SIRF: Synergistic Image Reconstruction Framework

Evgueni Ovtchinnikov[a], Richard Brown[b,*], Christoph Kolbitsch[f], Edoardo Pasca[a], Casper da Costa-Luis[d], Ashley G. Gillman[k], Benjamin A. Thomas[b], Nikos Efthimiou[e], Johannes Mayer[f], Palak Wadhwa[h,j], Matthias J. Ehrhardt[l], Sam Ellis[d], Jakob S. Jørgensen[i], Julian Matthews[g], Claudia Prieto[d], Andrew J. Reader[d], Charalampos Tsoumpas[h,j], Martin Turner[i], David Atkinson[c], Kris Thielemans[b,**]

[a] *Scientific Computing Department, Rutherford-Appleton Laboratory, UK Research and Innovation, Harwell Campus, Didcot OX11 0QX, UK*

[b] *Institute of Nuclear Medicine, University College London, 235 Euston Rd, London NW1 2BU, UK*

[c] *Centre for Medical Imaging, University College London, 43-45 Foley Street, London W1W 7TS, UK*

[d] *School of Biomedical Engineering & Imaging Sciences, King's College London, Lambeth Wing, St. Thomas' Hospital, Westminster Bridge Road, London SE1 7EH, UK*

[e] *PET Research Centre, School of Health Sciences, University of Hull, Hull HU6 7RX, UK*

[f] *Physikalisch-Technische Bundesanstalt (PTB), Bundesallee 100, Braunschweig D-38116 and Abbestrasse 2-12, Berlin D-10587, Germany*

[g] *Division of Informatics, Imaging and Data Sciences, University of Manchester, 46 Grafton Street, Manchester M13 9NT, UK*

[h] *Biomedical Imaging Science Department, University of Leeds, Leeds LS2 9JT, UK*

[i] *School of Mathematics, University of Manchester, Oxford Road, Manchester M13 9PL, UK*

[j] *Invicro London, Du Cane Rd, London W12 0NN, UK*

[k] *Australian e-Health Research Centre, CSIRO, Brisbane, Queensland 4029, Australia*

[l] *Institute for Mathematical Innovation, University of Bath, Convocation Ave, Bath BA2 7JU, UK*

Abstract

The combination of positron emission tomography (PET) with magnetic resonance (MR) imaging opens the way to more accurate diagnosis and improved patient management. At present, the data acquired by PET-MR scanners

*Joint first author

**Corresponding author. *E-mail address:* k.thielemans@ucl.ac.uk

are essentially processed separately, but the opportunity to improve accuracy of the tomographic reconstruction via synergy of the two imaging techniques is an active area of research.

In this paper, we present Release 2.1.0 of the CCP-PETMR Synergistic Image Reconstruction Framework (SIRF) software suite, providing an open-source software platform for efficient implementation and validation of novel reconstruction algorithms. SIRF provides user-friendly Python and MATLAB interfaces built on top of `C++` libraries. SIRF uses advanced PET and MR reconstruction software packages and tools. Currently, for PET this is Software for Tomographic Image Reconstruction (STIR); for MR, Gadgetron and ISMRMRD; and for image registration tools, NiftyReg. The software aims to be capable of reconstructing images from acquired scanner data, whilst being simple enough to be used for educational purposes.

The most recent version of the software can be downloaded from `http://www.ccppetmr.ac.uk/downloads` and `https://github.com/CCPPETMR/`.

Keywords: Image Reconstruction; PET-MR; Multi-modality; Medical Imaging; Open-Source Software

---

PROGRAM SUMMARY

Program Title: Synergistic Image Reconstruction Framework (SIRF)
Licensing provisions (please choose one): GPL 3 and Apache-2.0
Programming languages: C++, C, Python, MATLAB
Nature of problem:
In current practice, data acquired by PET-MR scanners are processed separately. Methods for improving the accuracy of the tomographic reconstruction using the synergy of the two imaging techniques are actively being investigated by the PET-MR research and development community, however, practical application is heavily reliant on software. Open-source software available to the PET-MR community – such as the PET package Software for Tomographic Image Reconstruction (STIR) [1] and the MR package Gadgetron [2] – provide a basis for new synergistic PET-MR software. However, these two software packages are independent and have very different software architectures. They are mostly written in `C++` but many researchers in the PET-MR community are more familiar with script-style languages, such as Python and MATLAB, which enable rapid prototyping of novel reconstruction algorithms. In the current situation it is difficult for researchers to exploit any synergy between PET and MR data. Furthermore, techniques from one field cannot easily be applied in the other.
Solution method:

In SIRF, the bulk of computation is performed by available advanced open-source reconstruction and registration software (currently STIR, Gadgetron and NiftyReg) that can use multithreading and GPUs. The SIRF `C++` code provides a thin layer on top of these existing libraries. The SIRF layer has unified data-containers and access mechanisms. This `C++` layer provides the basis for a simple and intuitive Python and MATLAB interface, enabling users to quickly develop and test their reconstruction algorithms using these scripting languages only. At the same time, advanced users proficient in `C++` can directly utilise wider SIRF functionality via the SIRF `C++` libraries that we provide.

1. Introduction

Magnetic resonance (MR) images can provide high resolution anatomical images with excellent soft tissue contrast, but provide only limited functional and metabolic information. In contrast, positron emission tomography (PET) is able to provide highly sensitive and specific functional imaging, but at low spatial resolution (4-7 mm). The many PET tracers available can provide information on a wide range of processes. Recognition of the complementary aspects of the two imaging modalities prompted the development of integrated clinical PET-MR scanners. Hybrid PET-MR imaging [3, 4] promises superior application in oncological, paediatric [5], cardiac [6] and neurological [7, 8] imaging, both in clinical and research settings. However, the technical hardware developments have not yet been paralleled by the development of an integrated software platform for processing data acquired by such scanners.

Image reconstruction is a challenging problem for each modality, albeit for somewhat different reasons. PET image quality is limited by noise and resolution thus it requires regularisation during image reconstruction [9]. On the other hand, MR data are often undersampled for acquisition speed [10], and model-based reconstruction is often needed [11, 12]. Both modalities could benefit from additional information in the reconstruction. For PET, this has most often been exploited by using a high quality MR image as "side-information" [13]. Similar strategies can be used for dynamic MR data [14]. Recently, researchers have started exploring methods for joint reconstruction of both data-sets, exploring the underlying structural similarity [15, 16, 17, 18, 19]. However, this research has been confined to single institutions using their own internal software packages.

Many open-source packages exist for single-modality image reconstruction, for instance for PET [1, 20, 21, 22, 23] or MRI [24, 2] (an extensive list of Open Source imaging software is provided at `https://www.opensourceimaging.org/category/software`, and `https://www.ismrm.org/mri_unbound/sequence.htm` and `http://stir.sourceforge.net/links` contain numerous links to MR and PET software). However, none of these packages allow the reconstruction of data from both modalities, and are therefore not suitable for joint reconstruction of PET and MR data. The Collaborative Computational Project for PET-MR (CCP-PETMR, `http://www.ccppetmr.ac.uk`) is a network established in April 2015 with the aim of facilitating the investigation of novel synergistic PET-MR image reconstruction methods by providing the PET-MR research community with a software-development platform. The platform should be simple enough to use for research and educational purposes and, at the same time, powerful enough to be able to handle raw data acquired by PET, MR and PET-MR scanners within a reasonable processing time. Both of the available integrated clinical PET-MR systems, the Siemens 3T Biograph mMR (Siemens Healthineers, Erlangen, Germany) [25] and the General Electric Signa PET/MR [26, 27], are being targeted by this software platform, and a suitable data format exchange is being negotiated with the manufacturers. This paper describes Release 2.1.0 of the open-source Synergistic Image Reconstruction Framework (SIRF).

2. SIRF structure

2.1. Rationale

Image reconstruction requires complex software and support for different types of scanners. Developing software that handles two substantially different imaging modalities is therefore a challenging task. While it would have been possible to extend one of the many open-source software projects that aim to support a single modality, this would inevitably have led to duplicating the effort. Instead, SIRF is designed to act as a "framework" providing a layer on top of existing packages for medical image reconstruction, referred to as *engines* in this document. This layer aims to present a consistent interface for data manipulation and image reconstruction, independent of the modality and the underlying engine. Such an architecture has the additional advantage that it allows SIRF to benefit from the active development in each of the underlying engines. Note that these design choices are similar to those

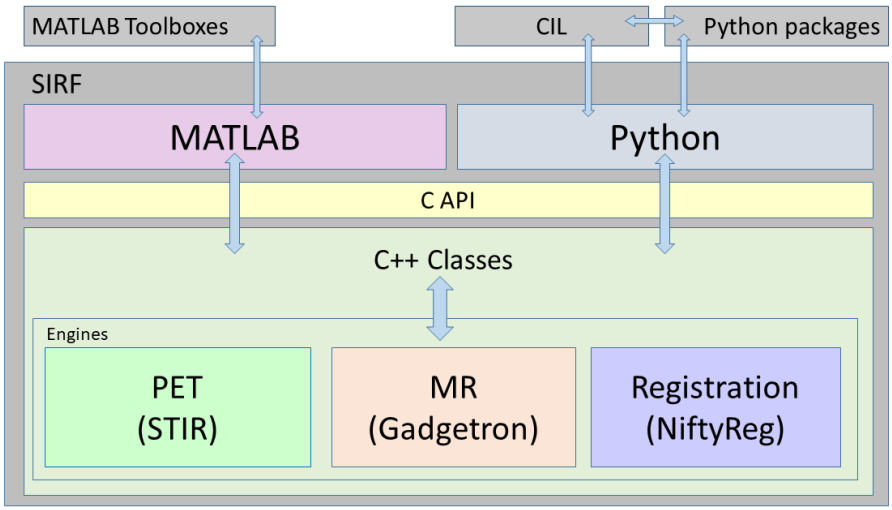

Figure 1: The architecture of SIRF: interfacing of underlying engines and most of the development is in `C++`. Interfacing MATLAB and Python with the underlying `C++` code is done via a thin C API. CIL is a Python library described in §5.1.1.

of Operator Discretisation Library (ODL) [28] and Core Imaging Library (CIL), see §5.1.1. However, both are Python-only and do not support MRI.

