furthermore, all software cited in the review chapter experience precarious graphical interface and the use of advanced concepts of programming in the source code, which makes difficult to extended demanding lots of modifications, being a point to be improved in our systems.^{29,30,72} In our implementation, the model will enable an exemplary separation between graphical and non-graphical items. The concepts of visions will provide simple graphical interfaces, in programming point-of-view, but with extreme functionality and innumerable forms of visualization.

^{||} As programmers say: ‘GUI makes the things easier. CLI makes impossible things possible.’

4.2 Definitions

Now that we have described the principles behind our systems, the next step is to define elements that will be part of the system's implementation, which are programming-language-specific in most cases.

4.2.1 Programming language

In general, people that will use our tool will be scientists who may not be programming experts.²⁸ For these people, our selected language, Python, which is very similar to Matlab (a language extensively used in the scientific community²⁸) will be easy to use.³⁰ Python is a trendy language which has extensive documentation and resources, including numerical, scientific libraries and graphical support. Python also includes the possibility of being used with more than one programming paradigm, allowing great flexibility. Even though it is an interpreted language, it can be compiled and run as multiplatform bytecode.⁸³ Additionally, Python can run on mobile and embedded systems.

The TIOBE⁸⁴ index, which tracks the popularity of various programming languages, shows that historically popular languages such as Java, C, C++ are on the decline while Python is on the rise. In August 2018, TIOBE trumpeted that Python had broken into the top-three most popular languages for the first time.⁸⁴ Other measures also show the outbreak in popularity of Python; IEEE Spectrum⁸⁵ sets Python as the number-one in the rank, followed by C++, Java, and C, reaching the top between 2017 and 2018. Also, PYPL index⁸⁶ cite Python as the top and with the higher growth in the last five years, followed by Java and Javascript. As a result, the majority of recent popular technologies such as machine learning, artificial intelligence, and data science use Python;^{87,88} technologies that have also starting to appear in magnetic resonance context.

Besides, other widely used and robust programming languages have negative characteristics, which make them less desirable for our application.^{30,32} Following the indexes rank, we have popular languages such as C/C++ that are difficult to use^{30,32} and Java that has poor graphical interface support.⁷² The use of those languages in magnetic resonance context was raised as a corner in the learning path.^{28,30,32,70} Moreover, neither of them are as straightforward as Python.⁸³ Also, one of the reasons why Python has become so popular is its ability to efficiently interface with other faster languages such as C and C++, used for numerical and scientific libraries (NumPy,⁸⁹ SciPy⁹⁰). This ability is an essential characteristic as many

tools cited in the review are C/C++ based, which promotes future integration feasibility in the form of wrappers. For all the reasons above, Python was chosen to be the primary language in our framework. The ranking trends also confirm the correctness of this choice.

4.2.2 Documentation

The second point to be discussed is documentation. Documentation in all software projects are essential and should be maintained up-to-date.^{76,79} One of the best ways to achieve this goal is to keep the documentation as close as possible to the code.^{76,79} In Python, documentation can be inserted as docstrings, strings that describe the code's local aspects such as classes and methods. These docstrings serve to improve the readability of the code and provide both user and developer documentation. As shown by the author,²⁸ the primary source of knowledge for users in the magnetic resonance field is documentation (manuals, tutorials) rather than courses and technical training.

In the review, it was raised the lack of to date documentation in the current magnetic resonance software. Thus, providing good documentation is critical for this project and it was included in the requirements specification in *Principles*. All documentation should be written in English, as a primary and universal language defined by the scientific community.

Since Python is being used as the main language in our framework, we can use Sphinx,⁹¹ a tool for creating inline Python documentation. It uses the reStructuredText (rst) markup language, an official Python standard.⁸³ Sphinx is the most supported documenting tool for Python, and reStructuredText is one of the most used and complete markup languages.⁸³ Sphinx can generate many rendered formats such as HTML, TEX, PDF, EPUB.⁹¹ It includes many extensions that assist with documenting. It has some features like linked source-code, which makes the source code accessible from the documentation; the possibility of creating *live* tutorials that can execute functional code block, providing an exemplified and to date material; and it can integrate with document processors,⁹¹ which enable Sphinx to add reports (e.g., code metrics and profiling). Sphinx can deal with documenting patterns that not only the simple reStructuredText format, which includes Google and NumPy styles. Both are more legible in between the code.

Although those tools are complete in the general aspects of the documentation process, they do not have a pattern to deal with real physical values. These values are floating point values with units that represent physical quantities, e.g., *1.2 cm* (one point two centimeters). Therefore, a proposal for a pattern to deal with physical quantities in documentation is in the

Appendix F - Documenting physical quantities. Also, we proposed a new syntax for this issue that can provide better readability and usability in the code and it is described in *Appendix G - Proposal for a new variable type.* That is our contribution to the scientific community that uses programming languages and wants to conveniently describe physical values in the documentation. For this project, it was decided to use Python with Sphinx, setting Google Python documentation style, adding the proposed style for physical quantities.

4.2.3 Graphical items

A graphical user interface that fulfills the user experience and interaction design requirements deliver a user-friendly interface, simplifying user tasks. Visual aspects and its functionality turn the interface into a guide for users, avoiding errors and keeping them focused.⁷⁶⁻⁷⁹ The graphical framework ought to deliver flexibility and graphical quality to the implementation by those prerogatives. In this subsection, we will start by describing our particular choices and then conclude discussing other options.

The graphical user interface is based on Qt framework and widget toolkit by Nokia Company.⁹² Qt is one of the most adopted, multi-licensed and cross-platform graphical framework available, which includes mobile and embedded systems.⁹² This framework also includes the possibility of creating dynamic *tours*, which will make easier the learning process. Although it is based on C++, it has two main Python bindings, PyQt⁹³ and PySide,⁹⁴ that could be interchangeable by using an abstraction layer called QtPy.⁹⁵ Recently, PySide (version 2) became the official Python binding for Qt, which means that the original company will maintain it⁹², bringing long-term support and up-to-date features. Qt and its derived products still support 2D and 3D graphics acceleration using Open GL.⁹² Besides, Qt is compatible with the main scientific graphics libraries PyQtGraph,⁹⁶ Mayavi,⁹⁷ and Matplotlib.⁹⁸ which will be discussed later in this text. The exposed tools in the *Review* employ these selected libraries (PyQt, PySide, PyQtGraph, Matplotlib) massively. Other options for the graphical framework may not have one or more of those requirements mentioned at the beginning of this paragraph; some examples are wxPython, PyGUI, PyGObject, Pyforms.

As noted previously, scientific graphics libraries will provide the graphical data visualization in the project. PyQtGraph⁹⁶ is inside the framework because it can manage big data files (or streams) with fast visualization, managing real-time plots. By its side, Matplotlib⁹⁸ still a better option for static graphics, ensuring publication-quality figures. Up to now in project development, all needs were met by those two libraries, including 3D data. In the future,

Mayavi⁹⁷ can be used for better 3D visualization and interaction.

In short, the graphical interface in PyMR uses QtPy⁹⁵ abstraction layer for Qt (versions 4 and 5), compatible with bindings PySide (versions 1 and 2)⁹⁴ and PyQt (versions 4 and 5).⁹³ The latest versions of each one behave smoothly with package management tool such as PIP and Conda. PyQtGraph⁹⁶ and MatPLOLib⁹⁸ are used to handle with scientific data graphics visualization. Also, some contributions to these tools were made in the PyMR progress path, and they are described in the *Appendix I - Other contributions by the author*. The design of large magnetic resonance icon package is another contribution to the magnetic resonance context, included in *Appendix E - Icons for magnetic resonance context*, and based on.⁷⁶⁻⁷⁹

4.2.4 File items

Communication requires standardized input and output formats and patterns. If the data can be stored, files must have the same standardized models. As cited in *Review* a lack of patterns in the file formats is a problem in the magnetic resonance field. Thus, some file formats and patterns were chosen to expect to contribute to this issue.

The first file format, an obvious one, is the Python source code. It provides the possibility of classes reimplementation, code execution, and debugging. Python files are mostly generated automatically by PyMR through the graphical user interface or by direct source code programming. Changes in those files via GUI are processed by the Rope⁹⁹ Python library, a refactoring library that smart-edits the source code, including linked ones, i.g., imported files. It is important to notice that using a common language approach, it enables the debug the code, what is unseen in most of the tools presented in review.

In some cases, exported data needs to be in a human-readable file format such as JSON,¹⁰⁰ YML,¹⁰¹ XML,¹⁰² which allows inspection and editing. This is needed when string messages are transferred or when configuration files are employed. For this purpose, the first two formats give better readability, from the users' point-of-view. For high-performance scientific data storage, HDF5¹⁰³ is the gold standard. This format stores data in a binary format for quick access. Portable viewers can visualize data in the same way as library functions or toolkits such as Matlab. HDF5 allows access to large amounts of data with disk-mapped memory¹⁰³, avoiding overloading the system's memory. It is also used by DAMARIS³⁰ and it should be used for raw-data storage. Other formats as for processed data such as the medical image format DICOM,¹⁰⁴ ISMRM,¹⁰⁵ and the PACS¹⁰⁴ system are beyond our scope at this point of the work.

4.2.5 Integrated development environment

Programming is not an easy task. It becomes even more difficult when projects become vast and complex - with hundreds of lines of code in many files when developing a new system or a highly complex pulse sequence. This problem motivates the introduction of the integrated development environment (IDE), a tool that helps to manage the project and provide a favorable environment.¹⁰⁶ An IDE combines elements of all processes involved in software production: file navigation, text editor with syntax highlighting, auto-completion, validation, and tools for debugging, testing, project building, and deployment.¹⁰⁶ Also, it integrates other services such as version control systems, issue tracker, logger, wizards, and automate systems for time-consuming tasks, which will be considered in the *Auxiliary tools and techniques*.

After highlighting the importance of an IDE, we can discuss the adoption of this type of tool in the magnetic resonance context. Introduced by the author,²⁸ the concept of a magnetic resonance sequence or system as a software project, requires more than a text file and a simple text editor, which is currently provided by commercial vendors and reviewed tools as shown in *Review*. It is important to say that introducing this concept does not mean to introduce more complexity into the system. Instead, the idea is to insert common tools from software engineering that will assist the user. As mentioned before, specific visions can match the user level requirements and expertise.

Our concept of IDE applied to the magnetic resonance area is the first of its kind, comprehending all the steps of a magnetic resonance system and sequence implementation. Thus, a more sophisticated tool to take care of the project, an IDE, has been developed. Our very first version of the project included our specialized IDE,²⁸ which was then migrated into the Eclipse framework, one of the most supported and feature-rich IDEs. It was then discovered that there are severe communication problems between Python, Java, and the PyDev Eclipse Plugin, and it became necessary to explore other options. After researching, we decided to use Spyder IDE instead of Eclipse. Written in Python and using our same choices for the graphical framework and its features are similar to the MatLab environment.⁶⁷ This IDE is a pythonic[†] one with interactive Python and data visualization integrated into it. Moreover, it is known and broadly used by the Python community, with good support. Spyder's plug-in system deals better with our implemented tools integration. For now, the integration with Spyder have been

[†] This term refers to Python language, meaning basically simplicity.

implemented as a plugin, in which lots of our tools can be used for other applications if needed (e.g., object inspection, edition, and graph visualization). These generic tools show another contribution to the general community not only the magnetic resonance one.

4.3 Framework structure

Now that project requirements and definitions have been discussed we can move on to describe the framework's structure and layers. The internal structure will show the main package structure split into sub-packages, modules, and classes. It also includes the model adopted. Then, a discussion about the relationship with other projects will take place. Finally, the framework's external structure, representing the framework specialization, will be elaborated. We should call the reader's attention to the fact that one of the most significant problems presented in the review is the poor structure the current tools have. Thus, fitting a flexible and easy-to-learn structure shall contribute to the improvement of this field. Following, in the Table 4.1 we present general patterns that will help us to keep an organized structure and remembering from requirements no jargons nor abbreviations.

Table 4.1 – Naming patterns.

Member	Pattern
Project	spinal-case
Package	lowercase
Module	lowercase
Module constant	UPPERCASE
Class	CamelCase
Attribute	dromedaryCase
Property	dromedaryCase
Method	dromedaryCase
Variable	snake_case
Argument	snake_case
Function	snake_case

Source: By the author.

4.3.1 Model

The three-layer model Model-Control-Vision (MCV)⁷⁶ is given to PyMR, where the control and vision layer remain together. This adaptation turns the model into a variant known as M-CV⁷⁶ since the control layer is strongly linked to the graphics interface in this application. In this way, the CORE package contains all the structure necessary to provide a framework through the command line interface for the user, including all object-managing and

data-manipulating functions, as seen in Figure 4.3. Within the CORE, exceptions are issued in case of failures.⁸³ The CORE also includes complete logging,⁸³ which is essential for properly informing the user about the system's state. The GUI package contains the entire graphics interface that communicates with the core. Exceptions generated by the CORE are handled at this level and transformed into actions or informative messages to the user. Also, logging messages are processed then displayed.

Figure 4.3 – General M-CV model applied to PyMR's structure. Model of two layers in which the CORE contains the model, and the GUI contains the control and the vision. The CORE do not contain any kind of graphical library associated with it, and it should depends on the smallest possible number of external libraries.

Source: By the author.

4.3.2 Internal structure

The principles generated in the master's thesis of the author²⁸ are applied to the internal structure of the CORE, in which the system is subdivided vertically into the peripheral, device, and sequence adding now the data (data manipulation and storage) and model (virtual or physical models) packages, Figure 4.4. Those last two packages do not appear in the original structure²⁸ because they were recently added, being closely related to the data, subject, and simulation. Each package or module uses specific libraries related to its functionality, e.g., NMRGlue for spectroscopy signal processing.

In the CORE package, there are elements to manage all objects and its relations, instantiation, validation and so on, represented by the manager module. It also includes auxiliary modules such as enumerators for better representativeness, which is represented by enums. Listed below, we will discuss shortly each module that compounds the PyMR as presented in Figure 4.4:

- Peripheral, far components of the system that, commonly, deal with high power or external (from spectrometer) equipment, representing the magnetic resonance physical layer:
 - *Amplifier*: represents general amplifiers as RF, gradient, preamplifiers, attenuators and power supply;

Figure 4.4 – The internal planned structure for PyMR CORE. It shows the main packages and its modules. The Manager module take control off all instances created and Enums provides all enumerator data for the whole package. PERIPHERAL deals with all kind of external (from spectrometer itself) equipment, such as amplifiers, and represents also the magnetic resonance physical layer. DEVICE represents the internal devices, or the magnetic resonance conversion layer, such as digital to analog converters, DAC's. SEQUENCE, then, will represent the magnetic resonance logic layer, such as pulses. DATA package take care of data operations, such as processing and reconstruction. Finally, the MODEL will involve all sample and subject representation.

Source: By the author.