An important choice is the programming language that the users of our platform will need to adopt for their own software development. Two languages and environments that are widely used by researchers are MATLAB and Python, in part because of the excellent prototyping abilities of these languages, while `C++` is more commonly used for intensive computational applications, often benefiting from multi-threading via OpenMP. Most of the existing open-source software packages consist of a core `C++`-library, sometimes with interfaces to MATLAB and/or Python. In SIRF, we have chosen to do most of the development in `C++`, including the interfacing to the underlying engines, as described in Figure 1. The `C++` code is then interfaced to simple Python and MATLAB building blocks. We aim to have close similarity between the `C++`, Python and MATLAB interfaces to SIRF, but expect that most of the users of our platform will use Python or MATLAB, and these interfaces are therefore currently most complete and well-documented.

SIRF uses an object-oriented approach to programming. A short description of this approach is presented in Appendix A and is illustrated by SIRF

usage examples.

2.2. Underlying engines

At present, we use STIR (Software for Tomographic Image Reconstruction) [1] as a PET engine, Gadgetron [2] as an MR engine and NiftyReg as the engine for registration and resampling [29, 30]. We briefly describe these engines here, but refer to their respective publications and web-sites for more detail.

STIR [1] implements a library of `C++` classes for performing PET reconstruction and related tasks such as data input / output. Parameters of a STIR reconstructor object are normally defined by the user in a text-based parameter file, and a set of executables is provided that read the parameter file specified in the command line and perform the required tasks, so that the user does not need to be an expert in `C++` or any other programming language. For users familiar with Python or MATLAB, the current development version of STIR offers an alternative usage via Python / MATLAB scripts. This requires building Python and MATLAB interfaces to STIR by SWIG [31]. However, these interfaces currently do not cover all functionality of STIR. Moreover, they expose the complexity of underlying `C++` library.

With Gadgetron [2], MR image reconstruction is performed by specifying a chain of 'gadgets', i.e. pieces of code implementing specific tasks within a streaming architecture. The chain of gadgets is executed on a server, which can be launched in the background of the same computer, or it can be another computer, or a Virtual Machine, or a cloud-based computational infrastructure. A client can communicate with the server via TCP/IP. In order to set up the gadget chain, the server needs to receive a text file in the Extensible Markup Language (i.e. `xml` file format) describing the chain. Then, the first gadget in the chain is activated and waits for the acquired data to arrive from the client in chunks of a certain size. Having processed a chunk of data, the first gadget passes the result to the second, starts processing the next chunk and so on. The last gadget sends the reconstructed images back to the client. As with STIR, a Gadgetron user is not required to have any knowledge of `C++`. Instead, the reconstruction tasks can be configured via parameter files in xml format.

NiftyReg [29, 30] is a `C++` image registration library, restricted to images stored in the NIfTI image format [32]. Rigid, affine and non-rigid image registrations are possible, and these are available in both symmetric and non-symmetric versions. From the registrations, it is possible to extract

transformation matrices (for rigid and affine registrations), and deformation and displacement field images. With motion information extracted, it is then possible to perform resampling as required.

2.3. SIRF software layers

2.3.1. C++

The `C++` layer provides the basic class hierarchies, in particular data container classes, and specific implementations where we wrap the functionality provided by the engines as well as implementing any extra components needed for the general usage of SIRF.

2.3.2. Interfacing into Python and MATLAB

At present, the SIRF user interface consists of the modules `sirf.Gadgetron`, `sirf.STIR` and `sirf.Reg` wrapping the engines, together with some additional functionality in `sirf.Utilities`.

To avoid difficulties interfacing between the `C++` code and the targeted script languages, we include an additional layer by wrapping `C++` code into `C`. This extra layer simplifies interfacing and widens the scope of our targets to practically any programming language in use.

Interfacing of `C` into MATLAB is facilitated by the availability of MATLAB functions (`loadlibrary`, `callib` etc.) that allow direct calls of functions from `C` libraries (with a little 'syntactic sugar' needed under Windows). For Python we use SWIG [31], which can also be used for a number of other prospective languages.

The MATLAB and Python interfaces based on this `C` library are not user-friendly and not intended to be accessed directly. Instead in addition to these, we have Object-Oriented MATLAB and Python modules mirroring and extending the `C++` class structure.

3. SIRF objects

This section briefly describes the type of objects with which SIRF users need to operate in order to perform image reconstruction and related tasks. A more detailed description is provided in the SIRF User's Guide, and examples are given in §4.

3.1. Image and acquisition data objects

In SIRF, acquired data are handled by objects of class `AcquisitionData` and images by objects of class `ImageData`. Both are derived from an abstract data class `DataContainer` that implements data algebra: the user's code can compute norms, dot products and linear combinations of data objects of the same class viewed as vectors.

We created a hierarchy of image data objects that allows us to handle representations of images employed in different reconstruction engines in a unified way. On the top of this hierarchy, we have an abstract base class `ImageData`, from which the abstract classes `PETImageData` and `MRImageData` are derived. Further down this hierarchy, we have various classes encapsulating different representations (e.g file or array) of image data used in different reconstruction engines, such as `STIRImageData`. SIRF uses a patient-based coordinate system to encode the physical position of a given volume element. This allows spatial context to be integrated between images from different engines.

3.2. Acquisition models

An 'acquisition model' is the terminology for a mathematical model of the scanner acquisition process, which can be represented by

$$A(x) = b \tag{1}$$

where x represents the scanned object, typically a vector of values at points in three-dimensional space defining a discretised image, b approximates the actual data collected by the scanner and A is an operator (generally non-linear) that models the acquisition process.

The computation of $A(x)$ for a given image data representation x is referred to as the *forward projection*. The *backprojection* is the application of the adjoint of the Fréchet derivative of A to acquisition data. For linear models, the backprojection corresponds to the Hermitian conjugate operation.

In SIRF, PET and MR acquisition models are implemented by their respective `AcquisitionModel` classes.

3.3. Reconstructors

Reconstructor classes implement algorithms for solving equation (1). In the current SIRF version, we expose Gadgetron MR reconstructors that implement direct algorithms for Cartesian acquisitions using Fourier transforms.

Planned future versions will also expose iterative algorithms for Cartesian and non-Cartesian acquisitions. For PET, the SIRF Filtered Back Projection reconstructor implements a direct algorithm. Other reconstructors implement iterative algorithms maximising the (optionally penalised) Poisson log-likelihood (see §3.4), with the possibility of using ordered subsets [33].

With SIRF iterative reconstructors, the user has an option of performing one iteration at a time in a loop, which allows them to inspect the current state of the iterated image, apply their own filtering and so on.

3.4. Objective functions

SIRF iterative reconstructors maximise a certain objective function related to the acquisition model (1). The user is provided with facilities to compute the value of this function and its gradient, so that they can employ their own or third-party optimisation algorithms. Penalty terms can be added to the objective function, which opens a way to employing images produced by an MR reconstructor as anatomical priors for PET reconstruction [34, 35, 13], see also §5.1.2.

3.5. Registration and resampling

As previously mentioned, through the wrapping of NiftyReg SIRF provides registration and resampling functionality. For rigid and affine registrations, the user may use `NiftyAladinSym`; for non-rigid registration, the user may use `NiftyF3dSym`. Lastly, resampling can be carried out with the use of `NiftyResample`. Example usage of the registration and resampling functionality can be found in §4.3.

Once registrations have been performed, it is clearly important to be able to manipulate the motion information. Affine transformation matrices can therefore be handled with `AffineTransformation`, and displacement and deformation field images are handled with `NiftiImageData3DDisplacement` and `NiftiImageData3DDeformation`, respectively.

It is possible to convert between these different transformation types (although, naturally, it is not possible to convert from non-rigid transformations back to transformation matrices). Moreover, SIRF provides functionality to compose multiple transformations (of differing types if desired), as well as averaging affine transformation matrices via their quaternions.

3.6. Other functionality

In addition to the functionality described in §3.1-§3.5, Python and MATLAB interfaces provide functionality for data input/output, object cloning, inspecting data dimensions and so on.

Data-access functionality warrants a special note, as all the actual acquisition and image data are handled exclusively by the engines. For performance reasons, the user does not have direct access to the underlying engine-specific data structures and normally would not need to know whether it is stored in the memory, in a file, in GPU memory, etc. Each data container has a method `as_array` that returns a copy of its data as a Python or MATLAB array, and a method `fill` that fills the container with the data from a user's array. In addition, SIRF contains converters to directly go between data structures of the different engines, if possible, albeit usually with some loss of associated meta-data.

4. Examples of usage

4.1. PET

Although many other radiotracers are available, PET is most often used to scan patients who have been injected with an ^{18}F-labelled Fluorodeoxyglucose (FDG) solution that provides information about glucose metabolism. FDG is commonly used to diagnose and stage tumours in oncology, but it also has uses in various other clinical domains such as in cardiology, which helps to assess the myocardial viability. PET data acquisition is usually carried out for about 30 minutes, about 60 minutes after the tracer injection.

Here, we show how an FDG-PET scan acquired on a 3T Siemens Biograph mMR [25] can be reconstructed using an ordered subsets expectation maximisation (OSEM) algorithm [33]. The following pseudo-code uses the Python interface of SIRF but can easily be adapted for MATLAB. Only parts of the necessary code are provided here. For a detailed line-by-line tutorial refer to the example file `reconstruct_from_listmode.py`.