- *Coil*: represents all kind of coils for the use in RF, gradients, and shimming subsystems;
- *Cable*: add information about cables, wires and connectors used that can interfere on the signal (losses);
- *Chiller*: represents all kind of chiller and refrigeration systems;
- *Magnet*: relates to the static magnetic field and how it is provided, e.g., permanent magnet or superconductor.
- Device package is related to the closest or internal and low power devices, providing the conversion between commands to electric signals and vice-versa.
 - *Transmitter*: RF transmission devices as DAC;
 - *Receiver*: RF signal detection as ADC;
 - *Gradient*: for manipulation of magnetic field variation;
 - *Shimming*: the system to provide corrections in the static magnetic field;
- Sequence package represents the logic of magnetic resonance programming, based on pulse sequences that control subsystems:
 - *Shape*: normally represents waveforms shapes, i.e., without time relationship;
 - *Pulse*: are time-related shapes available for each subsystem;
 - *Event*: relates the pulses for a different subsystem that linked provide, or causes a

behavior in the system;

- *Sequence*: a concatenation of events that provides a purpose into the magnetic resonance system, more comprehensive than the event itself. It can contain loops and statements, or be atomic (not allow interaction when executing);
- *Protocol*: protocol is a form of setup the sequence for a specific purpose. Thus, it can extract different information from the subject;
- Data package represents all manipulation, storage, and processing steps. Ordinarily, the first two in the list require high processing power:
 - *Reconstruction*: deals with the primarily received raw data that still unordered and unprocessed, making the first and ordinarily processing step (implementation in the process);
 - *Processing*: receives and processes the data already reconstructed, or processes images and spectra (implementation in the process).
 - *Geometry*: deals with image and system geometry, exemplified by the prescription process in an already acquired and processed image, selection of the region of interest (ROI), or magnet bore size (implementation in progress).
 - *Database*: to be used for system management, personal data, access permission (not implemented).
- Model package relay on the entities that are used to simulate systems parts and phenomena:
 - *Nucleus*: representation of a magnetic resonance functional nucleus. It is an essential element;
 - *Phantom*: simplified physical models of subjects or samples, which can include geometry;
 - *Sample*: solid and liquid models, simpler than subjects, but more detailed than phantoms (not implemented);
 - *Subject*: human or animal body parts, or whole body, used as models, including complex interactions, movement (not implemented);
 - *Emulator*: imitate a real system, i.e., Digital FPGA Spectrometer, providing cheaper test systems (implementation in progress);
 - *Simulator*: integrate the available tools, e.g., ODIN, JEMERIS, cited in the review

chapter. At this point, we included a basic Bloch evolve for simulation of systems of spins. The connection with other tools needs to be done.

Another main package, GUI, the structure is different, and it is presented in Figure 4.5. As it will use the vision concept, there is a graphical manager module which concentrates the management of all interfaces and communicates to the CORE manager.

Figure 4.5 – The internal planned structure of PyMR GUI package. It shows the packages and modules that represents the control and visions. This package has a graphical manager that controls all the other interfaces. The UI package has the main graphical data editor and visualizations or plots. Then, the package VISION includes auxiliary visions created from templates.

Source: By the author.

4.3.3 External structure

PyMR framework was designed to support a vast number of types of magnetic resonance equipment. It accomplishes this through the use of abstraction layers. The basic idea behind this is to create a few specialization layers that can be reused, describing the specificity level in the direction of the final product. Figure 4.6 shows the layers that represent each step in the process. This procedure helps developers to refine the framework members abstraction from both extremities (noticed by the yellow color). For example, when finding any *generic* item in a *specialized* layer, it can be moved to a high level of abstraction. Similarly, if any *specialized* item is found in a more *generic* layer, it should be moved forward. This structure is similar to the framework itself, but now the specialization is external to it. Each layer here can be stored in a different repository, keeping tracking on changes and issues. The following list defines the steps, including some examples of spectrometers.

- **Framework:** First implementation (which does not include meta-classes) should describe a generic magnetic resonance system, the PyMR framework itself. In the best case, all the input and output shall pass through the generic PyMR API, only instantiating and setting parameters, which will turn the code be functional in any system, what is desirable for reproducibility purposes. Examples: AnalogDigitalConverter, RFamplifierSet,

Figure 4.6 – Framework's specialization diagram. Each step into the final product means the more specialized the classes would be. In the reverse path, generic items are pushed to high abstraction layers. The number of layers can be defined as needed, and each one is stored in a different repository. Examples are given for each layer.

Source: By the author.

GradientAmplifier, SincPulse, SquareShape. All of them are the final classes of the framework layer.

- **Technique Specialization:** Derived from the first one, is the first reimplementation targeted to the system application type: Relaxometry, Spectroscopy, and Imaging. Those applications have some particularities that could change the system format. For example, an imaging system needs, ordinarily, gradient subsystems that are not used in Relaxometry and Spectroscopy modalities. Also, sequences, data reconstruction, and processing can vary significantly for each application.
- **Product Specialization:** Being the third implementation, it should specify a product line, e.g., generic to small differences among the products such as the number of channels or base frequency. Ex.: A spectrometer line named Hahn for Spectroscopy application with one or two channels that could deal with Hydrogen and Fluorine. A refinement can take place here, creating more layers, but increasing the number of layers can increase the structure's complexity, so it is discouraged.
- **Product Instanciation:** It is an instance layer, in which all values for a specific product are filled and set up. It is the final layer, ready to be used. Example, following the previous one: Hahn Spectrometer with 1 Channel for Hydrogen and another system for Hahn Spectrometer with 2 Channels for Fluorine.

4.3.4 Project relationships

Considering the scale of the whole project, and that some of its parts started at a different time, we show the whole relationship with other projects and infrastructure in Figure 4.7. The PyMR project seeks to provide a structure for integrating these various parts into a single system. The general scheme of the structure can be seen in Figure 4.7, which shows a simple, but complete system. The top infrastructure layer can contain different types of hardware, at different locations. In the middle, we have the back-end, a specialized version of our PyMR framework that considers hardware dedicated elements. In this layer, the elements can also be in different machines (e.g., PC and Raspberry Pi). For each package, discussed in the previous sections, there is a manager that take care of specific tasks for each service. For example, the device and peripheral manager can communicate using low-level communication such as SPI (Serial Peripheral Interface), which will not be necessary for other packages such as data and model. Then, in the bottom, a generic PyMR API is used. All of them should use the same and common CORE and GUI manager from PyMR to execute the tasks.

Figure 4.7 – PyMR framework and its relationship with other projects. Each container represents a set of elements compounding the project infrastructure that, ordinarily, are in different physical location or machines. The PyMR provides all the code structure to create the whole software system. The Spyder and our PyMR Plugin help to manage, program, validate, and check those created elements.

Source: By the author.

For each specialized manager, we will detail its responsibility and cite project related in the following list. Other people from CIERMag group implemented specific code that are being integrated into managers in the PyMR framework.

- The peripheral manager is a system that takes care of all peripherals.^{107,108} If any problem occurs, it can promptly respond, then it sends a notification to the host (console). The

system is currently running on Raspberry Pi, and it can manage many systems using the ethernet connection and SPI communication. The peripherals itself are commercial models such as amplifiers, whereas most of the coils are locally produced.

- Device manager is the system that deals with the device controller.^{28,50,107,108} It sends and receives data and commands to the spectrometer (in this case, the digital spectrometer implemented in FPGA by our group). At present, it can run on the same machine as the console, or it can communicate through the ethernet connection. It has auxiliary boards to input and output data.
- Sequence manager is the project that controls the experiment itself when running on a real system.^{28,50} It uses the core and graphical functionalities of PyMR to provide the control panel. From the panel, the user can choose a sequence, adjust the parameters and then run (device manager). Also, when data is acquired, the console will show the processed data.
- The data manager will process the data coming from the device manager or pre-acquired data, operating to reconstruct images or process signals, e.g., NMR spectra.^{28,50} Also, it will deal with images and experiment prescription, in which the user can set a new experiment based on the previous one. Then, the changes in the parameters should be passed to the sequence manager (console).
- The model manager, finally, deals with simulated data, models of body or emulation of our spectrometer.

All those subprojects have been contributing to new and modern forms of magnetic resonance system configuration and usage. More details from each one can be seen in its references. As cited at the beginning of this subsection, PyMR aims to concentrate them. A shared structure will provide a better communication interface and code reuse overall. The same concepts used in the framework development process will, then, be part of the whole system. This well-defined structure, including model, internal and external specialization layers are rarely or not seen in the magnetic resonance context. Thus, they can be included in our contributions as well.

4.4 Auxiliary tools and techniques

In previous sections, critical information was given, and the framework striking characteristics and decisions were specified. Now, we will discuss auxiliary tools and techniques.

All the items that will be considered also can benefit the magnetic resonance sequence and system development, becoming an enhancement in this field, which is not the current scenario in commercial products and literature available options. Other tools that can contribute, but they were not extensively used in this development can be seen in the *Appendix B - Access control, integrity*. As mentioned when talking about the IDE, all these tools can be integrated with the PyMR environment.

4.4.1 Logging and exceptions

A robust logging and exception systems can benefit all types of users, from the developer to the operator. These tools can help to track issues, keep the user informed, and to note changes they made. The logging system is based on the standard Python logging library, defined in PEP 282. The types of logs/exceptions are listed below.

- Critical: an error that possibly stops code execution, causing a crash, e.g., segmentation fault.
- Error (also an exception): an unexpected situation that can be solved without stopping the whole execution, usually solved by the user, e.g., file not found, could not write in the file. Those are provided in the exception list also.
- Warning (also an exception): an expected error in which fix is already known and continues the execution, e.g., the value is high than the maximum, setting the value to the maximum. Those are provided in the exception list also.
- Information: a piece of information for the user/developer about execution, e.g., the file was saved successfully.
- Debug: detailed information about the execution, commonly used by developers, e.g., executing the 13th 'for' loop iteration.

4.4.2 Work management

Due to the project size and complexity, people who are involved in the system's construction need to establish a workflow. The first item discussed is the approach for the development, using the Scrum framework.¹⁰⁹ Scrum is an Agile¹⁰⁹ technique, a suite of iterative and incremental work management methodologies, mainly applied to software development.¹¹⁰ Applying Scrum to a project is quite easy, fast and more flexible, compared to other non-agile methodologies.¹¹⁰ It is important to note that Agile techniques normally avoid spending

time on documentation, what goes against our requirements, so this characteristic will not be followed. The Figure 4.8 shows the workflow. There is a product backlog from which tasks are selected in the sprint planning, generating the sprint backlog. Items from sprint backlog are then implemented in the sprint, amidst of daily/weekly/monthly meetings. Thus, the item goes to a sprint review, being replanned or producing an increment in the final product. The sprint retrospective provides feedback to recycle. The cycle is repeated until the product is finished.

Figure 4.8 – Workflow diagram of the Scrum framework. Scrum is based on interactive and incremental work management, affording a working product increment.

Source: By the author.

4.4.3 Version control system

A version control system (VCS) tracks information about changes in files. This feature is crucial for all software projects and also for work coordination. It allows synchronization of data between remote servers and local machines. The version control system is a generalization of the source code management, in which the aim is the source code file. Three of the main VCS tools are Git,¹¹¹ Mercurial,¹¹² and CVS. PyMR and MR Sprint projects use both Git and Mercurial due to different project aims.

4.4.4 Integration system

As the code grows, simple tasks become more complicated, requiring many massive tasks that can be automated. Thus, continuous integration (CI) systems can help developers execute repeated tasks; this is done using a trigger.¹¹³ For example, pushing code to the server can be used as a trigger for the generation of rendered documentation. In our system, Jenkins CI¹¹⁴ assists the developer in automatically executing many tasks on the server, e.g., running tests and generating reports.

4.4.5 Code metrics

As the code is implemented, it is necessary to have metrics that can define the code quality in a more general format.^{76,79,110} Code metrics provide information about or to the organization, control, planning and improvements. Many of them are also available by the plugin system of the IDE for in the PyPI repository. Therefore, we are considering common code metrics applied to measure system quality as listed:

- Raw metrics: they are based only on raw values such as the number of sources lines of code, comment lines, blank lines. Raw metrics provides preliminary analysis, with basic information by proportionality;⁷⁶
- Cyclomatic complexity: it can measure how many paths could exist in a decision-based function. So, using lots of conditionals can reflect into a complex code, being difficult to understand the possible results (paths);⁷⁶
- Halstead metrics: it is expressed concerning the number of operations and operators that are used in the code. Many operations in a single line indicate a more complex code, which should be simplified;⁷⁶
- Maintainability index: even though the previous metrics already provides a type of complexity, which also indicates how difficult it would maintain it, this index consider X, Y and Z to calculate the values;
- The depth of inheritance: a long list of inheritance in class design can complicate its use, is related to the structure of the code. Three to four levels represent a balance between how many inheritances levels are ideal;⁷⁶
- Coupling: relates to the level of members overlays. More coupling means more thig they are, so more difficult will be to substitute the parts from each other. Reducing coupling will keep the elements independent;⁷⁶
- Cohesion: indicates how closely related the members are to other ones. Close members are desirable to have more cohesion, i.e., more specific tasks in the same place or near one another.⁷⁶

4.4.6 Versioning and distribution

We follow the versioning system described by Semantic Versioning (v2.0.0)¹¹⁵ which is compatible with PEP440. Distributions are labeled according to the following format.

Distributions with different major numbers have drastic differences, and there is no guarantee that their API's are compatible. Distributions that only differ in minor numbers have additional functionalities added. Moreover, distributions that only differ in patch number have bug fixes. In the case that changes have only been made to minor and patch numbers, the interface is preserved.¹¹⁵ For the distribution, PIP package management system is used to install the source code or generate the bytecode, offering advantages over other available package management systems such as *easy_install*, and *python setup install*.

4.5 Summary

Regarding our goals of providing a robust framework to develop, set up and control a magnetic resonance system, the choices we have made are in favor of the code and structure simplicity and readability. Using a programming language that is vastly used and considered as one of the simplest languages, we will reach the mid-term between the interface for programmers and specialists that program pulse sequences. Also, this enhances the power to create complex and high advanced systems. Besides, it brings a significant quantity different manners the user can program the sequence by providing templates for simple to advanced cases. All the tools coupled with the framework itself, such as the IDE and our specialized plugin, metrics for evaluating the system's code quality, or the introduction of version control if the user needs, support the robust environment for the development that was never used. That approach implies on assimilating concepts of many areas by not reinventing solutions that were well-established, to provide a robust environment. In the next section, we will show the implementation and results for the PyMR framework and related tools.

5 IMPLEMENTATION AND RESULTS

The results will focus primarily on the PyMR and the MR Sprint final characteristics and data. Then, an explicit example of a simple magnetic resonance sequence, a free induction decay (FID) acquisition and processing, will be debated in the MR Sprint using the PyMR integrated. This section describes the design and the methods applied to develop the PyMR like the fundamental principles required for good software development, the definition of the elements of our system implementation, the framework structure and the relationship with other modules and the auxiliary tools. Then, the following section depicts the implementation and results achieved by PyMR, the PyMR Spyder IDE plugin, as a second significant contribution of this thesis and the third major contribution, the MR SPRINT a new magnetic resonance experiment simulator, also supported by the PyMR framework.

5.1 General characteristics

The developed software is multiplatform compatible[†], including portability between Python distributions and machine configurations (R11). All the configuration used are listed below since PyMR has been used for a long time, almost three years with its earlier versions, which not included all features presented in this thesis. Thus, the footnote indicates the items used to provide the presented results. Also, we highlighted the recommended configurations.

- **Desktop machine configuration**

- *Intel Core i7-4790K@4.0GHz, 16GB RAM, 1TB HD, Intel Haswell Graphics[‡];*
- *Intel Core i7-3960@3.3GHz, 16GB DDR3, 1TB HD, Intel HD Graphics 630;*
- *Intel Core i7-3890@3.6GHz, 16GB DDR3, 1TB HD, NVIDIA GT640M 1GB;*
- *Intel Core i7-7700@3.6GHz, 8GB DDR4, 1TB HD, Intel HD Graphics 630;*
- *Intel Core i7-920@2.67GHz, 6GB DDR3, 1TB HD, NVIDIA GT9800 1GB;*
- *Intel Core i7-920@2.67GHz, 12GB DDR3, 100GB HD, Gallium Graphics 0.4NV96;*

[†] It still needs testing on OSx.

[‡] Configuration used to provide the results presented in this thesis.