The following files are required for this example reconstruction (the variables for the filenames for the following code snippets are given in brackets):

- Normalisation file in Siemens interfile format (`norm_file`)
- Attenuation map for the subject (usually obtained from a dedicated MR scan) in interfile format (`human_attenuation_file`).

- Attenuation map for the table (and MR receive coils for head scans) in interfile format (`hardware_attenuation_file`).
- Listmode data in Siemens PETLINK format with Siemens interfile-like header (`listmode_file`).
- Template for the projection data calculation in interfile format. This has to fit to the scanner geometry. Examples for this can be found in the sub-module `data`, for instance mMR_template_span11.hs in the `data/examples/PET/mMR/` folder.

As a first step, the listmode data are transformed to sinogram space using:

```
import sirf.STIR as PET
lm2sino = PET.ListmodeToSinograms()
```

The `lm2sino` object is then provided with the listmode file (`listmode_file`), the template file (`template_file`) and information about the time interval which shall be transformed to sinogram space. The calculation of sinogram data is performed by the method `process`, and the calculated data are returned by the method `get_output`. After calculating the sinogram data, a sinogram of the mean of the random events can also be estimated [36] using the `lm2sino` object:

```
randoms = lm2sino.estimate_randoms()
```

The attenuation maps can be read in as `ImageData` objects and then combined:

```
attn_image_hum = PET.ImageData(human_attenuation_file)
attn_image_hw = PET.ImageData(hardware_attenuation_file)
attn_image = attn_image_hum + attn_image_hw
```

Figure 2 shows an example for the combined attenuation map.

After combining the attenuation data and the information from the normalisation file, a reconstruction object for OSEM is created:

```
recon = PET.OSMAPOSLReconstructor()
recon.set_num_subsets(21)
recon.set_num_subiterations(63)
```

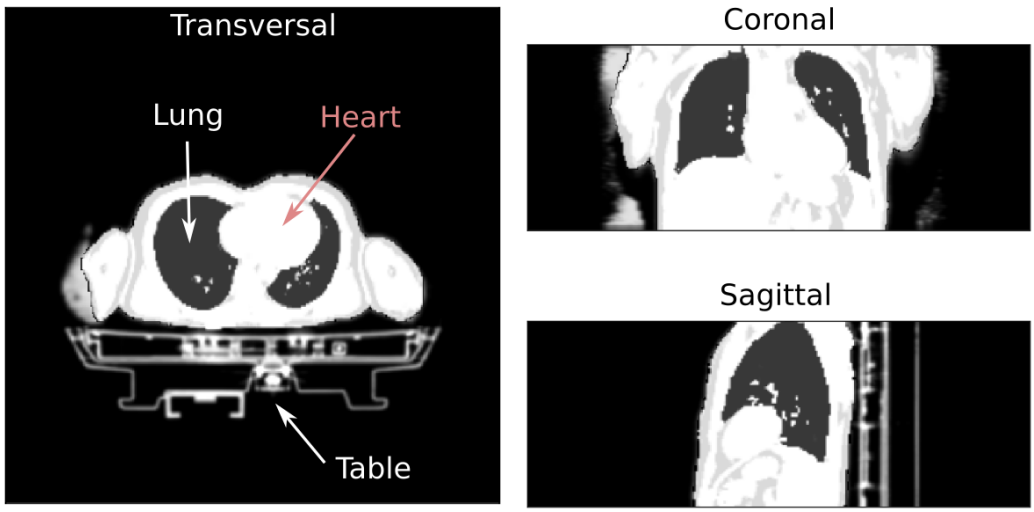

Figure 2: Combined attenuation map showing both attenuation coefficients for the subject and the scanner table in three different orientations.

The `OSMAPOSL` reconstructor is an implementation using ordered subsets of Green's One Step Late algorithm for Maximum A Posteriori reconstruction [37] which reduces to OSEM if no prior is set. Although this algorithm does not converge to the global maximum of the objective function but to an approximate maximum, it is still widely used in practice because it substantially speeds-up the calculations. See §5 for examples of other algorithms.

The objective function is defined as the logarithm of the Poisson likelihood, using a default (ray-tracing) linear acquisition model, with the data set to `acq_data`, the output returned by `lm2sino.get_output()`:

```
obj_fun = PET.make_Poisson_loglikelihood(acq_data)
```

The objective function requires an acquisition model. Here, we explicitly specify to use forward projection by a ray-tracing matrix multiplication:

```
acq_model = AcquisitionModelUsingRayTracingMatrix()
obj_fun.set_acquisition_model(acq_model)
```

but see the example script for details on how to include extra components in the acquisition model, such as attenuation and randoms. Results shown here also include Compton scatter in the acquisition model, in which the scatter is estimated using the Single Scatter Simulation method [38] as implemented

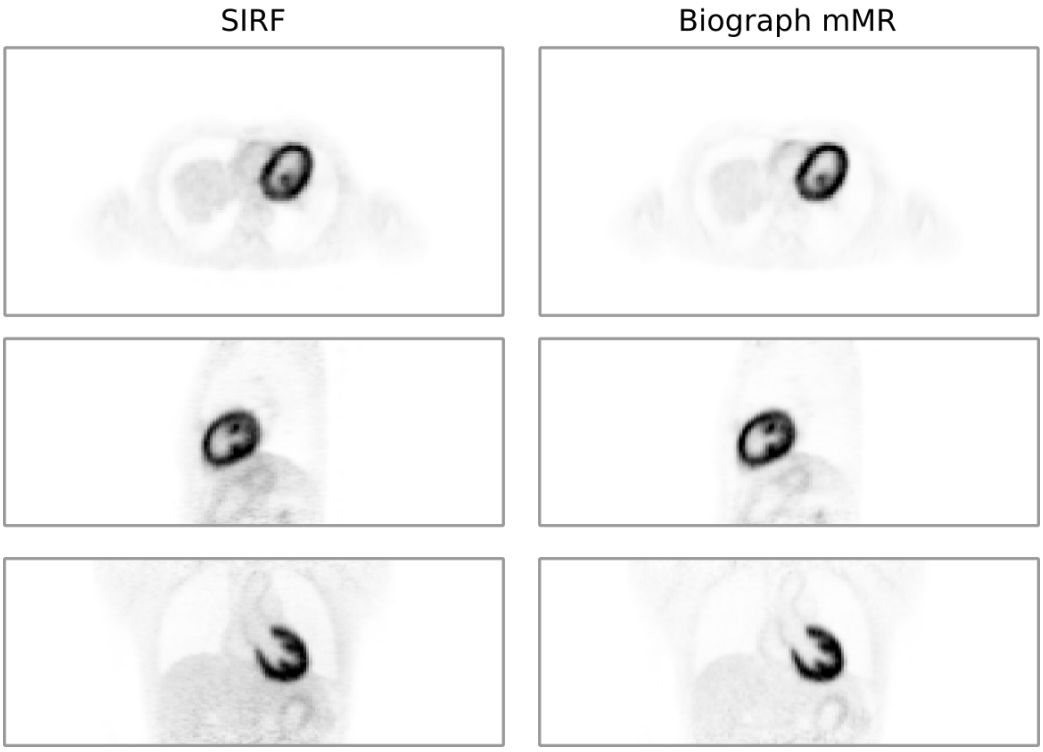

Figure 3: Transversal, sagittal and coronal views of the reconstructed PET images with SIRF reconstructions on the left and the reconstructions by the scanner software on the right. The high uptake of FDG in the heart muscle can be clearly seen.

in STIR [39], although this functionality is currently still on a development branch of SIRF and STIR.

Following the set up of the reconstruction object, the iterative reconstruction can be carried out and the reconstructed PET data can be obtained:

```
recon.process()
image_data = recon.get_current_estimate()
```

The final reconstructed PET image is shown in Figure 3. In addition, the standard PET reconstruction from the PET-MR scanner is also shown for comparison purposes. Post-filtering after image reconstruction was different for the images.

4.2. MR

Cardiac function – the contracting motion of the heart during the cardiac cycle – is an important clinical parameter when checking for potential problems in the heart. MR is often used for cardiac functional imaging (so-called cine imaging) due to its excellent soft tissue contrast. In order to minimise respiratory motion artefacts, 2D cine imaging is carried out during breath-hold. MR raw data are acquired in k-space over multiple cardiac cycles, and data from the same cardiac phase are retrospectively combined for image reconstruction. This yields dynamic 2D images showing the heart at different phases of the cardiac cycle. Parallel imaging techniques, such as GRAPPA [40], undersample data to speed up the acquisition and reduce breath-hold duration while maintaining high spatial and temporal resolution [10].

Here, we show how to reconstruct a 2D cine MR scan acquired with a 3T mMR Biograph during a 10 s breath-hold (repetition/echo time: 3.4 / 1.5 ms, bSSFP-sequence, flip angle: 50°, image resolution: 1.3 mm, slice thickness: 6 mm, undersampling factor: 2) using the Python interface of SIRF.