- **Laptop machine configuration**
 - Intel Core i7-4510U@2.0GHz, 8GB DDR3, 500GB HD, Intel HD Graphics, NVIDIA GT740M 2GB;
- **Operating system**
 - Microsoft Windows (8.1 and 10, 64 bits);
 - Linux Ubuntu (14.04, 16.04⁺, and 18.04, which are LTS versions, 64 bits);
 - Linux Arch (64 bits);
- **Python distribution**
 - Python (3.4, 3.6⁺, and 3.7);
 - Anaconda (3.5, 4.3, 4.5⁺, using previous Python versions);
 - WinPython (using previous Python versions);
- **Integrated development environment**
 - Spyder IDE (3.2.4, 3.3.1⁺);
 - Eclipse IDE (4.7) - now it is incompatible, migrated to Spyder;
- **FPGA spectrometer configuration and additional boards**
 - Terasic Altera DE3, Transmitter 2 Ch., Receiver 2 Ch., Gradient 4 Ch.;
 - Terasic Altera DE4, Transmitter 2 Ch., Receiver 2 Ch., Virtual Receiver 2 Ch., Gradient 4 Ch.;

5.2 PyMR framework

As we will see in the next subsections, our framework offers many graphical interfaces for different types of interaction and visualization of the system, pulse sequence, and data. Although they can be combined freely (docked) or exhibited in particular combinations for different contexts, it is recommended to provide wide displays for better usability. All files to create a new system are written in Python language. It is possible and easy to debug and test the system by employing the Spyder IDE with PyMR plugin, providing wizards to many tasks via the graphical interface, such as creating new elements and editing values or using similar programming tools. As mentioned before, using software metrics can help both, developer and user, to check the code quality. In the Table 5.1 we present the main metrics that give a demonstration of code quality. Also, in the Table 5.2 it is displayed the PyMR framework main

characteristics.

Table 5.1 – Code metric summary for PyMR framework.

Code metric	Value
McCabe's complexity (cyclomatic complexity)	A (1.6)
Maintainability index (Visual Studio)	A (87)
Lines of code - LOC	6485
Logical lines of code - LLOC	1811
Source lines of code - SLOC	1826
Comments - C	128
Single comments	268
Multi - M	2456
Blank	1935

Source: By the author.

Table 5.2 – PyMR framework main characteristics.

Entity	Value/Quantity
Packages + Subpackages	8
Modules	43
Metaclasses + Classes	125
Methods and Functions	248
Number of pages in PDF documentation	180

Source: By the author.

5.2.1 Structure

Accomplishing our aims established in the previous section, we show the current final structure provided by the PyMR in the Figure 5.1. Two main packages divide the PyMR into CORE and GUI structure, then other packages and modules follow the planned distribution. Next, we will exemplify some of the class hierarchy given by PyMR by the Figure 5.2, showing the Coil related classes; Figure 5.3 that provides three common types of magnets and Figure 5.4, depicting the shape waveforms that are ordinarily used in the pulse generation. Not all of them are exhibited because of license issues. The reader should notice the limited number of members in the hierarchy. In *Appendix C - Examples of documentation* and *Appendix C - Examples of documentation* there are some examples of both documentation and tutorial, respectively. They include the RFamplifier class, mentioned in the previous section, and the use of Shape to create a Pulse.

Figure 5.1 – Current package structure for PyMR framework. It is listed the packages and modules for all the PyMR structure.

Source: By the author.

Figure 5.2 – Class hierarchy for Coil and CoilSet peripherals. Coils were divided into subsystems. Each one has a type that describes its construction and functionality, e.g., Birdcage or Solenoid.

Source: By the author.

Figure 5.3 – Class hierarchy for Magnet and MagnetSet peripherals, they are divided in three types.
Source: By the author.

Figure 5.4 – Class hierarchy for Shape and ShapeSet that compound pulses in the sequences. Shapes do not have a time associated, but pulses have.

Source: By the author.

5.2.2 Object manager

The object manager is a generic handler for any object, for both GUI and CLI interfaces. The CORE manager performs the operations along with the command line interface, while the GUI manager can graphically edit members and their structure. All the operations are executed over objects, whenever its type is. Also, graphical interfaces provide searching filters and shortcuts, making tasks less time-consuming. Thus, we can provide two different paths to perform the same tasks, comprehending issues about CLI and GUI interactions (R5, R6, R10).

As all the interfaces are generic, therefore they can be adopted by all sub-projects, accelerating the development because it is not necessary implementing or inserting any unique code. This fact represents the completion of implement once, use anywhere requirement (R13), saving time and increasing production for programmers. Besides, any user, in any group level, can use the same and consistent interface to deal with the system. In other words, this condition meets the requirement of learning once, apply everywhere (R12).

Dock widgets allow the users to organize the GUI along with their preferences, making them highly personalized. Hovering graphical elements returns the documentation, internal to the PyMR, for the user as a tooltip. Inside the code, Spyder IDE automatically displays the code's documentation. This interaction is part of our GUI simplification, showing the users the information they need to know at the right time. This feature complies with part of the required by documentation issues (R3).

Icons depict the system's elements, providing a yet more user-friendly interface and further help for the learning process, memorization, and understanding (R1, R2). The complete class name `package.module.ClassName` maps the real path `package/module/ClassName.png`, consequently we can find the correct image. Details about the icons and their construction, specialized for magnetic resonance systems, are in *Appendix E - Icons for magnetic resonance context*. Also, whenever possible, we direct format code element names to the graphical interface. This concept provides two practical results, we do not need to create individual names to show in the GUI, and also, users utilizing the GUI will remember the names that match precisely when editing the source codes, accelerating learning.

In the next topics, we will see the three interfaces that fulfill all the essential user needs to create, control and edit the system elements. Then, we will show the vision manager that can provide a dynamic approach for user-group- or application-personalized views. Finally, we will discuss the implementation of the data manager that provides visualization and data edition in

our framework.

Object creator

The object creator receives the name of an installed Python package or module (one or more) internally. Then, it lists all classes with its respective icon, name, documentation (docstrings) and package information, Figure 5.5. Double-clicking or dragging-and-dropping an item, makes the object creator emits a signal to the object inspector, creating an instance of that class inside the current structure. The object inspector will be discussed later in this text. Virtual classes and other non-desirable classes are invisible to the user in the GUI. Our extension for the Python inspect library, called *inspectextend* *Appendix I - Other contributions by the author*, allows us to filter the package contents before they are displayed.

Figure 5.5 – The GUI for object creator with its view menu. a) The first column gives the class name and the icon that represents it. The second column shows the class documentation. Then, its base class and subclasses are listed. There are also more options that can be displayed if the user wants. b) The view menu can change the structure hierarchy displayed in the first two actions. The other ones apply pre-defined filters, for content-related members.

Source: By the author.

Package hierarchy and class hierarchy are two types of visions the user can choose from our interface to change the items arrangement. The interface has pre-defined filters that can show sequence-, device-, and peripheral-only related classes, reducing the number of items displayed. Other packages, i.e., data and model, will be included soon. This approach solves problems with all the interfaces of creating objects, e.g., in the system establishment we can drag an RF amplifier and drop it inside the inspector structure, or when creating a sequence, an RF pulse can be inserted identically (R13). Linking those pre-defined filters to the user's group or context warrants limited and focused vision, satisfying related issues (R2, R4, R7, R13).

Object inspector

Inside the object inspector, in the tree view, all the instances that are in the object manager (CORE) can be seen with their respective relationships. The tree shows the name of the instance, the label, and the class name, Figure 5.6. The inspector also concentrates all operations the manager can perform such as add, remove, update, import, export and so on, as seen on the right of Figure 5.6. All those actions can be accessed by the menu, toolbar, and the context menu inside the tree view. By double-clicking any item in the tree, a signal will be sent to the object editor, for the edition, which will be our next topic.

Figure 5.6 – The GUI for the object inspector with its menus. a) The tree view shows all the core managed objects with their instance name, label, and class name. b) The action menu can execute on any object of those operations. c) The view menu can show or hide parent and children instances.

Source: By the author.

Although the tree view is a natural and most common form of structure visualization, it is limited to linear and noncyclic representations. This problem seems to be neglected in the literature even though most of the magnetic resonance systems carry this problem with them. Imagine we have an RF coil that can be used to both transmit (T) and receive (R) signals, called T/R configuration. This coil should appear in the RF transmitter system and RF receiver system, but it stills an unique entity. In the tree view, it is necessary to duplicate the graphical item that represents the coil, which may cause problems if the automatic duplication is unnoticed. Thus, we introduce the use of graph structure providing a view for disambiguation that will avoid misinterpretation, which can be seen in Figure 5.7. Next, we will discuss the object editor. This use of graph technique to identify structural problems or validate a structure seems to be an

uncovered situation by the literature, in our first search. However, a profound revision must be done to confirm this proposal.

Figure 5.7 – Graph view option for object inspector. The graph view can show a cyclic structure that is hidden in the object inspector tree. The red highlighted node have to connections from receiver and transmitter, exemplifying the condition of transmitting and receiving with the same coil.

Source: By the author.

Object editor

Sending an object to the editor, it can determine the type for each object attribute or property, and then it selects the appropriate data-field editor, shown in Figure 5.8. The editor can manage simple data types (float, integer), composed types (lists, arrays, dictionaries), and so forth. PyQtGraph library provides simple data-field editors, and the integration with the Spyder library allows us to edit composed data. Other types of data editors were implemented by us such for enumerators and physical quantities (Pint library data type), discussed in the next paragraphs. Also, the editor can list the object values by its class hierarchy, which simplifies the understanding of the structure (R1, R2, R7, R12, R13). As the values are changed (initial or constants), the Rope and Spyder library performs the changes, such a refactoring mode, in which the changes are traceable.

As previously stated, we implemented some special data editors. The first one is the enumerator editor. Enumerators are essential to provide readability in the code, assigning names to values and also given context to them. In our implementation, the field-editor for enumerator lists all possible values as a drop-down menu, which is pre-defined in the enumerator class as

Figure 5.8 – The GUI for the object editor and its action menu. a) A right field-editor is inserted according to the value type. The editor can separate the properties by class hierarchy or by name, which makes easier the understanding of the class structure. b) Editor for an enumerator type that lists all possible values for it. c) Actions available for the editor.

Source: By the author.

attributes, as indicated the first arrow in Figure 5.8 a and in Figure 5.8 b. The reading is clearer when we found `self.amplificationBandwidth=BandwidthMode.Broadband` than remember that the value zero represents that configuration (i.g., `self.amplificationBandwidth=0`), or using something that needs to be parsed (i.g., strings, `self.amplificationBandwidth='Broadband'`).

Following the same principles and noticing that in our application most of the values are physical quantities, the introduction of values with units are essential. The back-end to deal with this values is the library Pint. As well as happens with the enumerators, it is easier to read `self.frequency=pint.PhysicalQuantity(23.0,'MHz')` than using just an annotation in the code as `self.frequency=23.0 # MHz`, or even parsing strings `self.frequency='23MHz'`. The second arrow in the Figure 5.8 a indicates that example. We have also proposed a new syntax in Python, to simplify the physical quantity value assignment, which can be seen in *Appendix G - Proposal for a new variable type*, but it requires more discussion within the Python community. However, if it was accepted, it will be the first programming language to deal with physical quantities naturally, being a significant contribution to the programming area dedicated to engineering applications.

Introducing enumerators and physical quantities promote great readability in the code and, also, avoid misinterpretation and conversion problems. Both accomplish some of the issues related to standards, mnemonic, readability, and so forth (R1, R2, R7).

5.2.3 Vision manager

Similarly to the object manager, the vision manager can create and deal with different visions for the system, specialized for the user group level or distinct applications. The vision manager can produce focused graphical interfaces, targeting dedicated commercial applications. Our proposed process to create new visions aims, primarily, the developers. If necessary, simplifications in this process could be made to transfer that idea to other group levels too. The GUI's for visions are created using the Qt Design, assigning PyMR class and attributes names to the widgets elements. Using a pattern assignment, we can fill the interface by walking through the objects of our current system, creating any vision.

Three naming possibilities supply the linking between the objects and the graphical interface. If the widget name contains the exact name of a PyMR class (*CamelCase*), this widget should represent the instances of that class. If the name is non-capitalized camel case (we named it *dromedaryCase*), it will represent a property or attribute. Noticeable, properties can have methods to “get” and “set” the values, so if just the “get” is available, it will be read-only. Otherwise, the user can edit it directly in the GUI. Lowercase names with underscores (*snake_case*), wherever they are, are just structural widgets, so they are neglected. We can see in Table 5.3 the naming patterns. As the names can repeat and Qt Design automatically number them, numbers are ignored too. Lowercase names with hyphens (*spinal-case*) are not permitted. The vision will fill the interface automatically with data from the objects, considering the patterns.

Table 5.3 – Patterns used in the template.

Pattern name	Match with	Text displayed	Example
CamelCase	ClassName	Object Label	ReceiverSet > Receivers
CamelCase	ClassName	Class Name	ReceiverSet > Receiver Set
dromedaryCase	propertyName	Property Name + Field	maximumPower > Maximum Power + Field
dromedaryCase	attributeName	Attribute Name + Field	gain > Gain + Field
snake_case	—	—	rf_frame_widget
spinal-case	Not permitted	—	—

Source: By the author.

Qt Design can manage widgets using drag-and-drop operations to create a structure in XML, and then it can be converted to a Python file using Qt binding in this process. There is also

an option to use directly the XML created, which loads the XML file using the functions available in the Qt bindings. Those files will be our templates for our vision GUI. The structure can be created manually by editing the XML or Python file, but it is hard to follow all the rules hence we recommend the graphical edition. In the Figure 5.9, is illustrated the Qt Design generated structure containing a simple example with transmitter and receiver subsystems. The resulting view will be shown in the next paragraphs.

Object	Class
simple_example	QWidget
subsystems_tab	QTabWidget
ReceiverSet	QWidget
Receiver	QGroupBox
nucleus_2	QComboBox
TransmitterSet	QWidget
transmitter_structure	QWidget
Transmitter	QGroupBox
nucleus	QComboBox

Figure 5.9 – The template of a simple structure generated by Qt Design. The pattern applied to the object names are: capitalized camel case represent object classes; non-capitalized camel case denote object properties and attributes; lower case names with underscores (only structural elements) and names with numbers (in replicated names) are ignored.

Source: By the author.

The object manager structure will be searched to fill the graphical interface with the necessary information. In the graphical interface, the widgets will be identified by the object label for better readability (R2). If the object label is empty or if it is a property or attribute, the graphical label will be generated converting the name into the title case, i.e., inserting blank spaces and capitalized words (e.g., `fieldOfView` will generate the label `Field Of View`). The graphical template and the resulting graphical interface for the previous example are presented in Figure 5.10. Note that if it exists more than one member, the structure is replicated, and then, inserted according to the layout chosen, e.g., the two receivers in Figure 5.10.

All widgets are automatically linked with the object manager elements, inspector and editor, and also with others visions, if existent. This construction creates a pattern in the interfaces that can be quickly learned by the user. Also, it simplifies to create new interfaces for different systems, given to them a personalized aspect. To create a new and fully functional interface is only necessary to provide a new template. Names that do not match the patterns, or do not match the structure (e.g., a nonexistent class instance in the current system) are just ignored. However, if it happens, it is logged a warning in the system.

It is important to emphasize that this process is also employed for exhibiting the graphical version of the pulse sequence. This simple structure can automatically construct the pulse sequence visualization, almost ready to be used as a graphical-sequence editor. To be

Figure 5.10 – Graphical template and its generated view. They are exemplified by a system with two receivers and one transmitter. a) Template for receiver set tab. b) Generated view for receiver set tab. If more than one member exists, then it is inserted another one, following the layout. c) Template for transmitter set tab. d) Generated view for the transmitter tab with only one transmitter.

Source: By the author.

finished, we need to implement drag-and-drop actions in this graphical interface, which then, will be reflected in the structure. Up to now, the structure can be reorganized or edited in the inspector, by drag-and-dropping the elements. To make our idea clearly, we present an example in the Figure 5.11, in which is simulated a sequence edition. Also, this example includes the object inspector, which indicates the location of that element in the structure. The editor, then, allows the user to edit all the attributes and properties of the current element. We call the readers attention to the fact that this feature is practically absent in any software available, and it is highly required for non-programmers.

Figure 5.11 – An example of a drag-and-drop edition for the sequence. The graphical edition will change the structure. Clicking on any element, give the user the opportunity to edit the object it represents. This action also selects the same object other views. a) Pre-edited sequence event state: Execution, Delay A, Acquisition, Delay B. b) Dragging the sequence event Delay A. c) Post-edited sequence event state: Execution, Acquisition, Delay A, Delay B.

Source: By the author.