As a first step, the filename of the MR raw data file in ISMRMRD format [41] is passed to the MR `AcquisitionData` class to create the corresponding object:

```
import sirf.Gadgetron as MR
acq_data = MR.AcquisitionData(mr_file)
```

Preprocessing of the data, such as removal of readout oversampling and noise decorrelation for multi-coil data, is carried out:

```
preproc_data = MR.preprocess_acquisition_data(acq_data)
```

The undersampling of the acquired k-space leads to a violation of the Nyquist-Shannon sampling theorem and hence to "undersampling/aliasing" artefacts in the reconstructed image. GRAPPA utilises spatial information from the local receiver coils to minimise these artefacts [42]. A full GRAPPA reconstruction is implemented in Gadgetron and can be called in a straightforward manner from the Python or Matlab interface of SIRF by creating a `CartesianGRAPPAReconstructor` object:

```
recon = MR.CartesianGRAPPAReconstructor()
```

or by specifying the gadgets explicitly

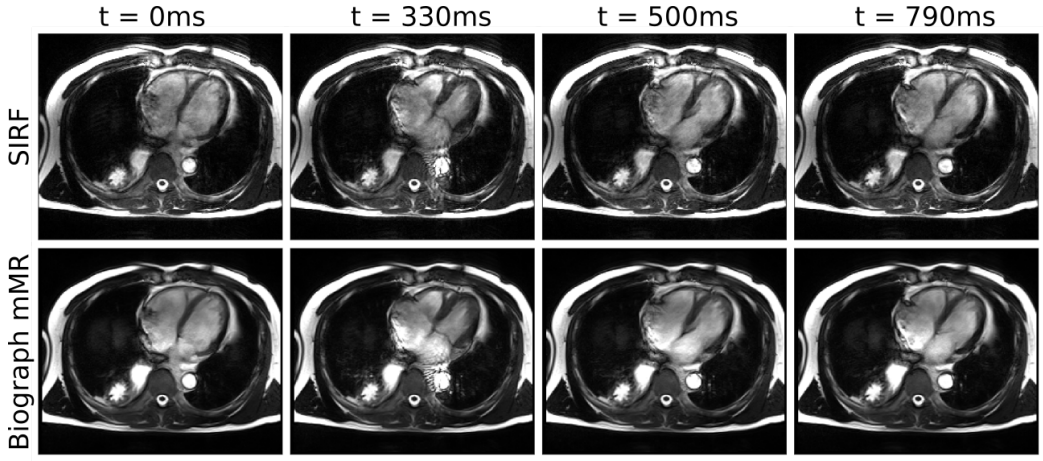

Figure 4: Reconstructed time frames of a MR cine acquisition using SIRF compared to the standard image reconstruction of the 3T Siemens mMR Biograph.

```
recon_gadgets = ['AcquisitionAccumulateTriggerGadget',
    'BucketToBufferGadget',
    'GenericReconCartesianReferencePrepGadget',
    'GenericReconCartesianGrappaGadget',
    'GenericReconFieldOfViewAdjustmentGadget',
    'GenericReconImageArrayScalingGadget',
    'ImageArraySplitGadget'
    ]

recon = MR.Reconstructor(recon_gadgets)
```

Once the preprocessed data are passed to this object, the reconstruction is carried out and the reconstructed image data is retrieved:

```
recon.set_input(preproc_data)
recon.process()
image_data = recon.get_output()
```

Figure 4 shows the cine data reconstructed with SIRF compared to the standard reconstruction from the MR scanner. In contrast to the standard image reconstruction performed on the PET-MR scanner, SIRF also provides further information, such as a g-factor map that describes the noise

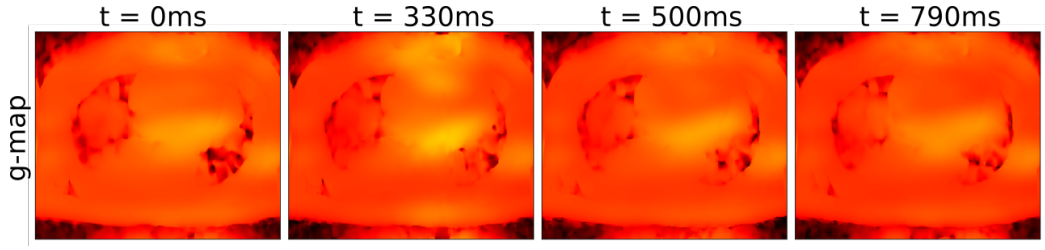

Figure 5: G-factor maps for the cine time frames shown in Figure 4. These maps provide a measure of the noise amplification related to the conditioning of the inverse problem, in this case a GRAPPA reconstruction.

amplification that results from reconstructing undersampled data. The g-factor depends upon the coil geometry, coil positioning, undersampling factor, undersampling pattern and the object. (Figure 5). An inhomogeneous distribution of values in the g-factor map can be, for example, an indicator that the sequence parameters were not optimised for the used receiver coil configuration. The calculation of the g-factor map can be enabled for the GRAPPA gadget using:

```
GenericReconCartesianGrappaGadget(send_out_gfactor=true)
```

and the g-factor maps are then retrieved using

```
gfactor_data = recon.get_output('gfactor')
```

The final number of dynamic 2D cine images depends on the subject's heart rate. In order to make the images more easily comparable between different subjects, the data are commonly interpolated to a predefined number of dynamic images. This can also be done using SIRF by adding the following lines to the reconstruction gadget chain:

```
'PhysioInterpolationGadget(phases=30, mode=0,
        first_beat_on_trigger=true, interp_method=BSpline)'
```

This yields 30 dynamic cine images which can be retrieved using:

```
image_data = recon.get_output('Image PhysioInterp')
```

4.3. PET-MR

Using SIRF, it is possible to perform motion-corrected static and dynamic PET reconstructions. This is highlighted in [43] (and briefly shown here), in which the example of head motion is given.

In this method, motion-free time frames are supplied from an external source (such as by using principal component analysis (PCA) on the PET listmode data [44, 45]). Using these time frames, the listmode data are then binned into corresponding sinograms. Non-attenuation corrected (NAC) reconstructions are performed on a frame-by-frame basis, and the resulting images are registered to a reference frame using rigid registration as follows:

```
import sirf.Reg as reg
algo = reg.NiftyAladinSym()
algo.set_parameter("SetPerformRigid","1")
algo.set_parameter("SetPerformAffine","0")
algo.set_floating_image(NAC_i)
algo.set_reference_image(NAC_ref)
algo.process()
TM = algo.get_transformation_matrix_forward()
```

Frame-by-frame attenuation-corrected (AC) reconstructions can then be performed by first transforming the MRAC into the space of the individual PET frames using the extracted transformation matrix, as such:

```
res = reg.NiftyResample()
res.set_floating_image(MRAC)
res.set_reference_image(NAC_i)
res.add_transformation(TM)
res.set_interpolation_type_to_linear()
res.process()
res.get_output().write("MRAC_def.nii")
```

Following the AC reconstruction of each of the frames, they can then be resampled back into the reference space, and indirect parametric estimation can be performed via STIR's Patlak analysis [46].

Figure 6 shows the movement during the scan by comparing the reference frame (left) to another frame earlier in the scan (middle). The earlier scan is then registered to the reference frame to account for motion (right).

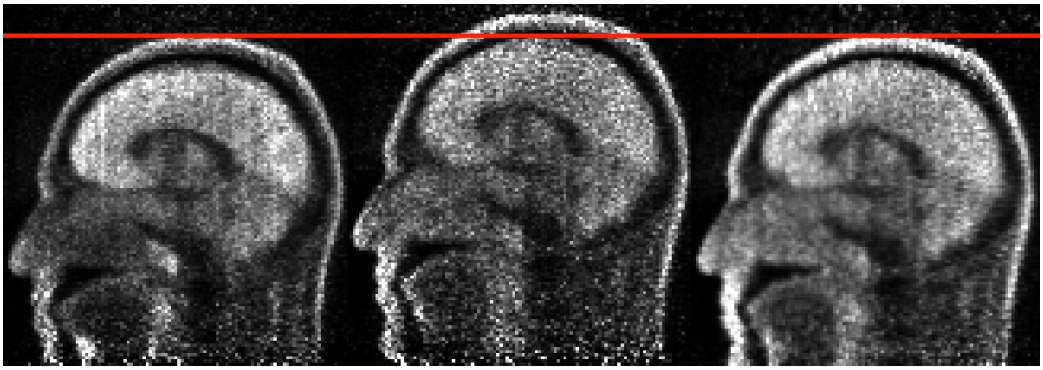

Figure 6: NAC reconstructions of the reference frame (left) and an earlier frame (middle), where movement occurred between the two. Finally, the early frame registered to the reference frame (right).

A well-known problem with the above methodology is that the registration of early NAC frames is often not reliable as these images are usually too noisy and the activity distribution is changing rapidly over time. In such circumstances, PET-MR is particularly useful if MR acquisitions of high temporal resolution, such as arterial spin labelling (ASL), were acquired during these early PET frames.

Figure 7 then shows the movement throughout the whole scan, as detected by frame-by-frame registration of NAC and ASL images.

Rapid movement according to the ASL registrations (at 900 s, for example) corresponds to gaps in the NAC data, suggesting that PCA was successfully used determine motion-free time frames for the PET reconstructions.

The NAC registrations appear to imply that rapid jittering motion is present at the beginning of the scan. This is likely a demonstration of the difficulty of registering early NAC images. This highlights the advantages of using complimentary MR information when available for PET-MR acquisitions.

Figure 8 shows indirect parametric reconstructions using Patlak analysis. On the left, motion correction was used (with motion information from a combination of NAC and ASL registrations), whereas no motion correction was used on the right-hand side. The superior section of the brain is underestimated when motion correction is not used as motion is in the superior-inferior direction (corresponding to the z-direction in Figure 7). This, again, shows the benefit of using complimentary PET-MR motion information.

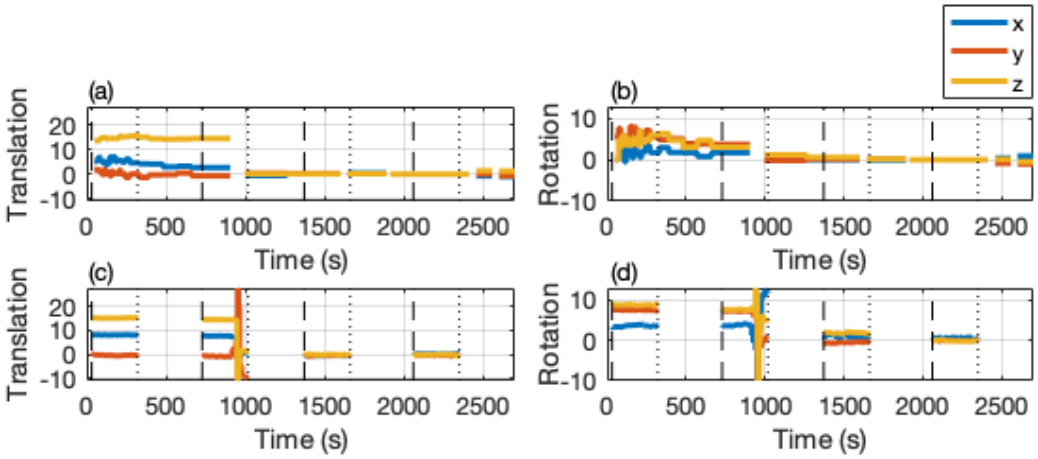

Figure 7: Motion based on frame-by-frame registrations. The top and bottom rows represent motion from registering NAC and ASL images, respectively. Further, the left-hand column shows translations (mm), whereas the right-hand column shows rotational motion via Euler angles (degrees). Gaps in the NAC data are present because motion-free frames have been determined (in this example with PCA) and periods of motion have been removed. Gaps in ASL data are present because ASL images were not acquired for the whole duration of the scan. Vertical lines show the start and end of ASL acquisitions (longer and shorter dashes for the start and end, respectively).

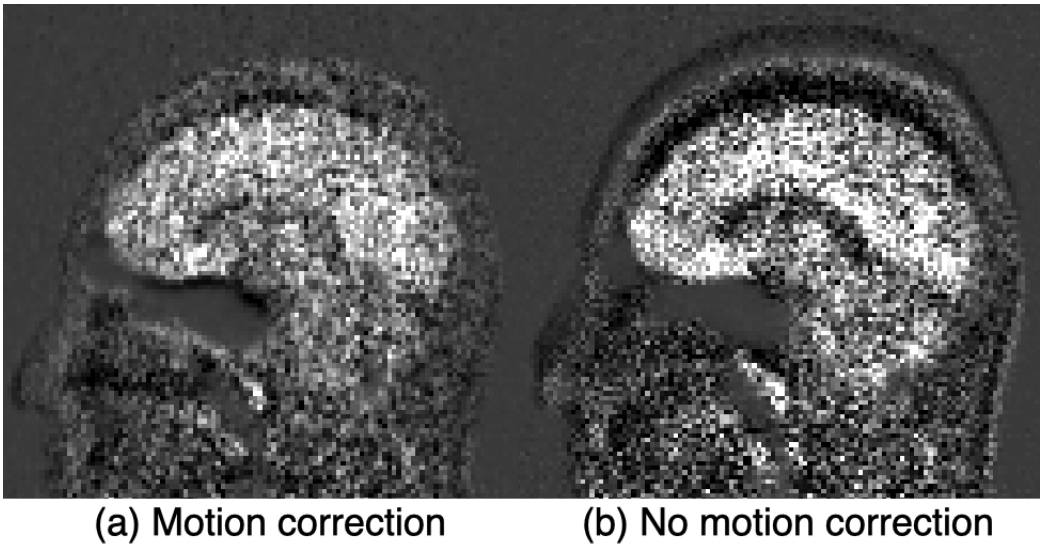

Figure 8: Indirect parametric reconstructions both (a) with and (b) without motion correction. Without motion correction, the top of the brain is underestimated due to motion in the inferior-superior direction.

5. Advanced usage

5.1. Implementing custom reconstruction algorithms

By using the functionality mentioned in this paper, it is possible to implement reconstruction algorithms directly in MATLAB or Python, for instance using `AcquisitionModel` operations, or in terms of the objective function and its `get_subset_gradient` method (see, for instance, the steepest ascent example).

In addition, it is often also convenient to use parts of an existing algorithm. Therefore, SIRF iterative reconstructors expose functionality such as:

- `update` and `update_current_estimate` to compute one update
- `get_current_estimate` and `set_current_estimate` to modify the estimate, for instance with filtering.

During a recent Hackathon [47], such functionality was used to implement new algorithms in Python:

- Total Variation regularised least squares with fast iterative shrinkage-thresholding algorithm (FISTA) [48], and
- De Pierro's maximum *a posteriori* expectation maximisation (MAP-EM) algorithm [49].

5.1.1. Use of CCPi optimisation and regularisation algorithms

The Collaborative Computational Project in Tomographic Imaging (CCPi, `https://www.ccpi.ac.uk/`), provides a collection of software modules to tackle different aspects of the X-ray CT data analysis pipeline from preprocessing to reconstruction to visualisation; this collection is referred to as the Core Imaging Library (CIL), [50]. The relevant parts of CIL employed to integrate into SIRF are the following:

- CCPi-Framework: An object-oriented optimisation framework [1]

[1] `https://github.com/vais-ral/CCPi-Framework/`

- CCPi-Regularisation-Toolkit: A set of CPU/GPU optimised regularisation modules for iterative image reconstruction and other image processing tasks [51] [2]

- CCPi-FrameworkPlugins: A set of wrappers to allow usage of CCPi-Regularisation-Toolkit corresponding modules in the CCPi-Framework [3]

The CCPi-Framework is developed in Python following an object-orientated approach and most of the classes follow the same naming convention as in SIRF, including `DataContainer`, `ImageData` and `AcquisitionData`. The CCPi-Framework was developed on the basis of the discrete imaging model $A(x) = b$, as discussed in §3.2. While in SIRF the operator A is called 'AcquisitionModel', in the CCPi-Framework it is called 'Operator' as it was decided to use the names of the underlying mathematical objects. The CCPi-Framework provides a range of generic implementations of optimisation algorithms to address a variety of smooth and non-smooth optimisation problems relevant for reconstruction purposes. One can seamlessly use SIRF objects in CCPi-Framework algorithms thanks to the dynamic/duck typing of Python and to the introduction of aliases of methods that are in CCPi-Framework objects to methods in SIRF objects. For example, the `forward` method of an `AcquisitionModel` is aliased as `direct`, and the `backward` method as `adjoint`.

The CCPi-Regularisation-Toolkit provides a range of different regularisation modules including Total Variation (TV) [52] and variants thereof, implemented in C/CUDA for accelerated computation on multi-core CPUs and GPUs. Through the CCPi-FrameworkPlugins wrappers, these tools are available in the CCPi-Framework and as a result can also be used from SIRF. In the remainder of the paper we will simply refer to CIL as meaning the collection of modules described above.

With the integration of CIL into SIRF, a whole plethora of optimisation algorithms and regularisers are now available. To demonstrate these, the Gradient Descent (GD) and FISTA algorithms [48] were used to solve the least squares and a regularised least-squared problem, respectively.

FISTA can be used to minimise objective functions which are the sum of a differentiable first term and a second term which is not necessarily differ-

[2] `https://github.com/vais-ral/CCPi-Regularisation-Toolkit`

[3] `https://github.com/vais-ral/CCPi-FrameworkPlugins`

entiable. The algorithm alternates between the two terms of the objective function allowing for the use non-differentiable regularisers, such as TV, to encourage preservation of edges in the reconstruction. The GD and FISTA algorithms of CIL are provided by the CCPi-Framework, whereas the TV regulariser and in particular its proximal operator is provided by the CCPi-Regularisation-Toolkit by implementing the Fast Gradient Projection algorithm [53].

Figure 9 shows the results of the reconstruction of a simulation of the Shepp-Logan phantom using CIL algorithms. First, we used SIRF to reconstruct Shepp-Logan phantom x_{SL} from the acquisition data generated by the ISMRMRD utility `ismrmrd_generate_cartesian_shepp_logan` and then used SIRF AcquisitionModel class to generate simulated acquisition data $b = A(x_{SL})$ (in the notation of (1)). Then, we applied the GD algorithm to the least squares problem (minimisation of $\|A(x) - b\|_2^2$) and the FISTA algorithm to the TV-regularised least-squares problem, with the TV term operating only on the real part of the solution x. The results for Figure 9a and 9b were obtained for 100 and 50 iterations, respectively. The edge-preserving effect of the TV-regulariser is clearly seen in Figure 9b. These examples are provided as proof of concept of the versatile reconstruction options enabled by the integration of CIL into SIRF. Future work will expand the set of algorithms from CIL available in SIRF and test their application to real data. The code for this example can be found in the SIRF-Contribs repository.

5.1.2. De Pierro's MAP-EM

The goal here was to incorporate new MAP-EM algorithms that could be used both independently and as a part of synergistic PET-MR reconstruction algorithms and to demonstrate the prototyping potential of SIRF.

The code was implemented using SIRF's pre-existing example Python code for the one-step-late isotropically-weighted quadratically-penalised MAP-EM algorithm as a starting point. The code was then tested for two cases:

1. Uniform (i.e., isotropic) weights; and
2. Bowsher weights [54], calculated using a prior class also introduced during the Hackathon.