Even though the developer can create any vision they want, we provide some of the

most required interfaces in all types of magnetic resonance systems (R?), which will be following discussed. Those standard interfaces deliver to any system a similar process to interact with it (R1, R2, R7). Each of them has a fundamental purpose that fills the main steps for running an experiment in the system, as seen in *Concepts of magnetic resonance*. At the first time creating the system, therefore, there is no need to create a new interface (R13).

The first of its kind is the setup vision which aims to set up the system, by the manager user group, and it is displayed using tabs to identify each subsystem. We show in the Figure 5.12 three of the ordinarily subsystems: transmitter, receiver, and gradients. The elements organization is based on the path of the electric signal. The control vision, Figure 5.13, is an interface used to control the experiment in the console, at runtime. This interface has a reduced number of members, and they are disposed to the specialist user's group. It still has some minor configuration and adjusts before start the experiment. In the *fast* vision, the interface is planned to be ordinarily used by the operator in the one-click mode, almost without any adjusts. This vision can be seen in Figure 5.14.

To reinforce the importance of this approach we call the attention to the fact that for all those different interfaces, or visions, we only provided a template file. No further programming was necessary to connect the signals or fill the values. Although the templates are currently being generated in the Qt Designer, in the future this composition could be done inside our framework.

5.2.4 Data manager

In our system, there is another vision, called plot view, that can combine all the elements of the sequence and plot them as a single plot. This vision will be shown later. All the graphics and data visualization is for all of the framework structure. The scientific graphics provide the most common interaction (zoom, selection, rescaling, import and export data). It is also essential to present data visualization for setup or to calibrate the system, not only for the acquired data. We can cite field-map visualization for the magnet and coils, the calibration process when matching the system resonance frequency, temperature monitoring, data from input and output ports (i.e., oscilloscope mode).

As mentioned in the methods, most of our views are based on the PyQtGraph library, exemplified by Figure 5.15. In this figure, we show 2D plot data, showing a magnetic resonance sequence preview and a simulated signal. In some cases, static graphs with high quality and better format (white background, for example) can be created using the Matplotlib library,

(a)

(b)

Figure 5.12 – Graphical interface for the *setup* vision. It should be used by the manager user group to configure new systems. It relies on the subsystems-separation vision and the electric-signal-ordered elements. a) Gradient subsystem. b) Receiver subsystem. c) Transmitter subsystem.

Source: By the author.

Figure 5.13 – Graphical interface for the *control* vision. This vision should be used to guide the specialist in the control console, at experiment runtime.

Source: By the author.

Sample	System	Sequence	Processing
Composition: <input type="text" value="H2O + CuSo4"/>	Coil Set: <input type="text" value="Solenoid T/R 25"/>	Sequence: <input type="text" value="CPMG"/>	Procedure: <input type="text" value="FFT + Peak Find"/>
Nucleus: <input type="text" value="H1 Nucleus"/>		Protocol: <input type="text" value="Calibration CuSO4"/>	
		Averages: <input type="text" value="10"/>	

Figure 5.14 – Graphical interface for the *fast vision*. This simplified interface is planned to be used by the operator, containing just the essential parameters necessary to the experiment.
Source: By the author.

which can provide publication-quality graphics. The plot view also has data exportation as data file format or image.

Figure 5.15 – Plot view for 2D data. a) Magnetic resonance pulse sequence (spin echo) visualization. The first three subplots represent the RF composition amplitude (red), frequency (green), and phase (blue); the last three show gradient components directions x (red), y (green), and z (blue). b) Signal visualization simulated in MR SPRINT. The first three show the magnetization amplitude for each direction x (red), y (green), and z (blue); then, the total magnetization and phase, relative to the current Larmor frequency, is presented (yellow). The items *a* and *b* are not related in this figure.

Source: By the author.

Graphics tools can improve the user experience and understanding of the system or its data. Thus, included alongside the 2D plot, we offer a 3D plot view, Figure 5.16. This view is focused into the visualization of models (a simple phantom in this case) magnetic field map inside the magnet bore, spin dynamics, magnetic field gradient, and so on. More examples will be given when talking about the MR SPRINT.

5.2.5 Auxiliary views and dialogs

Figure 5.16 – Plot view for 3D data. a) A simple phantom sample is providing three spaced elements with different densities (indicated by transparency). b) Spin dynamics plot for the provided sample, item a, in which are shown the initial magnetization (grey), current total magnetization (white), off-resonance magnetization (magenta and cyan). c) 3D plot menu.

Source: By the author.

Log View

Our framework has a highly integrated logger system that is configured from a JSON file. In the default mode, it logs from warning messages to critical errors. Thus, our log interface has a level filtering, so it gives less or more information about the system. The log-message pattern can include information about the line, file name, time and it can be configured as needed. The handlers take care of displaying the log view and, also, writing in files, which provides crucial information for the developers. Following the recommendations stated in the *Design and methods*, all users can enjoy an efficient system to communicate from inside the program to the outside user. Inserting the code below, the user programming a sequence, or anything else, can communicate properly with the system, using the right way for that and avoiding *prints* along with the code. In the Figure 5.17, in the first item is shown the messages in the standard output, the terminal; the second item shows the graphical log interface, with different message patterns. Pop up dialogs can also be configured.

```
logger.debug("The current average index is: 23")
logger.info("File exported successfully (home/sweet/home/coil.json)")
logger.warning("pinVoltage=5V is too high, setting to maximum of 3V")
logger.error("File exporting error, file in use (home/sweet/home/coil.json)")
logger.critical("The main program was stopped! Segmentation fault.")
```

```

DEBU | simple_fid | setSequence | 110 | The current average index is: 23
INFO | simple_fid | setSequence | 111 | File exported successfully (home/sweet/home/coil.json)
WARN | simple_fid | setSequence | 112 | pinVoltage=5V is too high, setting to maximum of 3V
ERRO | simple_fid | setSequence | 113 | File exporting error, file in use (home/sweet/home/coil.json)
CRIT | simple_fid | setSequence | 114 | The main program was stopped! Segmentation fault.

```

(a)

(b)

Figure 5.17 – Log view with messages for different levels of priority. Letters and colors indicate the level. a) Standard output (terminal or command line). b) Graphical interface output. Levels are represented by the letters and colors, from the low to high, debug (D, blue), information (I, green), warning (W, orange), error and critical error (respectively, E and C, red). The logging system can be easily configured with a JSON file, providing as many outputs and patterns as the user needs.

Source: By the author.

Dialogs

When errors or warnings should require more interaction with the user, or should stop a process and so on, the already existent system of exceptions are used. An exception can be quickly raised, requiring, for example, the attention of the user. Moreover, why it is essential to use that system because you can easily debug the system, create tests, and they are dealt with the right manner by the Python and operational system. Inventing new formats could be hard, time-consuming and dangerous after all those tools were tested and still by millions of people, what will not be reached by simple solutions. The exception system is also easy to use, similar to the logging. The behavior can be controlled from the interface or configuration files in JSON, e.g., if the user does not allow dialogs in its application, the exceptions will be only logged. Otherwise, it will rise adequately pop up dialogs for them. Included to that, for professional users and developers, all of the exceptions can be checked, tested and processed within the IDE, any of them.

5.3 PyMR Spyder Plugin

The PyMR Spyder Plugin implementation is based on Spyder IDE version 3, the latest stable version, and PIP package manager to install our plugin, which will be automatically loaded when started Spyder. Our plugin integrates the PyMR interfaces, shown in the previous section *PyMR framework*, making them straightforward available to the user in Spyder IDE.

Personalized visions, e.g., *setup vision*, are not automatically displayed, however, the user can select one or more to be generated at runtime, in a pop-up style, avoiding overload the GUI. In Figure 5.18, PyMR toolbars are shown integrated into the main toolbar menu, on the top. PyMR windows are highlighted on the right side. In the middle, there is some code opened in the editor. On the right side, the file inspector and outline are opened.

Figure 5.18 – PyMR Spyder Plugin installed and running in Spyder IDE. On the top, PyMR toolbars are shown integrated into the main toolbar menu. PyMR windows are highlighted on the right side. The integration provides highly professional project management, frequently combining well-known technologies included in IDE, like code highlighting, code-completion, inspection, debug, test, and graphical visualization.

Source: By the author.

Again, all those object manager interfaces are generic to support the use of any library. For this reason, a configuration file is required to set up the file paths and library names. The Figure 5.19 exhibits our configuration window, equally integrated into the Spyder IDE. In this case, we are displaying the use of the PyMR framework and the CIERMag framework specialization layer.

Creating magnetic resonance projects using our specialized Spyder IDE should be easy for the users since dialogs and templates guide them. The plugin structure has templates for relaxometry, spectroscopy, and imaging techniques and, for both, systems and sequences, Figure 5.20. Currently, those templates contain fundamental elements that guide the user when generating new projects, providing folders (packages) and files (modules), filling them with regular information as imported libraries and some examples. More clean and specialized templates will be provided when using the structure over time.

Figure 5.19 – Configuration window for PyMR Spyder Plugin. PyMR framework and CIERMag framework specialization layer configuration are exemplified.

Source: By the author.

Figure 5.20 – Project menus and dialogs in PyMR Spyder Plugin. a) Project menu. b) New project dialog. c) Project type drop-down menu from the project dialog.

Source: By the author.

It should be thought that the Spyder IDE interface and functionalities are advanced as they appear to be at the first time. Indeed, the IDE has many features that are not necessary for all user groups we have in the system, but they are essential for developers and advanced ones (R4). For this reason, contexts for each group will be created as exemplified in the PyMR interfaces in the next topic, reflecting the requirements (R8, R9). The interface presented in this section is our earlier version, focused on the system development that soon will have the context modifiers. Code highlight, auto-complete, code check, linting, profiling, and integration with the code versioning raises numerous professional functionalities that are currently neither available nor applied to magnetic resonance systems (R5, R14). Our concept allows ordinarily use of well-known and robust mechanisms, which also can be run in other IDEs, aggregating efficiency, improved development, and quality control, keeping the reasonable simplicity. Thus,

Spyder IDE and our specialized plugin will offer a complete environment for developing new systems and pulse sequences.

5.4 MR SPRINT

The MR SPRINT a short name for *magnetic resonance experiment simulator and visualization tool* have been started to be developed in parallel with the PyMR. It was initially used to test specific simulations. Then it was added ideas of being an educational tool to explain magnetic resonance phenomena in a whole experiment. As the PyMR is registered and already licensed, we cannot publish together with the MR SPRINT. On the other hand, we have used most of the structure of MR SPRINT to produce the new visions the PyMR, that was already shown in the previous section. In this section, we will present MR SPRINT main functionalities along with a simple simulation of FID acquisition. The main window of the MR SPRINT has in the middle a 2D or 3D view, depending on the context. On the right side, it has a 2D editor for the matrices (3D sliced matrices). On the left, there is the context menu and the actions for that context. At the top, all actions for the viewers and editors are available.

5.4.1 Sample context

In this context, the user can select the sample from the list of available systems or create its sample. The user can select or create a sample by editing the 2D sliced image, Figure 5.21 right. Also, it is possible to navigate in the 3D matrix that represents the sample. In the middle, there is a 3D view. Each color represents one parameter in the sample. Blue is the density of spins. Red is T1. Green is T2.

5.4.2 System context

In the system context, the user can load or create a magnetic field map. In the current case, Figure 5.22, the system have no inhomogeneity.

5.4.3 Sequence context

At this moment in the interface, it is not possible to create a sequence. The user can write it in a Python file and then load in the interface. All available sequences will appear in the navigator Figure 5.23 (bottom left). However, the user can see the loaded sequence in the 2D view, as already mentioned. Following we will see an example to set a sequence.

Figure 5.21 – Sample context in the MR SPRINT. In the top-left it is shown the parameters that can be changed for each element or for the whole sample. In bottom-left it is listed the samples available. At the center, the 3D sample is exhibited. In the right there is an element editor in a spreadsheet format.

Source: By the author.

Figure 5.22 – System context in the MR SPRINT. On this interface, the user can select a field map to be loaded, like a magnetic field map. In this case, there is no inhomogeneity in the magnet. In the top-left it is shown the parameters that can be changed for each element or for the whole sample. In bottom-left it is listed the samples available. At the center, the 3D inhomogeneity field is exhibited. In the right there is an element editor in a spreadsheet format.

Source: By the author.

Figure 5.23 – Sequence context in the MR SPRINT. The user can select a previously created sequence to run from the navigator. The 2D view shows the full sequence, in which each line plot represents an channel in the system. This case it is shown the sequence bellow, a simple FID acquisition. The parameters of the sequence can be changed from the interface (top-left).

Source: By the author.

A Python file defines the sequence. The class is derived from the *Sequence* class. In the *init* we declare the parameters we want to use in the interface. Other ones can be created to help the organization, as the *p90x* and *acq*.

```
class FIDSequence(Sequence):
    """Class of a FID acquisition sequence."""

    def __init__(self):
        super().__init__()
        # defining one event of the sequence manually
        # using the sequence parameters that automatically
        # appears on the GUI, with no programming
        # initializing sequence parameters

        self.flipAngle = 90.0 _unit.deg
        self.delayPre = 1 * _unit.ms
        self.acquisition = 100 * _unit.ms
        self.delayPos = 10 * _unit.ms

        # definitions of main pulse and acquisition
        self.p90x = None
        self.acq = None

        # initializing rf and gradient systems
        self.rf = None
        self.gradient = None
```

The validation method validates the state of the instance before actions. There is an example of value validation in which we have used both the logging and exception system. The

first one will not stop the execution, and it also sets the value to a permitted one but warning the user. The second can raise an exception that, depending on the system, should stop the execution.

```
def validate(self):
    """Validate sequence parameters."""

    # some dummy example of validations
    min_acq_time = 0.01 * _unit.ms
    max_acq_time = 10.0 * _unit.s

    # this should only warn the user and continues the execution
    if self.acquisition > max_acq_time:
        self.acquisition = max_acq_time
        logger.warning("Acquisition is too long, using maximum: ", max_acq_time)

    # this should stop the execution
    if self.acquisition < min_acq_time:
        raise ValueError("Acquisition time is too short!")
```

In the method *set* the user can associate values to variables already created in the *init* (or local ones, of course). Also, an example of debugging is given.

```
def set(self):
    """Set the sequence for FID acquisition."""

    # definitions of main pulse and acquisition
    self.p90x = pulse.SquareRFPulse(flip_angle=self.flipAngle, label='Excitation')
    self.acq = pulse.RFACquisition(time=self.acquisition, label='Acquisition')

    # logging debug information
    logger.debug("p90x shape: ", self.p90x.shape)
    logger.debug("acq shape: ", self.acq.shape)
```

Then, the *run* method will execute the sequence. Inside it has methods to *pulse*, *acquire*, and *wait*; the last one to insert delays. Inside the *run*, the user can use loops (*for*).

```
self.run(self):
    """Run the sequence already set."""

    self.pulse(self.p90x)
    self.wait(self.delayPre)
    self.acquire(self.acquisition)
    self.wait(self.delayPos)
```

Finally, in the *plot* function, the user has access to the data acquired, and they can process and return to a new plot window.

```
def plot(self):
    """Return result data for plotting."""

    # data can be plot automatically,
    # however it will plot all points
    data_plot = {'data': self.results.real,
                 'x_label': "Time (s)",
                 'y_label': "Amp. Real (a.u.)"}
```

(continues on next page)

(continued from previous page)

```
# example of processed data plot
proc_plot = {'data': np.fft(self.results).real,
            'x_label': "Frequency (Hz)",
            'y_label': "Amplitude (a.u.)"}

return data_plot, proc_plot
```

5.4.4 Simulator context

In the simulator context, all previously selected values are sent to the Bloch Simulator to evolve the magnetization. In the end, the user can see the running simulation. In the middle, Figure 5.24, there is a 3D image of the magnetization vectors evolving. During the exhibition, the user can also see the values for the magnetization. However, as noted, there is no oscillation as we are using the rotating frame of reference.

Figure 5.24 – Simulator context in the MR SPRINT. The user can see the simulation running step by step. At the center we show to magnetization voxels. The translucent line indicates the initial magnetization. The white line is the total magnetization at the current time and the two lines, magenta and cyan, represents the isocromates. In the right side, it is shown the magnetization values over the time for each direction, including the total magnetization and its phase.