Figure 10 shows the use of the newly implemented algorithm. In Figure 10b a standard OSEM reconstructed image is displayed next to images reconstructed using the uniform (Figure 10c) and Bowsher (Figure 10d) weights of the De Pierro's algorithm. In Figure 10a, the MPRAGE that was used

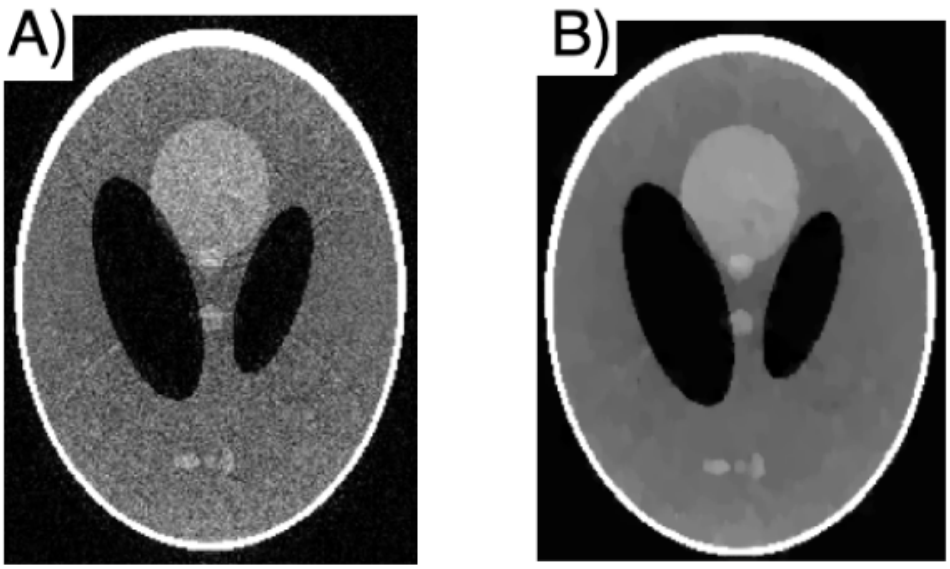

Figure 9: Reconstructions of an MR simulation of the Shepp-Logan phantom using newly implemented algorithms that are now available in SIRF thanks to the integration of CIL. The results correspond to (a) the gradient descent (after 100 iterations) and (b) Total Variation regularised reconstruction using FISTA (after 50 iterations).

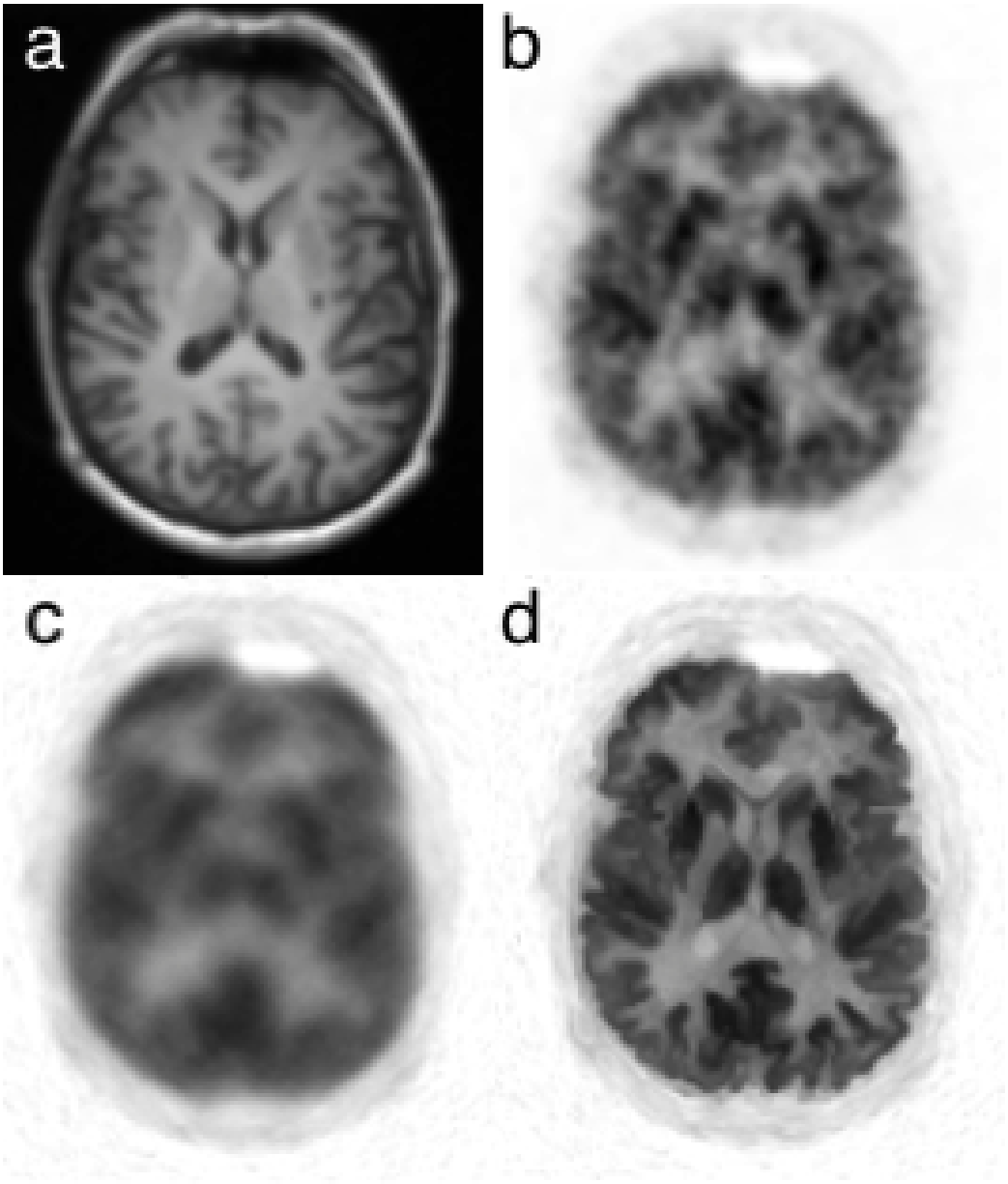

Figure 10: Reconstructions using the recently-implemented De Pierro's MAP-EM algorithm. (a) The anatomical MR image (MPRAGE), (b) a standard OSEM reconstruction for comparison, (c) a reconstruction using De Pierro's MAP-EM with uniform weights, (d) a reconstruction using De Pierro's MAP-EM with Bowsher weights (using the MPRAGE).

to calculate the Bowsher weights is also shown. From these images, it can be seen that the weight for both Figures 10b and 10c is substantially strong; the objective of this task, however, was not to fine-tune the parameters, but simply to demonstrate the images reconstructed by this algorithm. The code for this implementation can be found in our SIRF-Contribs repository.

6. SIRF software management and deployment routes

6.1. Software distribution via GitHub

We use GitHub with its issue tracking, milestones and release management facilities (visit `https://github.com/CCPPETMR/SIRF`). SIRF contains a series of examples that also serve as manual tests. In addition, SIRF has a test-suite for automatic testing using CTest. We also employ the Travis Continuous Integration system in which every GitHub code change (commit) is compiled and tested automatically. The build/test status is reported and can be checked online at any time (see `https://travis-ci.org/CCPPETMR`). Furthermore, we maintain a separate repository *SIRF-Exercises* which contains Python examples using Jupyter notebooks, `https://jupyter.org`. These examples have been used as the basis for hands-on training sessions.

6.2. SuperBuild installation

The functionality of SIRF relies upon a multitude of open-source software packages. In addition to the reconstruction packages (currently STIR and Gadgetron), SIRF requires a number of general-purpose C++ libraries such as Boost, HDF5 etc. All these SIRF dependencies need to be built using appropriate versions and configurations, and in turn have their own dependencies. To reduce the burden on the user of providing all the associated packages, *SIRF-SuperBuild* has been developed and it is the recommended route if building SIRF from source.

The SIRF-SuperBuild is a CMake project which downloads, configures, builds and installs all dependencies required to build SIRF. Once these packages are installed, the SuperBuild will build SIRF. This mechanism works for Linux and macOS, with Windows support pending. By default, the latest release of SIRF will be installed. The user can select whether to build the Python and/or MATLAB bindings using CMake. For developers and advanced users, it is possible to configure the SuperBuild to reuse packages that already exist on the system – skipping the building of particular dependencies – and also to select development package versions.

Changes to the SuperBuild are also tested on Travis. This includes automatic testing of dependencies.

6.3. Docker

Docker (available from `https://docs.docker.com/install`) is a container-based low-overhead alternative to Virtual Machines (§6.4). The command `docker pull ccppetmr/sirf` will download the latest ready-made SIRF docker image. The SuperBuild repository (§6.2) contains the docker source code and documentation. Docker images are automatically built on Travis. Various different SIRF images (corresponding to different versions and software) are available and listed at `https://hub.docker.com/r/ccppetmr/sirf`.

The docker images can be used to run Gadgetron as a server (for instance on Windows where building Gadgetron is difficult) and/or a Jupyter notebook server such that a user can run SIRF from a web browser running on their local machine without further configuration. It is also possible to run Python scripts via docker without Jupyter, although in this case display of any figures needs X forwarding or remote desktop set-up. Docker images with GPU support are also provided for systems which support nvidia-docker (`https://github.com/NVIDIA/nvidia-docker`).

6.4. Virtual machine

A binary installation of SIRF is provided by means of a VirtualBox virtual machine (VM), running Ubuntu 18.04 LTS, which can be downloaded from the project's website `http://www.ccppetmr.ac.uk`. The VM contains a (minimal) Linux system complete with developer tools, X windowing system and Jupyter notebook server.

The *CCPPETMR_VM* GitHub project contains configurations and scripts to create a VM and build SIRF using the SIRF-SuperBuild project. A new VM is created by means of the Vagrant application `https://www.vagrantup.com`, which takes as the input a text file that describes the type of machine and how to configure and provision it. A simple bash script, run by Vagrant, configures the SuperBuild and builds SIRF on the VM.

The VM can be used as a fully-fledged Linux machine. Alternatively, it can be used to run the Gadgetron and/or Jupyter notebook server like the Docker image (for instance on Windows versions, where Docker needs VirtualBox).