Source: By the author.

5.4.5 Processing context

In the last one, processing, the user can process data received from the simulator. Up to know it is also necessary to program the processing.

Figure 5.25 – Processing context in the MR SPRINT. The processing is executed on data received from the simulator. The graphics are also interchangeable. The main plot indicates the Fourier transform for simulated signal, showing just one correspondent peak of a single frequency.
Source: By the author.

In the near future we will use the main interface of MR SPRINT to the PyMR console controller. The disposal of menus, icons, and other features have been pleasing many users. For a didactic tool, it shows interesting viewers for the magnetization, the timeline the student can follow the within the sequence, the 3D view of spins systems.

6 CONCLUSION

This project provided PyMR framework for magnetic resonance systems, planned and implemented to provide a versatile and innovative tool to design new systems and sequence, which shall create numerous new applications - PyMR is already being integrated of two new projects, see *Appendix H - Production and projects*. It overtakes magnetic resonance programming to a new and extensive context of a software project, without embedding complex concepts or requiring in-depth knowledge. Abstraction layers created in the class and framework level keep each user group focused on their respective activity, demanding only the layer know-how.

The graphical interfaces and structure employed to our framework presented a fast initial implementation to a new system and a user-friendly interaction. Templating a new interface all the elements can be connected automatically, supplying high powerful editors in an accessible format. Our implementation covered most of all requirements and literature deficiencies in the interfaces. Easy navigation, changeable, resourceful and high integrable. All types of users will have the best-fit interface for them.

The programming language Python with its built-in functionalities as exceptions and logging provided a right concept of usage. It also affected the code readability and integration with a vast environment of scientific libraries, and yet, a scalable solution. Using an established language, the user can debug, test or implement code using any library available. Using graph techniques and visualization introduced a natural form to validate the structure with general rules, without the need for programming or implementing the code. The range of this application should be deeply investigated, may it can contribute to general software issues.

Those results also have two way to contribute, simplifying the code structure, usage, and coding and also inserting essential tools to the context of magnetic resonance. Those contributions allow the use of secure computer science techniques and tools in another context. Version control system, for example, can help most of them including the sequence development, keep track of changes, history, and evolution. The integrated development environment can, now, be used to improve the code quality and development in the magnetic resonance context. Our plugin for the Spyder IDE already integrated the PyMR in a graphical format with it. Soon, new functionalities will be included.

Our implementation is not focused on working on pre-existent commercial tools, as this involves much effort and law-related issues. Whereas our framework used in all our systems that

are spreading along with the scientific community. Adding, our graphical interface and the front end should integrate existing and beneficial tools, such as ODIN, JEMRIS, DAMARIS.

Finally, our simulator, MR SPRINT, will provide an educational tool for demonstrating the magnetic resonance concepts and how the system works. Its use merged with the PyMR framework provides a new environment for tests. Also, more quantitative results should come with the implementation of new systems and projecting new test suites. Following we list the main contributions created from this project to the magnetic resonance field and computer science. By the end, we list new perspectives for future development.

6.1 Contributions

- In the magnetic resonance system field:
 - Integrable, scalable, easy to use and modern framework;
 - User-friendly graphical interfaces for the whole users;
 - Consolidation of the robust approach and all-embracing of pulse sequences and system parts as software projects;
 - Consolidation of use of IDE to build, design and test new pulse sequences;
 - A didactic simulator, MR SPRINT, which will provide the first version of a new controller.
- Computer science and the scientific community:
 - Proposal for a new syntax that can include physical quantity in an explicit and short format (*Appendix G - Proposal for a new variable type*);
 - Proposal for documenting physical quantities in the documentation - docstrings (*Appendix F - Documenting physical quantities*);
 - Contributions in many software initiatives having proposed new applications, new features and bug fixes (*Appendix I - Other contributions by the author*).
 - Study the applicability of graph theory for check system consistency;

6.2 Perspectives

- Create user group contexts for the Spyder IDE, which means personalizing the graphical interface for each user group level;

- Integrate all system components (geometry editor, peripheral manager, device manager) that are currently being developed by the group;
- Planning the integrating with new libraries that can support our necessities;
- Integration with those main tools in the area, such as ODIN and JEMRIS;
- Python integrated more closely to the Digital Spectrometer (Python interpreter inside);
- Run PyMR interconnected with Arduino for educational or simplified systems.

REFERENCES

- 1 BLOCH, F. Nuclear induction. **Physics Review**. v. 70, n. 460, p. 7-8, 1946.
- 2 VAN GEUNS, R-J. M. et al. Basic principles of magnetic resonance imaging. **Progress in Cardiovascular Diseases**, v. 42, n. 2, p. 149–156, 1999. doi:10.1016/S0033-0620(99)70014-9.
- 3 HAGE, M. C. F. N. S.; IWASAKI, M. Imagem por ressonância magnética: princípios básicos. **Ciência Rural**, v. 39, n. 4, p. 1275–1283, 2009. doi:10.1590/S0103-84782009005000041.
- 4 SLICHTER, C. P. **Principles of magnetic resonance**. Berlin: Springer, 1990. (Springer series in solid-state sciences, v. 1)
- 5 AI, T. et al. A historical overview of magnetic resonance imaging, focusing on technological innovations. **Investigative Radiology**, v. 47, n. 12, p. 725–741, 2012. doi:10.1097/RLI.0b013e318272d29f.
- 6 JOHANSSON, A. Magnetic resonance imaging with ultrashort echo time as a substitute for X-ray computed tomography. 2014. Available from: <<http://www.diva-portal.org/smash/record.jsf?pid=diva2:760894>>. Accessible at: 23 Jan. 2017.
- 7 DILL, T. Contraindications to magnetic resonance imaging: non-invasive imaging. **Heart (British Cardiac Society)**, v. 94, n. 7, p. 943–948, 2008. doi:10.1136/hrt.2007.125039.
- 8 KEUPP, J. et al. Parallel RF transmission based MRI technique for highly sensitive detection of amide proton transfer in the human brain. 2011. Available from: <<http://cds.ismrm.org/protected/11MProceedings/files/710.pdf>>. Accessible at: 31 May. 2018.
- 9 HEIDEMANN, R. M. et al. A brief review of parallel magnetic resonance imaging. **European Radiology**, v.13, n. 10, p. 2323–2337, Oct. 2003. doi:10.1007/s00330-003-1992-7.
- 10 KATSCHER, U.; BÖRNERT, P. Parallel RF transmission in MRI. **NMR in Biomedicine**, v.19, n. 3, p. 393–400, May 2006. doi:10.1002/nbm.1049.
- 11 HAËN, C. Conception of the first magnetic resonance imaging contrast agents: a brief history. **Topics in Magnetic Resonance Imaging**, v.12, n. 4, p. 221–230, Aug. 2001.
- 12 TANNÚS, A. **Desenvolvimento da tecnologia de tomografia por ressonância magnética nuclear - TORM**. 1987. 227 f. Tese (Doutorado) – Instituto de Física e Química de São Carlos, Universidade de São Paulo, São Carlos, 1987.
- 13 MARTINS, M. J. **Desenvolvimento de um tomógrafo de ressonância magnética: integração e otimização**. 1995. 78 p. Tese (Doutorado) – Instituto de Física e Química de São Carlos, Universidade de São Paulo, São Carlos, 1995.

- 14 DESHMANE, A. et al. Parallel MR imaging. **Journal of Magnetic Resonance Imaging**, v. 36, n. 1, p. 55–72, July 2012. doi:10.1002/jmri.23639.
- 15 BUXTON, R. B. **Introduction to functional magnetic resonance imaging**: principles and techniques. Cambridge: Cambridge University Press, 2002.
- 16 GULTEKIN, D. H.; GORE, J. C. Measurement of thermal diffusivity by magnetic resonance imaging. **Magnetic Resonance Imaging**, v. 24, n.9, p. 1203–1207, Nov. 2006. doi:10.1016/j.mri.2006.03.014.
- 17 GEVA, T. Magnetic resonance imaging: historical perspective. **Journal of Cardiovascular Magnetic Resonance**, v. 8, n. 4, p. 573–580, 2006. doi: 10.1080/10976640600755302.
- 18 GOUD, S.; HU, Y.; JACOB, M. Real-time cardiac MRI using low-rank and sparsity penalties. In: IEEE INTERNATIONAL SYMPOSIUM ON BIOMEDICAL IMAGING: FROM NANO TO MACRO, 2010., Rotterdam. **Proceedings...** Rotterdam, Netherlands: IEEE, 2010. p. 988–991 Available from: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5490154>>. Accessible at: 23 Sept. 2018.
- 19 BLAIMER, M. et al. SMASH, SENSE, PILS, GRAPPA. *Topics in Magnetic Resonance Imaging*, v. 15, n. 4, p. 223–236, Aug. 2004. doi:10.1097/01.rmr.0000136558.09801.dd.
- 20 GALANTE, A. et al. Fast room temperature very low field-magnetic resonance imaging system compatible with magnetoencephalography environment. **PLOS ONE**, v. 10, n. 12, p. e0142701, Dec. 2015. doi:10.1371/journal.pone.0142701.
- 21 TALANTSEV, E. F. et al. The onset of dissipation in high-temperature superconductors: magnetic hysteresis and field dependence. **Scientific Reports**, v. 8, n. 1, p. 14463, Dec. 2018. doi:10.1038/s41598-018-32811-6.
- 22 ZIMMERMAN, C. L. **Low-field classroom nuclear magnetic resonance system. 2010.** 109 p. (Master thesis) - Department of Electrical Engineering and Computer, Massachusetts Institute of Technology, Massachusetts, 2010. Available from: <<http://web.mit.edu/~20.309/www/Zimmerman2010.pdf>>. Accessible at: 30 Jan. 2018.
- 23 FIT: fine instrument technology. 2018. Available from: <www.fitinstrument.com/FITbr/FineInstrumentTechnology.html>. Accessible at: 07 Jan. 2018
- 24 GARWOOD, M. G.; VAUGHAN, J. T. Imaging Brain Function in Real World Environments & Populations with Portable MRI. Available from: <<http://grantome.com/grant/NIH/R24-MH105998-02S1>>. Accessible at: 17 July 2018
- 25 CARDOSO, C. **Estudo da morfologia de rochas sintéticas através de imagens por Ressonância Magnética.** 2018. 74 p. Dissertação (Mestrado em Física Aplicada) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2018.

- 26 COLNAGO, L. A.; ANDRADE, F. B.; SOUZA, A. A. et al. Why is inline NMR rarely used as industrial sensor? challenges and opportunities. **Chemical Engineering and Technology**, v. 37, n. 2, p. 191–203, 2014. doi:10.1002/ceat.201300380.
- 27 CONSALTER, D. M. **Implementação de aquisição paralelas de imagens utilizando bobinas de RF tipo phase array e sampled array**. 2017. 119 p. Tese (Doutorado em Física Aplicada) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2017.
- 28 PIZETTA, D. C. **Biblioteca, API e IDE para o desenvolvimento de projetos de metodologias de Ressonância Magnética**. 2014. Dissertação (Mestrado em Física Aplicada) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2014.
- 29 JOCHIMSEN, T. H.; MENGERSHAUSEN, M. von. ODIN-object-oriented development interface for NMR. **Journal of Magnetic Resonance**, v. 170, n. 1, p. 67–78, 2004. doi:10.1016/j.jmr.2004.05.021
- 30 GÄDKE, A. et al. DAMARIS — a flexible and open software platform for NMR spectrometer control. **Magnetic Resonance Imaging**, v. 25, n. 4, p. 576, 2007. doi:10.1016/j.mri.2007.01.081.
- 31 STECKNER, M. C. Advances in MRI equipment design, software, and imaging procedures. medical systems American. 2006. Available from: <<https://www.aapm.org/meetings/amos2/pdf/26-5961-46702-744.pdf>>. Accessible at: 07 July 2018.
- 32 MAGLAND, J. F.; LI, C.; LANGHAM, M. C. et al. Pulse sequence programming in a dynamic visual environment: sequence tree. **Magnetic Resonance in Medicine**, v. 75, n. 1, p. 257–265, 2016. doi:10.1002/mrm.25640.
- 33 LAYTON, K. J. et al. Pulseq: a rapid and hardware-independent pulse sequence prototyping framework. **Magnetic Resonance in Medicine**, v. 77, n. 4, p. 1544–1552, 2017. doi:10.1002/mrm.26235.
- 34 RAVI, K. S. et. al. Pulseq-graphical programming interface: open source visual environment for prototyping pulse sequences and integrated magnetic resonance imaging algorithm development. **Magnetic Resonance Imaging**, v. 52, p. 9–15, 2018. doi:10.1016/j.mri.2018.03.008.
- 35 NIELSEN, J. F.; NOLL, D. C. TOPPE: a framework for rapid prototyping of MR pulse sequences. **Magnetic Resonance in Medicine**, v. 79; n. 6; p. 3128–3134, 2018. doi:10.1002/mrm.26990.
- 36 MICHAL, C. A. A low-cost spectrometer for NMR measurements in the Earth's magnetic field. **Measurement Science and Technology**, v. 21, n. 10, p. 105902, 2010. doi:10.1088/0957-0233/21/10/105902.
- 37 BAUM, K. G.; MENEZES, G.; HELGUERA, M. Simulation of High-Resolution Magnetic Resonance Images on the IBM Blue Gene/L Supercomputer Using SIMRI. **International Journal of Biomedical Imaging**, v. 2011, p. 1–8, 2011. doi:10.1155/2011/305968.

- 38 SMITH, S. A.; LEVANTE, T. O.; MEIER, B. H. et al. Computer simulations in magnetic resonance. An object oriented programming approach. **Journal Magnetic Resonance A**, v. 106, n. 1, p. 75–105, 1994. doi:10.1006/jmra.1994.1008.
- 39 HELMUS, J. J.; JARONIEC, C. P. Nmrglue: an open source Python package for the analysis of multidimensional NMR data. **Journal of Biomolecular NMR**, v. 55, n. 4, p. 355–367, 2013. doi:10.1007/s10858-013-9718-x.
- 40 VAHEDIPOUR, K.; SHAH, N. J.; STÖCKER, T. Fast MR simulations with JEMRIS 2.1 - Disclosing the secrets of MRI sequence development. **International Society for Magnetic Resonance in Medicine**, v. 17, p. 3039, 2009. Available from: <<https://cds.ismrm.org/protected/09MPProceedings/files/03039.pdf>>. Accessible at: 18 May. 2018
- 41 STÖCKER, T.; VAHEDIPOUR, K.; PFLUGFELDER, D. et al. High-performance computing MRI simulations. **Magnetic Resonance in Medicine**, v. 64, n. 1, p. 186–193, 2010. doi:10.1002/mrm.22406.
- 42 HACKLÄNDER, T.; MERTENS, H. Virtual MRI: A PC-based simulation of a clinical MR scanner. **Academic Radiology**, v. 12, n. 1, p. 85–96, 2005. doi:10.1016/j.acra.2004.09.011.
- 43 LIU, F. et al. Fast realistic MRI simulations based on generalized multi-pool exchange tissue model. **IEEE Transactions on Medical Imaging**, v. 36, n. 2, p. 527–537, 2017. doi:10.1109/TMI.2016.2620961.
- 44 KOSE, R.; KOSE, K. BlochSolver: a GPU-optimized fast 3D MRI simulator for experimentally compatible pulse sequences. **Journal of Magnetic Resonance**, v. 281, p. 51–65, 2017. doi:10.1016/j.jmr.2017.05.007.
- 45 HACKDAY.IO. 2018. Available from: <<https://hackaday.io/project/1376-pypppm-a-proton-precession-magnetometer-for-all>>. Accessible at: 10 July 2018.
- 46 BELMONTE, S. B.; SARTHOUR, R. S.; OLIVEIRA, I.S. et al. A field-programmable gatearray-based high-resolution pulse programmer. **Measurement Science and Technology**, v. 14, n. 1, p. N1–N4, 2003.
- 47 SNYDER, A. L. S. et al. MRI by steering resonance through space. **Magnetic Resonance in Medicine**, v. 72, n. 1, p. 49–58, 2014. doi:10.1002/mrm.24888.
- 48 ZHANG, J. et al. Gradient modulated SWIFT. **Magnetic Resonance in Medicine**, v. 75, n. 2, p. 537–546, 2016. doi: 10.1002/mrm.25595.
- 49 UNIVERSIDADE DE SÃO PAULO. Instituto de Física de São Carlos. Mateus José Martins; Edson Luiz Gea Vidoto; Alberto Tannús. **Espectrômetro para uso em sistemas de Ressonância Magnética**. BR1020150006241, 12 jan. 2015.
- 50 SILVA, D. M. D. **Desenvolvimento de console multiplataforma para aquisição, organização e visualização de dados do espectrômetro digital de RM do CIERMag: ToRM Console**. 2014. 145p. Dissertação (Mestrado em Física Aplicada) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2014.

- 51 TANNÚS, A.; PIZETTA, D. C.; SILVA, D.M.D.D; VIDOTO, E. L. G; LOURENÇO, G. V.; MARTINS, M. J. Revista da Propriedade Industrial - RPI n.2365, p.351, item 120. **Registro de software:** ToRM-IDE - Integrated development environment for magnetic resonance applications. Processo: BR512015001484-6. Data de registro: 07 dez. 2015.
- 52 TANNÚS, A.; PIZETTA, D. C.; SILVA, D.M.D.D; VIDOTO, E.L.G; MARTINS, M. J. Revista da Propriedade Industrial - RPI n.2365, p.350, item 120. **Registro de software:** Subsistema multiplataforma para controle de aquisição, visualização e organização de dados de espectrômetro digital de RM: ToRM Console. Processo: BR512015001485-4. Data de registro: 07 dez. 2015.
- 53 HANSON, L. G. **Introduction to magnetic resonance imaging techniques**. 2009. Available from: <http://www.drcmr.dk/Docs/MRI_English_a4.pdf>. Accessible at: 23 Jan. 2018.
- 54 BROWN, R. W.; CHENG, Y. N; HAACKKE, E. M. et al. **Magnetic resonance imaging: physical principles and sequence design**. New York: John Wiley & Sons, 2014. 944 p.
- 55 PANEPUCCI, H.; TANNÚS, A. Magnetic resonance imaging. tutorial I. 1994. Available from: <http://perso.ensta-paristech.fr/~manzaner/Cours/OLD_Ese21/Docs_IRM/MRITutor.pdf>. Accessible at: 05 Oct. 2018.
- 56 MA, D. et al. Magnetic resonance fingerprinting. **Nature**, v. 495, n. 7440, p. 187–192, 2013. doi:10.1038/nature11971.
- 57 CANESE, R.; PODO, F. Introduction to magnetic resonance for clinical use. physical principles and instrumentation. **Annali dell'Istituto Superiore di Sanità**, v. 30, n. 1, p.7–29, 1994.
- 58 LAKRIMI, M. et al. The principles and evolution of magnetic resonance imaging. **Journal of Physics: conference series**, v. 286, n. 1, 2011. doi:10.1088/1742-6596/286/1/012016.
- 59 TIROTTA, I. et al. ¹⁹(F) magnetic resonance imaging (MRI): from design of materials to clinical applications. **Chemical Reviews**, v. 115, n. 2, p.1106-29, 2015. doi: 10.1021/cr500286d.
- 60 VESPA: versatile simulation, pulses and analysis for magnetic renosance spectroscopy. 2018. Available from: <<http://scion.duhs.duke.edu/vespa/project>>. Accessible at: 07 Jan. 2018.
- 61 GAMMA: general approach to magnetic resonance mathematical analysis. GitHub. 2018. Available from: <<https://github.com/tesch1/GAMMA>>. Accessible at: 07 Jan. 2018.
- 62 DELAGLIO, F. et al. NMRPipe: a multidimensional spectral processing system based on UNIX pipes. **Journal of Biomolecular NMR**, v. 6, n. 3, p. 277–293, 1995. doi:10.1007/BF00197809.
- 63 NMRGULE: a module for working with NMR data in python. 2018. Available from: <<https://www.nmrglue.com/>>. Accessible at: 17 Sep. 2018.

- 64 DAMARIS. 2018. Available from: <https://element.fkp.physik.tu-darmstadt.de/damaris_cms/>. Accessible at: 07 Jan. 2018.
- 65 ODIN: object-oriented development interface for NMR. 2018. Available from: <<http://www.opensourceimaging.org/project/odin-object-oriented-development-interface-for-nmr/>>. Accessible at: 18 Sep. 2018.
- 66 HASS. Open source imaging Jeremis. 2018. Available from: <<http://www.opensourceimaging.org/project/jemris>>. Accessible at: 07 Jan. 2018.
- 67 JI, J. X.; SON, J. B.; RANE, S. D. PULSAR: a Matlab toolbox for parallel magnetic resonance imaging using array coils and multiple channel receivers. **Concepts in Magnetic Resonance Part B: magnetic resonance engineering**, v. 31B, n. 1, p. 24–36, 2007. doi:10.1002/cmr.b.20081.
- 68 MRILAB. 2018. Available from: <<http://mrilab.sourceforge.net/>>. Accessible at: 07 Jan. 2018.
- 69 TOPPE: a framework for rapid prototyping of MR pulse sequences. 2018. Available from: <<https://toppemri.github.io/6>>. Accessible at: 07 Jan. 2018.
- 70 CATTIN, H. B. et al. The SIMRI project: a versatile and interactive MRI simulator. **Journal of Magnetic Resonance**, v. 173, n.1, p. 97–115, 2005. doi.org/10.1016/j.jmr.2004.09.027.
- 71 SIMRI: a versatile and interactive magnetic resonance imaging (MRI) simulator. 2018. Available from: <<https://sourceforge.net/projects/simri/>>. Accessible at: 18 Jan. 2018.
- 72 DEBBINS, J. et al. Novel software architecture for rapid development of magnetic resonance applications. **Concepts in Magnetic Resonance**, v. 15, n. 3, p. 216–237, 2002. doi:10.1002/cmr.10039.
- 73 WRIGHT, S. M. et al. A desktop magnetic resonance imaging system. *Magma*, v. 13, n. 3, p. 177–85, 2002.
- 74 HASSELWANDER, C. J.; CAO, Z.; GRISSOM, W. A. gr-MRI: a software package for magnetic resonance imaging using software defined radios. **Journal of Magnetic Resonance**, v. 270, n. 11, p. 47–55, 2016. doi:10.1016/j.jmr.2016.06.023.
- 75 SFETSOS, P. et al. A comparative study on the effectiveness of patterns in software libraries and standalone applications. In: INTERNATIONAL CONFERENCE ON THE QUALITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY (QUATIC). 2014. Available from: <<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6984106>>. Accessible at: 23 Jan. 2017.
- 76 PRESSMAN, R. **Software engineering: a practitioner's approach**. McGraw-Hill, e. 7, 2010. New York, NY, USA.
- 77 NIELSEN, J. Usability inspection methods. In: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS (CHI'94), 1994. New York: ACM, 1994.

Available from: <<http://portal.acm.org/citation.cfm?doid=259963.260531>>. Accessible at: 13 Mar. 2014.

78 SEARS, A.; JACKO, J. A. **The human-computer interaction handbook: fundamentals, evolving technologies, and emerging applications**. Mahwah, New Jersey: Lawrence Erlbaum Associates, 2003. ISBN 0805844686.

79 KITCHENHAM, B. A.; DYBA, T.; JORGENSEN, M. Evidence-Based Software Engineering. 2004. Available from: <<http://dl.acm.org/citation.cfm?id=998675.999432>>. Accessible at: 03 Aug. 2018.

80 TULACH, J. **Practical API design: confessions of a java framework architect**. Berkeley: Apress, 2008. 417 p.

81 BLANCHETTE, J. **The little manual of API design**. 2008. Available from: <<https://people.mpi-inf.mpg.de/~jblanche/api-design.pdf>>. Accessible at: 23 Jan. 2017.

82 RONACHER, A. **Good API design: study, improve & create**. 2010. Available from: <<http://dev.pocoo.org/~mitsuhiko/API.pdf>>. Accessible at: 13 Apr. 2014.

83 PYTHON. 2018. Available from: <<https://www.python.org/>>. Accessible at: 19 Jun. 2018.

84 TIOBE: the software quality company. 2018. Available from: <<https://www.tiobe.com/tiobe-index/>>. Accessible at: 31 Jan. 2018.

85 IEEE SPECTRUM. 2018. Available from: <<https://spectrum-ieee-org.ez67.periodicos.capes.gov.br/at-work/innovation/the-2018-top-programminglanguages>>. Accessible at: 07 Feb. 2018.

86 PYPL: popularity of programming language. 2018. Available from: <<http://pypl.github.io/PYPL.html>>. Accessible at: 27 Mar. 2018.

87 IBM. 2018. Available from: <https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en>. Accessible at: 25 Apr. 2018.

88 TOWARDS DATA SCIENCE. 2018. Available from: <<https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-le>>. Accessible at: 30 Jan. 2018.

89 NUMPY. 2018. Available from: <<http://www.numpy.org/>>. Accessible at: 30 Jan. 2018.

90 SCIPY. 2018. Available from: <<https://www.scipy.org/>>. Accessible at: 17 Jul. 2018.

91 SPHINX. 2018. Available from: <<http://www.sphinx-doc.org/en/master/>>. Accessible at: 17 July 2018.

- 92 QT: software development made smarter. 2018. Available from: <<https://www.qt.io/>>. Accessible at: 07 Jan. 2018.
- 93 PYQT. 2018. Available from: <<https://wiki.python.org.br/PyQt>>. Accessible at: 01 June 2018.
- 94 QT pyside: software development made smarter. 2018. Available from: < <https://wiki.qt.io/PySide>>. Accessible at: 06 Jan. 2018.
- 95 QTPY. GitHub: new behavior when start import qtpy - setting qt binding. Available from: <<https://github.com/spyder-ide/qtpy>>. Accessible at: 05 Apr. 2018.
- 96 PYQTGRAPH: scientific graphics and gui library for python. 2018. / Available from: <<http://www.pyqtgraph.org>>. Accessible at: 15 May. 2018.
- 97 AUTODESK. 2018. Available from: <<https://www.autodesk.com.br/products/maya/overview>>. Accessible at: 17 Jun. 2018.
- 98 MATPLOTLIB. 2018. Available from: < <https://matplotlib.org/>>. Accessible at: 17 June 2018.
- 99 PYTHON ROPE. GITHUB. 2018. Available from: <<https://github.com/python-rope/rope>>. Accessible at: 11 May.
- 100 JSON. 2018. Available from: < <https://www.json.org/>>. Accessible at: 11 May 2018.
- 101 YML. 2018. Available from: <<http://yaml.org/>>. Accessible at: 18 Mar. 2018.
- 102 W3C. 2018. Available from: < <https://www.w3.org/>>. Accessible at: 05 June 2018.
- 103 HDF5: support the HDF group. 2018. Available from: <<https://support.hdfgroup.org/HDF5/>>. Accessible at: 15 June 2018.
- 104 LAROBINA, M.; MURINO, L. Medical image file formats. **Journal of Digital Imaging**, v. 27, n. 2, p. 200–206, Apr. 2014. doi:10.1007/s10278-013-9657-9.
- 105 INATI, S. J. et al. ISMRM Raw data format: a proposed standard for MRI raw datasets. **Magnetic Resonance in Medicine**, v. 77, n. 1, p. 411–421, Jan. 2017. doi:10.1002/mrm.26089.
- 106 EE Times. **Ide evolution continues beyond eclipse**. 2018. Available from: <https://www.eetimes.com/document.asp?doc_id=1150355>. Accessible at: 23 Aug. 2018.
- 107 SOUZA, P. V. B. D. **Desenvolvimento de um subsistema non-real-time para o gerenciamento de dispositivos periféricos e desenvolvimento de interfaces gráficas**. 2016. 122p. Dissertação (Mestrado em Física Básica) – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2016.

- 108 COELHO, F. B. **Controlador para bobinas de shimming via HTTP**. 2016. 68p. Dissertação (Mestrado em Física Básica) – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2016.
- 109 SCRUM.ORG: the home of scrum. 2018. Available from: <<https://www.scrum.org/>> Accessible at: 17 Jul. 2018.
- 110 BOEHM, B. W. Software process: emerging extensions. **Software Engineering**, p. 535–543. 2011. doi:10.1109/9780470187562.ch6.
- 111 GIT. 2018. Available from: <<https://git-scm.com/>>. Accessible at: 23 Jul. 2018.
- 112 MERCURIAL. 2018. Available from: <<https://www.mercurial-scm.org/>> Accessible at: 31 Jan. 2018.
- 113 CIRCLECI: the shortest distance from idea to execution. 2018. Available from: <<https://circleci.com/>>. Accessible at: 09 Sep. 2018.
- 114 JEKINS: build great things at any scale. 2018. Available from: <<https://jenkins.io/>>. Accessible at: 07 Aug. 2018.
- 115 SEMANTIC version 2.0.0. 2018. Available from: <<https://semver.org/>> Accessible at: 31 Jan. 2018.
- 116 PIZETTA, D. C.; WANDERLEY, A. B. **Standing sound waves** **GitHub**. 2016. Available from: <<https://github.com/dpizetta/standing-sound-waves>>. Accessible at: 15 June 2018.
- 117 PIZETTA, D. C. et al. Uma avaliação experimental do tubo de ondas sonoras estacionárias. **Revista Brasileira de Ensino de Física**, v. 39, n. 3, p. 3301, 2017. doi:10.1590/18069126-rbef-2016-0264.
- 118 PIZETTA, D. C.; DAFLITA, W. **Wavy tool**: simple GUI that acquires data from input devices, plot and export files. **GitHub**. 2015. Available from: <<https://gitlab.com/dpizetta/wavytool>, doi:10.5281/zenodo.809476.>. Accessible at: 23 Sept. 2018.
- 119 PIZETTA, D. C.; RONQUI, J. R. F.; CAMPOS, T. **SciCorpus**: scientific corpus manager. **GitHub**. 2015. Available from: <<https://github.com/zericardo/sci-corpus>>. Accessible at: 15 Jun. 2018.
- 120 THOMPSON, A.; TAYLOR, B. N. **Guide for the use of the international system of Units (SI)**. **NIST special publication**. n. 811. 2008. Available from: <<https://physics.nist.gov/cuu/pdf/sp811.pdf>>. Accessible at: 20 Oct. 2018.
- 121 THE NIST REFERENCE ON CONSTANTS, UNITS AND UNCERTAINTY. Available from: <<https://physics.nist.gov/cuu/Units/>>. Accessible at: 20 Oct. 2018.
- 122 SCIPY. 2018. Available from: <<https://www.scipy.org/>>. Accessible at: 17 July 2018.

123 TANNÚS, A. et al. PyMR - a framework for programming magnetic resonance systems. Process number: CC-PC-2017-0035. Registry request: 31 Aug. 2017. Process in progress.

124 PIZETTA, D. C. et al. Magnetic resonance system configuration and editing tools. In: IUPESM WORLD CONGRESS ON MEDICAL PHYSICS & BIOMEDICAL ENGINEERING, 2015. Available from: <<http://wc2015.org/wp-content/uploads/2015/06/IUPESM2015-Onsite-Program-and-Abstract-Book.pdf>>. Accessible at: 15 Apr. 2018.

125 PIZETTA, D. C.; PESSOA, V. H. M.; PAIVA, F .F. Simulating magnetic resonance experiments with Precession. In: SIMPÓSIO DE INSTRUMENTAÇÃO E IMAGENS MÉDICAS (SIIM), 8; SIMPÓSIO DE PROCESSAMENTO DE SINAIS (SPS), 7, 2017. Santo André. **Anais...** (electronic) ID 169.