6.5. Microsoft Azure cloud

SIRF can also be deployed on Azure cloud infrastructure. We use an application called Terraform, see `https://www.terraform.io`, to create a virtual machine running SIRF on Azure. Terraform handles the interaction with Azure to provision a virtual machine in the cloud. SIRF is then built using the scripting mechanism for virtual machine configuration as described in §6.4. In addition to SIRF, the Azure virtual machine automatically runs a Jupyter notebook server which can be accessed through any browser worldwide. Remote Desktop, to access the machine as if it was a local desktop, has also been enabled. Code to deploy SIRF virtual machines on cloud infrastructure besides Azure will be published on GitHub in the future.

7. Discussion

In this paper, we presented release 2.1.0 of SIRF, open-source software for PET-MR image reconstruction. SIRF is designed to provide a simple and object-oriented framework to enable implementation of novel algorithms exploring joint processing of dual-modality imaging data. Data access and manipulation is consistent between the two modalities. SIRF has already been used in several teaching courses as well as in hackathon events [47], showing its potential for rapid code development that can then be used for testing in patient data. SIRF relies on other open-source software packages ("engines") for many of the underlying operations. This has the benefit of capitalising on the active development of specialist tools. On the other hand, it inherits the capabilities and limitations of these engines. For instance, while support for PET and MR data of the Siemens mMR Biograph [25] is mostly complete, support for the GE Signa PET/MR [27] has not been made available, yet. Recently, there has been good progress regarding the PET data of this scanner model in STIR [55, 56], and the convertor from the GE native MR data format to ISMRMRD is under active development [57]. The architecture of SIRF also allows adding other "engines" in the future with initial discussions started related to CASToR [20]. To cope with version changes in the employed engine packages, SIRF releases link to specific versions. At the same time, we provide advanced users with the opportunity to try the latest versions of all used software, in which case we, of course, cannot always guarantee that everything will work.

The current release of SIRF provides the basis for the implementation of existing and novel algorithms for exploiting the synergy in multi-modality

imaging data with initial work in combining high quality side information [58, 59, 60, 61] and joint simultaneous reconstruction of PET and MRI images [15, 19]. Making these algorithms freely available to the wider community of researchers will be an important step towards assessing their benefits for clinical application and translation.

In addition to PET-MR, there is active research in exploiting synergy in other modalities, where multiple parameters of one underlying object can be extracted from multiple acquired data-sets, such as multi-sequence MR [62, 16, 63, 64], dual-energy/spectral CT [65, 66, 67, 68], and even CT-MR [69, 70]. Many of the algorithms developed in PET-MR research are transferable to those research areas and vice versa. The architecture of SIRF would enable incorporating these modalities as well as including PET/CT, SPECT/CT and SPECT/MR in a common framework in the future.

Another focus for the development of SIRF in the future will be around deployment options. Ensuring SIRF is readily available and simple to use by a non-technical researcher will allow SIRF to be used in higher-level medical research. The use of SIRF in such research potentially enables superior quantitative accuracy in medical investigations, and the ability to demonstrate the utility of the implemented algorithms in a clinical research context before being demonstrated in routine clinical practice. Existing deployment options were outlined in §6, but in the future may be augmented by deploying Gadgetron's inline facilities, or as a hosted reconstruction service. A production deployment of SIRF is a possibility, which is however not within the grasp of our team, mainly because of regulatory issues. This is less of an issue for pre-clinical systems.

8. Conclusion

The open-source Synergistic Image Reconstruction Framework (SIRF) is a new software tool for research in image reconstruction and related manipulations including motion estimation and compensation. It currently targets joint image reconstruction of PET-MR data but with a view towards extension to other cases where synergistic reconstruction can be beneficial. SIRF enables researchers to explore the benefits of synergistic reconstruction in a framework that has been successfully used for educational purposes whilst being powerful enough for processing data from state-of-the-art scanners.

Acknowledgements

CCP PET-MR is funded by the EPSRC (grant EP/M022587/1) and the associated Software Flagship project "A framework for efficient synergistic spatiotemporal reconstruction of PET-MR dynamic data" (grant EP/P022200/1). This work made use of computational support by CoSeC, the Computational Science Centre for Research Communities. We thank Siemens Healthineers and General Electric Healthcare for information and support with file formats. We also wish to thank Michael Hansen, David Hansen and Hui Xue for help with Gadgetron, and Camila Munoz, James Bland and Abolfazl Mehranian for helping with the coding during the hackathon leading to §5.1.2, and the wider CCP PET-MR community for feedback. We wish to acknowlege CCPi with the funding support by the EPSRC grants (EP/P02226X/1) 'Reconstruction Toolkit for Multichannel CT' and the network initiative (EP/M022498/1). Lastly, we wish to thank St Thomas' PET Centre for the use of their patient data.

Appendix A. Object-oriented paradigm

This appendix is mainly intended for readers who are not familiar with Object-Oriented programming.

In Object-Oriented approach, instead of having data containers (arrays, files etc.) and functions that operate on them, one employs *objects*, which contain data and come along with sets of functions, called *methods*, which operate on data. In object-oriented languages, an object type is referred to as its *class*. Each object has a special method called constructor, which has the same name as the object class name and must be called to create that object. For example, to create an object of class `ImageData` that handles MR image data and fill it with data stored in the HDF5 file `my_image.h5` one needs to call for an assignment, such as:

```
image = ImageData('my_image.h5')
```

We note that an MR ImageData object contains not only the voxel values, but also a number of parameters specified by the ISMRMRD format of MR image data [41]. The object data is encapsulated, i.e. it is not directly accessible from the user's code (being handled mostly by the underpinning `C++` code) and is processed by the object methods. For example, to display the data encapsulated by image, one needs to call its method `show()`:

```
image.show()
```

and if anyone needs to copy the data into a MATLAB array, they would use the method `as_array()`:

```
image_data_array = image.as_array()
```

Parameters of objects are modified/accessed via set/get methods (mutators and accessors). For example, the value of an objective function handled by the object named `obj_fun` on an image data object is computed by its method `get_value()` as:

```
obj_fun_value = obj_fun.get_value(image)
```

The mutators are also responsible for basic error checking.

Some classes are derived from other classes, which means that they contain all the methods of the classes they are derived from (i.e. *inherited* methods). For example, the class `AcquisitionModelUsingMatrix` is derived from the class `AcquisitionModel`, so an object of the former class inherits all methods of the latter.

Bibliography

[1] K. Thielemans, C. Tsoumpas, S. Mustafovic, T. Beisel, P. Aguiar, N. Dikaios, M. W. Jacobson, Physics in Medicine and Biology 57 (4) (2012) 867–883.

[2] M. S. Hansen, T. S. Sørensen, Magnetic Resonance in Medicine 69 (6) (2013) 1768–1776.

[3] H. F. Wehrl, A. W. Sauter, M. R. Divine, B. J. Pichler, Journal of Nuclear Medicine 56 (2) (2015) 165–168.

[4] Z. Chen, S. D. Jamadar, S. Li, F. Sforazzini, J. Baran, N. Ferris, N. J. Shah, G. F. Egan, Human Brain Mapping 39 (12) (2018) 5126–5144.

[5] F. Nensa, K. Beiderwellen, P. Heusch, A. Wetter, Diagnostic and Interventional Radiology (Ankara, Turkey) 20 (5) (2014) 438–447.

[6] P. M. Robson, D. Dey, D. E. Newby, D. Berman, D. Li, Z. A. Fayad, M. R. Dweck, JACC: Cardiovascular Imaging 10 (10 Part A) (2017) 1165–1179.

[7] H. Barthel, O. Sabri, in: L. Umutlu, K. Herrmann (Eds.), PET/MR Imaging: Current and Emerging Applications, Springer International Publishing, Cham, 2018, pp. 99–106.

[8] X. Y. Zhang, Z. L. Yang, G. M. Lu, G. F. Yang, L. J. Zhang, Frontiers in Molecular Neuroscience 10 (2017).

[9] J. Qi, R. M. Leahy, Physics in Medicine and Biology 51 (15) (2006) R541–R578.

[10] K. P. Pruessmann, NMR in biomedicine 19 (3) (2006) 288–299.

[11] J. A. Fessler, IEEE Signal Processing Magazine 27 (4) (2010) 81–89.

[12] K. G. Hollingsworth, Physics in Medicine and Biology 60 (21) (2015) R297–R322.

[13] B. Bai, Q. Li, R. M. Leahy, Seminars in Nuclear Medicine 43 (1) (2013) 30–44.

[14] A. Mehranian, E. De Vita, R. Neji, C. J. McGinnity, A. Hammers, A. J. Reader, in: PSMR 2018, Isola de Elba, Italy, 2018, p. 3.

[15] M. J. Ehrhardt, K. Thielemans, L. Pizarro, D. Atkinson, S. Ourselin, B. F. Hutton, S. R. Arridge, Inverse Problems 31 (2015) 015001.

[16] F. Knoll, M. Holler, T. Koesters, R. Otazo, K. Bredies, D. K. Sodickson, IEEE Transactions on Medical Imaging 36 (1) (2017) 1–16.

[17] J. Rasch, E.-M. Brinkmann, M. Burger, Inverse Problems 34 (1) (2017) 014001.

[18] A. Mehranian, M. A. Belzunce, C. Prieto, A. Hammers, A. J. Reader, IEEE Transactions on Medical Imaging 37 (1) (2018) 20–34.

[19] A. Mehranian, M. A. Belzunce, C. J. McGinnity, A. Bustin, C. Prieto, A. Hammers, A. J. Reader, Magnetic resonance in medicine 81 (3) (2019) 2120–2134.

[20] T. Merlin, S. Stute, D. Benoit, J. Bert, T. Carlier, C. Comtat, M. Filipovic, F. Lamare, D. Visvikis, Physics in Medicine & Biology 63 (18) (2018) 185005.

[21] T. Kösters, K. P. Schäfers, F. Wübbeling, in: 2011 IEEE Nuclear Science Symposium Conference Record, 2011, pp. 4365–4368.