126 PIZETTA, D.C.; SILVA, L.C.R.; VIDOTO, E.L.G.; MARTINS, M.J.; TANNÚS, A. A framework for programming a modern magnetic resonance system. In: SIMPÓSIO DE INSTRUMENTAÇÃO E IMAGENS MÉDICAS (SIIM), 8; SIMPÓSIO DE PROCESSAMENTO DE SINAIS (SPS), 7, 2017. Santo André. **Anais...** (electronic) ID 178. 126.

APPENDIX A - ANALOGIES OF MAGNETIC RESONANCE SYSTEMS

Understanding the relationship between all elements that compound a generic magnetic resonance system may be hard at first seen. Therefore, we are led to illustrating the system with two analogies to get a simple picture of the whole system organization in mind. The first one, more general, compares the system to an orchestra; the second, to a desktop computer.

The orchestra

In our first analogy, a magnetic resonance system and the experiment can be seen as an orchestra playing a musical piece, as shown in Figure A.1.

Figure A.1 – Vancouver Symphony Orchestra with Bramwell Tovey. A magnetic resonance system can be seen as an orchestra: pulses are the notes; sequence is the music sheet; devices meant to be the musicians; peripherals are the instruments; the spectrometer itself is the conductor; the subject is the audience; the experiment is the musical piece, and; the data is the audience reaction.

Source: Wikipedia.

In this analogy, the sheet music with notes will be the pulse sequence, in which each note is positioned and synchronized among different layers to create the melody. Thus, a single timed note is a pulse in the sequence and each sheet music, for different groups of instruments, is a subsystem that will be used. Now, each musician (or groups of them) can read a part of the sheet music, interpret and then execute, transforming the notes into movements. This step is related to how devices deal with the sequence. As the conductor coordinates the musicians, the spectrometer processors will synchronize the subsystems, in parallel. As it should be guessed, the instruments, intimately connected to the musician, will be the peripherals. Each instrument

emits a different sound that compounds the melody. The movements generated by the musicians promote the sound. In the equipment, the peripherals transform the electrical signals into another form of energy, being anything such as electrical, movement, heat and so on. Playing the music piece means start from reading the sheet music until the end, producing sounds in the instruments. This process represents our experiment. The melody will produce a stimulus on the audience that is listening, which we figure out that will be the subject in the experiment. Finally, the listener's reaction to this stimulus symbolizes the signal acquired from the subject. The reaction, i.g., applauses or boos, provide feedback or the result to the orchestra's master, meaning the data acquired.

To depict the abstraction layers we mentioned in the *Concepts of magnetic resonance* note that in this analogy, the sheet music does not mention who (e.g., Ritchie Blackmore), or how (e.g., using his hands), or which instrument model (e.g., Gibson ES-335) the musician should play. It only contains the notes for each group in the orchestra; therefore, it represents the logical layer. The conversion layer is represented by the musicians themselves. The conductor (e.g., Malcolm Arnold), also in the conversion layer, will sync and manage all of them. Finally, the instruments (e.g., Gibson ES-335) are in the physical layer. Those who have already experienced an MRI exam should have noticed the sweet experiment melody resonating in the ears by almost a half-duration of Concerto For Group And Orchestra[†], however, less pleasant.

The computer

A second analogy we can make is with a desktop computer. In this case, examples of each part should promote further understanding of its relationships. The programs running on the computer will represent the pulse sequence. A program is an ordered set of commands that is executed to generate the desired output, fulfilling a purpose. Each command inside the program represents a pulse in the sequence. We can compare the antivirus, browsers, word processors and so on, to different pulse sequences, which have defined purposes. Next, the internal PC devices, such as the processor, motherboard, and memory are linked with the devices in the magnetic resonance system. Those devices convert commands to a physical response, i.e., electrical signal. When playing a video, processors should execute commands then send electrical signals to the video output. Then the video output is connected to external objects, named here as peripherals. Each system external connected to the PC cabinet is a peripheral.

[†] Deep Purple and The Royal Philharmonic Orchestra, 1969, it is about 60-minutes long.

Video monitors, speakers, keyboard, printers, all of them transform electrical signals to another physical quantity, i.e., sound, light, motion. Executing a program, which ordinarily provides a form of interaction between the computer and the user, should be interpreted as an experiment. Consequently, the users will be the subject, their reaction to the output is the result of that experiment. This result can be seen as feedback if it produces another input into the system.

Similarly to the previous analogy, imagine that the user wants to print a PDF file, coincidentally, a sheet music (e.g., Deep Purple - Child in time.pdf); it does not mind for the user in which port the printer is connected (e.g, COM3), or what printer is (e.g, HP LaserJet). In this example, the action of printing a file is in the logical layer. Then, the conversion of that PDF in bytes into the port is in the conversion layer, along with the processor, which will sync and manage the action among all of the devices. Finally, the printer is in the physical layer. Thus, this action can be executed in any computer with any printer, which generalizes the action of printing something.

APPENDIX B - ACCESS CONTROL, INTEGRITY

Access control

In the *Design and methods* it was suggested that there should be a system in place to assign different user permissions in order to protect the system from attacks. Malicious sources, which includes users with a low degree of technical knowledge with access to critical functions, can damage the system. Two approaches can be used to control access to classes, methods, and operations inside a real system setup.

The first one uses decorators on methods and functions. They can call the *permission manager* before the execution of the method or function itself. The main limitation of this method is that it requires inserting hard-coded decorators. Depending on the system implementation, when changes are made to user permissions, changes will need to be made to the system code too.

In order to avoid implementing control access in PyMR itself, the proxy approach was selected. This component is completely isolated from the PyMR and works by defining structures that contribute to proxy implementation. The proxy is the one that controls access to instances of the class. As such, it is a stand-alone module that is only required in the final product.

Code integrity

Complementing, a way to guarantee the file integrity (MR sequence, for example) must be defined. A common approach is known as the checksum, in which an algorithm can make operations over the file data to generate a unique hash, identifying it. Any modifications in the file reflect on checksum changes, causing the system's alert.

APPENDIX C - EXAMPLES OF DOCUMENTATION

PyMR Documentation, Release 0.12

since 2013/07/15

author Daniel Cosmo Pizetta

contact daniel.pizetta@usp.br

class **Amplifier** (parent=None, children=None)

Bases: `pymr.core.peripheral.peripheral.Peripheral`

Class that implements a basic amplifier.

Reference: [2] [7] [4]

addChild (child)

Add a new child in the children's list.

Parameters **child** (object) – object to be add.

Raises `ValueError` – if the given child could not be inserted as a child.

Returns children objects after child insertion.

Return type list

chainGain ()

Gain in whole chain of the system for each channel.

chainLoss ()

Loss in whole chain of the system for each channel.

isPossibleChild (obj)

Return whether or not is it possible to add the object class has a child.

Parameters **obj** (object) – object to be tested.

Returns false if not possible, true if possible.

Return type bool

maximumInputCurrent ()

The maximum input current is the minimum maximum value for each channel.

maximumInputPower ()

float [W]: Maximum input power [Power].

maximumInputVoltage ()

The maximum input voltage is the minimum maximum value for each channel.

registerObserver (*observer*)

Register observers to be updated.

removeChild (*child*)

Removes a child in the children's list.

Parameters **child** (*object*) – object to be remove.**Raises** `KeyError` – if the given child is not found in children's list.**Returns** children objects after child removal.**Return type** list**setUp** ()

Sets up the class, with loaded fields.

Should be re implemented in the derived class.

updateObservers ()

Update the observers.

validate ()

Validates class fields.

Should be re implemented in the derived class.

amplificationBandwidth = `None`*AmplificationBandwidth* – Amplification bandwidth.**amplificationClass** = `None`*AmplificationClass* – Amplification class type.**amplificationDevice** = `None`*AmplificationDevice* – Amplification device type.**amplificationOperation** = `None`*AmplificationOperation* – Amplification operation type.**description***str* – object description.**drop** = `None`*float [%]* – Drop [Percentage].

For long pulses or pulse trains, the drop becomes the most self.important parameter.

fallTime = `None`*float [us]* – Fall time [Time].

Rise and fall times are not much of a problem since the component with the smallest bandwidth (which determines the relevant rise and fall time) is usually the probe.

gain = None
float [dB] – Current gain value [Level of electric power].

gainRange = None
float [dB] – Minimum and maximum gain [Level of electric power].

inputImpedance = None
complex [ohm] – Input impedance [Electrical impedance].

label
str – Human readable label.

maximumDCPower = None
float [kW] – Maximum DC power [Power].

maximumPeakPower = None
float [kW] – Maximum peak power [Power].

model
str – human readable model.

name
str – Object name.

onOffRatio = None
float [dB] – On/off ratio [Level of electric power].
 The on/off ratio is always important since residual noise self.or even RF output will lead to increased noise in the spectrum or artifacts (spikes).

outputImpedance = None
complex [ohm] – Output impedance [Electrical impedance].

overshoot = None
float [%] – Overshoot [Percentage].
 Most important for good performance of short excitation pulses self.is a symmetric pulse shape with little or no visible overshoot and ripple.

possibleClassParents
 Returns a list of possible class parents.
 Should be re implemented in the derived class.

recoveryTime = None
float [us] – Time to recovery after a pulse [Time]

ripple = None
float [%] – Ripple [Percentage].
 Most important for good performance of short excitation pulses is a symmetric pulse shape with little or no visible overshoot self.and ripple. Low frequency ripple usually comes from insufficient power supplies, high frequency ripple from RF impurities.

riseTime = None
float [us] – Rise time [Time].
 Rise and fall times are not much of a problem since the component with the smallest bandwidth (which determines the relevant rise and fall time) is usually the probe.

APPENDIX D - EXAMPLE OF TUTORIAL

CHAPTER FIVE

TUTORIAL

Creating a Sinc Pulse

First, you need to create a Sinc Shape Set, i.e. a set with AM, FM and PM:

```
import pymr.core.methodology.shape as shape
sincSet = shape.SincRFShapeSet(number_of_points=512,
                                zeros_to_left=4,
                                zeros_to_right=4,
                                labelSet="CuteSinc")
```

The parameters `zeros_to_left` and `zeros_to_right` determines the number of zero-crossings to the left/right of the central lobe.

The AM table has values from -1 to 1, and FM/PM table are filled with 1.

You can use any label to this set.

We can now create a RF Sinc Pulse:

```
import pymr.core.methodology.pulse as pulse
rf = pulse.RFPulse(time_duration = 0.005,
                   rf_shape_set = sincSet,
                   name="HappyPulse")
```

Duration of the pulse is set in seconds, here it was 0.005 s (or 5 ms).

And set a multiplicative factor for AM:

```
rf.amMultiplyFactor = 42.
```

it is analogous for FM and PM.

To access AM (or FM/PM) table:

```
rf.amTable()
```

It will return a numpy array with values ranging -amMultiplyFactor to amMultiplyFactor.

To save this RF Pulse as a JSON file:

```
rf.write(file_path="WhereToSaveTheJsonFile")
```

To read a saved RFPulse:

```
rf_new = pulse.RFPulse()  
rf_new.read(file_path="WhereTheSavedFileIs")
```

Whenever a Shape/Pulse file is read, all information about the parameters used are discarded, in this example, we lost the information about the number of zero-crossings of the AM Shape. It is a clever idea to store this stuff in the pulse/shape label, as the label is read from the file.

Please, refer to `SincRFShapeSet()` and to `RFPulse()` for more details.

APPENDIX E - ICONS FOR MAGNETIC RESONANCE CONTEXT

Icons representing magnetic resonance elements are rare or poor designed on available software. Consequently, knowing the whole system and its characteristics, it was possible to create an appropriate icon representation package for use in magnetic resonance software. In the next pages, tables present the icon sub-packages we have created using Inkscape software. The key aspects are depicted in the Figure E.1. They include the indication of sets, subsystems, derived classes, and standard actions.

Figure E.1 – Magnetic resonance icon package fundamental aspects. a) Characteristic signal waveforms indicate the subsystems. b) Shadows represent sets. c) Derived classes keep their base class main image. d) Main actions use popular designs.

Source: By the author.

Project, file, and edit

Table E.1 – Project icons.

Name	Icon
Print	
Import	
Open	
Save As	
New	
Save	
Export	

Source: By the author.

Table E.2 – File icons.

Name	Icon
Print	
Import	
Open	
Save As	
New	
Save	
Export	

Source: By the author.

Table E.3 – Edit icons.

Name	Icon
Undo	
Cut	
Copy	
Redo	
Paste	

Source: By the author.

Peripherals

Table E.4 – *Peripheral* icons.

Name	Icon
Peripheral Set
Peripheral
RF Attenuator
RF Switch

Source: By the author.

Table E.5 – *Magnet* icons.

Name	Icon
Magnet
Magnet Set
Resistive Magnet
Permanent Magnet
Superconducting Magnet

Source: By the author.

Table E.6 – *Cable* icons.

Name	Icon
Cable
Wire
Connector

Source: By the author.

Table E.7 – *Chiller* icons.

Name	Icon
Chiller
Chiller Set

Source: By the author.

Table E.8 – *Amplifier* icons.

Name	Icon
Shim Amplifier
Gradient Amplifier Set
Shim Amplifier Set
Amplifier
RF Amplifier Set
RF Preamplifier
RF Preamplifier Set
Gradient Amplifier
Amplifier Set
RF Amplifier

Source: By the author.

Table E.9 – *Coil* icons.

Name	Icon
Shim Coil
RF Coil
Coil
Gradient Coil
RF Coil Set
Gradient Coil Set
Shim Coil Set
Coil Set

Source: By the author.

Devices

Table E.10 – Device icons.

Name	Icon
Filter
ActuatorDAC
Finite Impulse Response Filter
Numerically Controlled Oscillator
Analog Digital Converter
Device Set
Actuator
Cascade Integrator Comb Filter
SensorDAC
ActuatorADC
Sensor
Digital Analog Converter
Infinite Impulse Response Filter
SensorADC
Device
Emphasis Filter
Transducer

Source: By the author.

Table E.11 – Gradient icons.

Name	Icon
Gradient Set
Gradient

Source: By the author.

Table E.12 – Shim icons.

Name	Icon
Shim
Shim Set

Source: By the author.

Table E.13 – Receiver icons.

Name	Icon
Receiver
Receiver Set

Source: By the author.

Table E.14 – Transmitter icons.

Name	Icon
Transmitter Set
Transmitter

Source: By the author.

Methods

Table E.15 – *Shape* icons.

Name	Icon
Quarter Sin Shape
Trapezoid Shape
Gradient Shape Set
Sech Shape
Spiral Shape
RF Shape Set
Sinc Shape
Triangle Shape
Lorentz Shape
Ramp Shape
Gaussian Shape
Half Sin Shape
Shape Set
Shape
Constant Shape
Rectangle Shape
Square Shape

Source: By the author.

Table E.16 – *Event* icons.

Name	Icon
Delay Event

Source: By the author.

Table E.17 – *Pulse* icons.

Name	Icon
Gradient Pulse
RF Pulse
Pulse

Source: By the author.

Table E.18 – *Method* icons.

Name	Icon
Method
Plane

Source: By the author.

Table E.19 – *Loop* icons.

Name	Icon
Loop

Source: By the author.

Table E.20 – *Protocol* icons.

Name	Icon
Protocol
Protocol Set

Source: By the author.

Table E.21 – *Safety* icons.

Name	Icon
Safety

Source: By the author.

*Subject*Table E.22 – *Nucleus* icons.

Name	Icon
Li7 Nucleus	
F19 Nucleus	
N15 Nucleus	
N14 Nucleus	
H2 Nucleus	
Nucleus	
P31 Nucleus	
He3 Nucleus	
H1 Nucleus	
Na23 Nucleus	
Nucleus Set	
Fe57 Nucleus	
C13 Nucleus	
O17 Nucleus	
Xe129 Nucleus	

Source: By the author.

Table E.23 – *Phantom* icons.

Name	Icon
T1T2 Phantom	
Signal Phantom	
Aqueous Copper Phantom	
Aqueous Manganese Phantom	
Linearity Phantom	
Fat Saturation Phantom	
Water Saturation Phantom	
Resolution Phantom	
Gelatin Niquel Phantom	
Diffusion Phantom	
Aqueous Niquel Phantom	
Aqueous Oxygen Phantom	
Phantom	
Homogeneity Phantom	
Phantom Set	

Source: By the author.