[22] S. Pedemonte, K. Van Leemput, N. Fuin, occiput.io.
http://tomographylab.scienceontheweb.net

[23] P. J. Markiewicz, M. J. Ehrhardt, K. Erlandsson, P. J. Noonan, A. Barnes, J. M. Schott, D. Atkinson, S. R. Arridge, B. F. Hutton, S. Ourselin, Neuroinformatics 16 (1) (2018) 95–115.

[24] M. Uecker, J. Tamir, F. Ong, C. Holme, M. Lustig, BART: version 0.4.01 (Jun. 2017).
https://doi.org/10.5281/zenodo.817472

[25] G. Delso, S. Fürst, B. Jakoby, R. Ladebeck, C. Ganter, S. G. Nekolla, M. Schwaiger, S. I. Ziegler, Journal of Nuclear Medicine 52 (12) (2011) 1914–1922.

[26] A. M. Grant, T. W. Deller, M. M. Khalighi, S. H. Maramraju, G. Delso, C. S. Levin, Medical Physics 43 (5) (2016) 2334–2343.

[27] C. S. Levin, S. H. Maramraju, M. M. Khalighi, T. W. Deller, G. Delso, F. Jansen, IEEE Transactions on Medical Imaging 35 (8) (2016) 1907–1914.

[28] Operator Discretization Library.
https://odlgroup.github.io/odl/index.html

[29] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, S. Ourselin, Computer Methods and Programs in Biomedicine 98 (3) (2010) 278–284.

[30] M. Modat, D. M. Cash, P. Daga, G. P. Winston, J. S. Duncan, S. Ourselin, Journal of Medical Imaging (Bellingham, Wash.) 1 (2) (2014) 024003.

[31] Simplified Wrapper and Interface Generator.
http://www.swig.org/index.php

[32] NIfTI-2 Data Format – Neuroimaging Informatics Technology Initiative. https://nifti.nimh.nih.gov/nifti-2

[33] H. Hudson, R. Larkin, IEEE Transactions on Medical Imaging 13 (4) (1994) 601–609.

[34] J. E. Bowsher, V. E. Johnson, T. G. Turkington, R. J. Jaszczak, C. E. Floyd, R. E. Coleman, Medical Imaging, IEEE Transactions on 15 (5) (1996) 673–686.

[35] M. J. Ehrhardt, P. Markiewicz, M. Liljeroth, A. Barnes, V. Kolehmainen, J. S. Duncan, L. Pizarro, D. Atkinson, B. F. Hutton, S. Ourselin, K. Thielemans, S. R. Arridge, IEEE Transactions on Medical Imaging 35 (9) (2016) 2189–2199.

[36] D. Hogg, K. Thielemans, S. Mustafovic, T. J. Spinks, in: 2002 IEEE Nuclear Science Symposium Conference Record, Vol. 3, IEEE, 2002, pp. 1519–1523, event-place: Norfolk, VA, USA.

[37] P. Green, IEEE Transactions on Medical Imaging 9 (1) (1990) 84–93.

[38] C. C. Watson, D. Newport, M. E. Casey, in: P. Grangeat, J. L. Amans (Eds.), Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Kluwer Academic, Amsterdam, Netherlands, 1996, pp. 255–268.

[39] I. Polycarpou, K. Thielemans, R. Manjeshwar, P. Aguiar, P. Marsden, C. Tsoumpas, Annals of Nuclear Medicine 25 (9) (2011) 643–649.

[40] M. A. Griswold, P. M. Jakob, R. M. Heidemann, M. Nittka, V. Jellus, J. Wang, B. Kiefer, A. Haase, Magn Reson Med. 47(6) (2002) 1202 – 1210.

[41] S. J. Inati, J. D. Naegele, N. R. Zwart, V. Roopchansingh, M. J. Lizak, D. C. Hansen, C.-Y. Liu, D. Atkinson, P. Kellman, S. Kozerke, H. Xue, A. E. Campbell-Washburn, T. S. Sørensen, M. S. Hansen, Magn. Reson. Med. 77 (1) (2017) 411–421.

[42] M. A. Griswold, P. M. Jakob, R. M. Heidemann, M. Nittka, V. Jellus, J. Wang, B. Kiefer, A. Haase, Magnetic Resonance in Medicine 47 (6) (2002) 1202–1210.

[43] R. Brown, B. A. Thomas, A. Rashidnasab, K. Erlandsson, E. Ovtchinnikov, E. Pasca, A. J. Reader, J. C. Matthews, C. Tsoumpas, K. Thielemans, in: 2018 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), IEEE, Sydney, Australia, 2018.

[44] K. Thielemans, P. Schleyer, J. Dunn, P. K. Marsden, R. M. Manjeshwar, in: 2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (2013 NSS/MIC), 2013, pp. 1–5.

[45] P. J. Schleyer, J. T. Dunn, S. Reeves, S. Brownings, P. K. Marsden, K. Thielemans, Physics in Medicine and Biology 60 (16) (2015) 6441–6458.

[46] C. Tsoumpas, K. Thielemans, Nuclear Medicine Communications 30 (7) (2009) 490–493.

[47] R. Brown, S. Ellis, E. Pasca, E. Ovtchinnikov, A. Gillman, C. Munoz, J. Bland, A. Mehranian, C. Prieto, A. J. Reader, K. Thielemans, in: Abstracts of PSMR2019, Muenich, Germany, 2019.

[48] A. Beck, M. Teboulle, SIAM Journal on Imaging Sciences 2 (1) (2009) 183–202.

[49] A. R. De Pierro, IEEE Transactions on Medical Imaging 14 (1) (1995) 132–137.

[50] CCPi, Ccpi core imaging library (june 2019). `https://www.ccpi.ac.uk/CIL`

[51] D. Kazantsev, E. Pasca, M. J. Turner, P. J. Withers, SoftwareX 9 (2019) 317 – 323.

[52] L. I. Rudin, S. Osher, E. Fatemi, Physica D: Nonlinear Phenomena 60 (1) (1992) 259 – 268.

[53] A. Beck, M. Teboulle, IEEE Transactions on Image Processing 18 (11) (2009) 2419–2434.

[54] J. E. Bowsher, Hong Yuan, L. W. Hedlund, T. G. Turkington, G. Akabani, A. Badea, W. C. Kurylo, C. T. Wheeler, G. P. Cofer, M. W. Dewhirst, G. A. Johnson, in: IEEE Symposium Conference Record Nuclear Science 2004., Vol. 4, 2004, pp. 2488–2492 Vol. 4.

[55] N. Efthimiou, E. Emond, P. Wadhwa, C. Cawthorne, C. Tsoumpas, K. Thielemans, Physics in Medicine and Biology 64 (3) (2019) 035004.

[56] P. Wadhwa, K. Thielemans, N. Efthimiou, K. Wangerin, D. Timoth, G. Delso, K. Nicholas, D. Daniel, E. Emond, M. Tohme, F. Janssen, R. N. Gunn, D. L. Buckley, W. A. Hallett, C. Tsoumpas, Methods (in-press).

[57] V. Roopchansingh, J. A. Derbyshire, GE to ISMRMRD converter, original-date: 2016-02-16T14:34:23Z (May 2019).
`https://github.com/ismrmrd/ge_to_ismrmrd`

[58] G. Wang, J. Qi, IEEE Transactions on Medical Imaging 34 (1) (2015) 61–71.

[59] J. Bland, M. A. Belzunce, S. Ellis, C. J. McGinnity, A. Hammers, A. J. Reader, IEEE Transactions on Radiation and Plasma Medical Sciences 2 (5) (2018) 470–482.

[60] D. Deidda, N. A. Karakatsanis, P. M. Robson, Y.-J. Tsai, N. Efthimiou, K. Thielemans, Z. A. Fayad, R. G. Aykroyd, C. Tsoumpas, Inverse Problems 35 (4) (2019) 044001.

[61] D. Deidda, N. A. Karakatsanis, P. M. Robson, N. Efthimiou, Z. A. Fayad, R. G. Aykroyd, C. Tsoumpas, IEEE Transactions on Radiation and Plasma Medical Sciences 3 (4) (2019) 400–409.

[62] M. J. Ehrhardt, M. M. Betcke, SIAM Journal on Imaging Sciences 9 (3) (2016) 1084–1106.

[63] B. Bilgic, T. H. Kim, C. Liao, M. K. Manhard, L. L. Wald, J. P. Haldar, K. Setsompop, Magnetic resonance in medicine 80 (2) (2018) 619–632.

[64] A. Mehranian, C. Prieto, R. Neji, C. J. McGinnity, A. Hammers, A. J. Reader, in: ISMRM 2018, Paris, France, 2018.

[65] C. H. McCollough, S. Leng, L. Yu, J. G. Fletcher, Radiology 276 (3) (2015) 637–653.

[66] I. Danad, Z. A. Fayad, M. J. Willemink, J. K. Min, JACC: Cardiovascular Imaging 8 (6) (2015) 710–723.

[67] A. C. Silva, B. G. Morse, A. K. Hara, R. G. Paden, N. Hongo, W. Pavlicek, RadioGraphics 31 (4) (2011) 1031–1046.

[68] B. Chen, Z. Zhang, E. Y. Sidky, D. Xia, X. Pan, Physics in Medicine and Biology 62 (22) (2017) 8763–8793.

[69] G. Wang, J. Zhang, H. Gao, V. Weir, H. Yu, W. Cong, X. Xu, H. Shen, J. Bennett, M. Furth, Y. Wang, M. Vannier, PLoS ONE 7 (6) (2012) e39700.

[70] Y. Xi, J. Zhao, J. R. Bennett, M. R. Stacy, A. J. Sinusas, G. Wang, IEEE Transactions on Biomedical Engineering 63 (6) (2016) 1301–1309.