Miscellaneous

Table E.24 – *Parameter* icons.

Name	Icon
Parameter Number	
Parameter	
Parameter Set	
Parameter Map	
Parameter Bool	
Parameter Array	

Source: By the author.

Table E.25 – *Action* icons.

Name	Icon
Activate	
Edit	
Add	
Remove	
Enable	
Duplicate	
Update	
Validate	

Source: By the author.

Table E.26 – *View* icons.

Name	Icon
Parent	
Package Hierarchy	
All	
Class Hierarchy	
Object Hierarchy	
Children	

Source: By the author.

APPENDIX F - DOCUMENTING PHYSICAL QUANTITIES

Magnetic resonance systems, among many other physics-related systems, deal with many values that are physical quantities such as the distance in meters, the power in watts, or the electrical current in amperes. If they are not well documented or defined in the code, they could cause serious problems[†].

In the Python language, neither Python itself nor Sphinx⁹¹ library has a proper way to inform the reader (by annotation) that a specific variable is related to a physical quantity[‡]. We propose a clear pattern that could also be included into the official Sphinx repository or in a Sphinx extension. Other programming languages or documenting tools could adopt those patterns as a reference.

The International System of Units (SI[§]) is the primary worldwide-used system of units.¹²⁰ The main reference to units names, quantities, symbols, and composition is the *Guide for the Use of the International System of Units (SI)*.^{120,121} A physical quantity has some essential parts that will be explained as follows. The Table F.1 has some examples of physical quantities and its characteristics. We also indicate the unit symbol and name, prefixes, quantity name, and if the quantity name derives from a personal name.

- **Quantity name:** Is what the quantity measures. They must not contain prefixes, i.e., nano, kilo in quantity field.¹²⁰ The use of quantity field is import to create a system-unit independent annotation. Ex.: Electric current, time, amount-of-substance, energy, length, mass.
- **Dimension (quantity symbol):** Represents the physical quantity dimension, a symbol for quantity name. As the quantity name, the dimension is independent of the system of units adopted. Ex. L (length), M (mass), T (time).
- **Unity name:** Names of units are always in lowercase, even if they are derived from personal names. They could contain prefixes if necessary. As it will be noticed, we will not use unit names since they are already cited by its symbol (and the quantity name which

[†] In 1999, the \$125-million NASA Mars probe was lost because of the English to the metric system of units. Source: L.A.Times.

[‡] The information refers to the annotation about the physical quantity, e.g., 1m;

[§] The International System of Units is abbreviated to “SI” from the French *Système international (d’unités)*.

they represent). Ex.: ampere, second, millisecond, kilogram, nanovolt per millimeter.

- **Unity symbol:** A symbol is used to identify the unity name. Note that if it is derived from a personal name, the unit symbol has capital letters, otherwise, it is all lowercase.¹²⁰ Ex.: A (ampere, related to the name Ampère), s (second), V (volt, related to the name Volta), mV (millivolt, as the last one).
- **Prefixes:** Prefixes are used to compose unit names and symbols. They do not appear in quantity name or dimension information. They are factors to multiply the elementary quantity value. Ex.: mT (millitesla = (1/1000) T), cm (centimeter = (1/100) m).
- **Operators:** Operators appears with symbols and prefixes in composed units. If just “*” and “/” are needed, precedence order wins. The use of parenthesis is permitted (and necessary) when more than one unit make part of the denominator. It is permitted just one operator of division per composed unit. Ex.: mT/cm (millitesla per centimeter), W/(m*K) (watt per meter kelvin), J/(kg*K) (joule per kilogram kelvin), V/m/T (volt ver meter per tesla, it is wrong defined).

Table F.1 – Examples of physical quantities and their composition.

Physical Quantity	Unit Symbol	Unit Name	Prefixes	Unity Quantity	Personal Name
5.3 mA	mA	miliampere	milli	Electrical current	Ampère
3 nV	nV	nanovolt	nano	Voltage	Volta
2.9 V/m	V/m	volt per meter	—	Voltage / Length	Volta
258 MJ	MJ	joule	mega	Energy	Joule
680 J	J	joule	—	Heat	Joule
2 T*m/s	T*m/s	tesla meter per	—	Magnetic field flux * Length /	Tesla
		second		Time	
2.9 m	m	meter	—	Length	—
2.1 kg	kg	kilogram	kilo	Mass	—

Source: By the author.

Format definition and examples

The characters used to indicate the unit symbol and quantity name in the docstrings are brackets (“[“, “]”), a universal symbol on physical books to make the dimensional analysis. Besides that, brackets are not a marker inside the Sphinx library⁹¹ nor reStructuredText markup language. The *symbol* contains the unit symbol of that physical quantity, i.e. a short format. The *quantity* itself contains what that unit describes, avoiding misinterpretation and not depending on the system of units selected. We do not use the unit name because is too long compared to the symbol neither the dimension because they only exist for base units. The two main possible usage formats are defined as follows.

```
# attributes and parameters definition
variable_name (type [Symbol]): Variable description [Quantity].

# property and return definition
type [Symbol]: Return description [Quantity].
```

The next examples are based on Google Style documentation, used in the PyMR, but the same definitions can be used for the Sphinx and NumPy styles. Also, some wrong definitions (double quoted) are shown.

```
# attributes and parameters definition
max_rate (float [mT/s]): Maximum field rate [Magnetic field flux / Time].
max_power (float [W]): Coil maximum continuous power [Power].

# property and return definition
float [W]: Coil maximum continuous power [Power].
float [mT*m/s]: Slew rate for the magnetic gradient [Magnetic field flux * Length / Time].

# wrong definitions, marked with double quotes (")
current [A]: Current in the coil ["Ampere"].
current ["i"]: Current in the coil [Electric current].
float [mT*(m/s)]: Slew rate for the magnetic gradient [Magnetic field flux * "(Length / Time)".
```


APPENDIX G - PROPOSAL FOR A NEW VARIABLE TYPE

Python, like other programming languages, not deal with physical quantities directly. In Python, many libraries propose to solve this problem, but none of them could do it overall. As mentioned at the conference of SciPy^{90,122}, is still a better way and it is needed an enormous effort to solve this problem. Up to now, the best option is to use the Pint library to declare a new class. That is what we are using at this moment. Another option, which could contribute to the progress of the Python language, would be to declare a new basic type of variable derived from the float, making it different from other ones by the unit. In Python, it is prohibited to declare a variable name using a sequence starting with numbers, or dot, then letters (this returns a syntax error message). So we can use this to our favor. We will post this proposal in the PEP8 forum. Thus, instead of declaring (using Pint), lines 2-3, if we change the Python syntax, it will be possible to declare (using a currently prohibited syntax⁸³), lines 5, 6, 8.

```

1 # declaring with pint library
2 distance = PhysicalQuantity(1.2, 'm') # meters or
3 distance = PhysicalQuantity('1.2m') # meters
4 # new syntax
5 distance = 1.2m # meters
6 distance = 5.3mm # millimeters, using prefix
7 # also possible using composed units
8 velocity = 5.3m / 2.8s # meters per second

```

In the next example, note that in line 2, the division operator continues to be used inside the unit and it can be split as in the line 4. All non-float numbers can be converted to float then operating division, line 6. The precedence for the operators is also valid, and they will generate the line 8. The printed value of this operation can be seen in line 10.

```

1 # operating sum and subtraction, using composed units
2 total_velocity = 2.5m/.s + 3.m/.s - 5m/.s
3 # which is equal to
4 total_velocity = 2.5m / .s + 3.m / .s - 5m / .s
5 # which is equal to
6 total_velocity = 2.5m / 1.0s + 3.0m / 1.0s - 5.0m / 1.0s
7 # which will be processed like this
8 total_velocity = (2.5m / 1.0s) + (3.m / 1.0s) - (5.0m / 1.0s)
9 # returning value has the unit `m/s`, which includes an operator
10 >>> total_velocity = 0.5m/.s

```

Now, two examples of division and multiplication are shown. In all those examples,

the rules of precedence in the operations could be used to determine the correct calculus. The validity of the operations and syntax will not be discussed here, as they have been already implemented in all those unit libraries.

```

1 mass = 10kg
2 light_velocity = 300000.m/.s
3 # formula
4 energy = mass * light_velocity ** 2
5 # this formula should be processed like this
6 energy = mass * (light_velocity ** 2)
7 # using the values
8 energy = 10.0kg * (( 300000.0m / 1.0s) ** 2 )
9
10 # another example
11
12 acceleration = 10.m/.s**2
13 time = 3.5s
14 # formula
15 position = acceleration * time ** 2 / 2
16 # this formula should be processed like this
17 position = (acceleration * (time ** 2)) / 2.0
18 # using the values
19 energy = (10.0m / 1.0s ** 2 * (( 3.5s ) ** 2)) / 2.0

```

Another example, but now using the wrong definitions is given.

```

1 # using composed units with wrong definitions
2 velocity = 10m/s
3 # which will be processed like this
4 velocity = 10.0m / s      # "s" is read as a variable here
5 # correct quantity/unit for distance
6 distance_a = 5.3m
7 # wrong quantity/unit for distance
8 distance_b = 5s
9 # wrong definition of distance when mixing quantities
10 distance = distance_b - distance_a

```

APPENDIX H - PRODUCTION AND PROJECTS

There are listed below, intellectual properties, publications, and participation in projects during the Ph.D. progress. They are only main-theme related. Others contributions can be seen in the next appendix.

Intellectual properties

- TANNÚS, A.; PIZETTA, D.C.; SILVA, D.M.D.D; VIDOTO, E.L.G; LOURENÇO, G.V.; MARTINS, M.J. Revista da Propriedade Industrial - RPI n.2365, p.351, item 120. **Software registry.** ToRM-IDE - Integrated development environment for magnetic resonance applications. Process number: BR512015001484-6. Registry date: 07 dec. 2015.⁵¹
- TANNÚS, A.; PIZETTA, D.C.; SILVA, D.M.D.D; VIDOTO, E.L.G; MARTINS, M.J. Revista da Propriedade Industrial - RPI n.2365, p.350, item 120. **Software registry.** Subsistema multiplataforma para controle de aquisição, visualização e organização de dados de espectrômetro digital de RM: ToRM Console. Process number: BR512015001485-4. Registry date: 07 dec. 2015.⁵²
- TANNÚS, A.; PIZETTA, D.C.; VIDOTO, E.L.G.; MARTINS, M.J.; SILVA, L.C.R.; SHIMADA, D.H.T.; FALVO, M.; SOUZA, P.V.B.D.; BITTENCOURT, H.P. **Software registry.** PyMR - A framework for programming magnetic resonance systems. Process number: CC-PC-2017-0035. Registry request: 31 aug. 2017. Process in progress.¹²³

Publications

- PIZETTA, D.C.; LOURENÇO, G.V.; SILVA, D.M.D.D; L.G. VIDOTO, E.L.G; MARTINS, M.J.; TANNÚS, A. Magnetic resonance system configuration and editing tools. In: IUPESM World Congress On Medical Physics & Biomedical Engineering, Toronto, Canada, 2015. **Abstract Book.** 667–8 p.¹²⁴
- PIZETTA, D.C.; PESSOA, V.H.M.; PAIVA, F.F. Simulating magnetic resonance experiments with Precession. In: Simpósio de Instrumentação e Imagens Médicas (SIIM), 8th and Simpósio de Processamento de Sinais (SPS), 7th, Santo André, 2017. **Anais.** [electronic] ID 169.¹²⁵
- PIZETTA, D.C.; SILVA, L.C.R.; VIDOTO, E.L.G.; MARTINS, M.J.; TANNÚS, A.; A framework for programming a modern magnetic resonance system. In: Simpósio de Instrumentação e

Imagens Médicas (SIIM), 8th and Simpósio de Processamento de Sinais (SPS), 7th, Santo André, Brazil, 2017. **Anais.** [electronic] ID 178.¹²⁶

Projects

- TANNÚS, A. (HEAD); MARTINS, M.J.; PAIVA, F.F.; PAPOTI, D.; PIZETTA, D.C.; COELHO, F.B.; MALHEIROS, J.M.; VIDOTO, E.L.G; CONSALTER, D.M.; FREIRE, R.T.; SILVA, L.C.R.; VAZ, F.R.; SHIMADA, D.H.T. (PARTICIPANTS). **Project.** Desenvolvimento de subsistemas de equipamento de imagens por ressonância magnética. **Funding.** Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq. 2014-current.
- UNIVERSITY OF MINNESOTA; UNIVERSITY OF SÃO PAULO; YALE UNIVERSITY; COLUMBIA UNIVERSITY; VICTORIA UNIVERSITY OF WELLINGTON; MGH/HARVARD. (CONSORTIUM). **Project.** Imaging human brain function with minimal mobility restrictions. **Funding.** NIH R24 MH105998 (2014-2017), NIH U01 EB025153 (2017-2022).
- UNIVERSITY OF MINNESOTA; UNIVERSITY OF SÃO PAULO; KATHOLIEKE UNIVERSITEIT LEUVEN; UNIVERSITY OF ARIZONA. (CONSORTIUM). **Project.** 0.5 T desktop pO₂ scanner. **Funding.** Schott Foundation, Olson Foundation, MLDF, JDRF.

APPENDIX I - OTHER CONTRIBUTIONS BY THE AUTHOR

Several contributions to open software projects have been released in the course of the author's Ph.D. program. The following items listed have been affected by the PyMR outcome. Those contributions are pull-requests submitted to the original repositories or new ones, extensions, and plugins. The list does not include simple contributions as reported issues.

- **Sphinx** - Python Documentation Generator.⁹¹ A bug fix in the inheritance diagram function. Bug found when generating PyMR documentation.
- **QtPy** - Abstract layer for PyQt5/PyQt4/PySide2/PySide.⁹⁵ New features when QtPy attempts to import non-installed Qt bindings and improvements in documentation, under pull request #156. A feature that simplifies binding setup in PyMR.
- **PyQtGraph** - Data visualization library.⁹⁶ Implementation of array-like parameter type for visualization - not published yet. It is needed for the edition of array-like data that is not implemented by the original library, used in PyMR.
- **InspectExtender** - Extension for the native Python library "inspect". Implementation of functions to classify and get complex information about Python members (classes, methods, attributes, properties, etc) using "inspect" original module as back-end. This code populates object's creator, inspector, and editor in PyMR.
- **Handler for physical units in Sphinx** - Including new definitions for units in the classical documentation, this should avoid doubts or induced errors. Not public yet.

Besides, other projects, unrelated to PyMR or its necessities, were created to cover problems in the PAE program, which is part of the Ph.D. program requirements. Those tools are in use in the physics laboratory, a subject for undergraduate students, at the University of São Paulo, São Carlos. The last one was developed meanwhile coursing Scientific Writing subject at the beginning of the Ph.D. program. Those projects are available as open source under MIT license.

- **Standing Sound Waves** - A Python program used to control data acquisition via microphone and a stepper motor driven by Arduino.¹¹⁶ It was adopted to automate procedures when acquiring data in the Kundt's Tube experiment. It enabled new demonstrations and analysis of phenomena not seen in the conventional data acquisition method (manual and inaccurate). This project also generates the publication entitled "An

experimental valuation of standing sound waves in pipes. *Rev. Bras. Ensino Fís.*, São Paulo, v.39, n.3, e3301, 2017”¹¹⁷ in collaboration with Adilson Barros Wanderley, and professors Valmor Roberto Mastelaro and Fernando Fernandes Paiva.

- **Wavy Tool** - A simple graphical Python tool that acquires data from input devices, plot and export data files.¹¹⁸ This project aims to provide a system to acquire data of the experiment Ruchardt’s method for C_p/C_v determination in gases. The data acquired by this method and its results still under study and it will be published soon.
- **Sci Corpus** - A scientific corpus data manager.¹¹⁹ It was developed to help the compilation of texts and notes extracted from papers using the technique “learning by doing”. José Ricardo Furlan Ronqui, Thiago Campos, and Thiago Mosqueiro collaborate with this project. A paper about this tool is under construction.