International Journal on Magnetic Particle Imaging
Vol 5, No 1, Article ID 1907001, 9 Pages

Infinite Science Publishing

Research Article

MPIReco.jl: Julia Package for Image Reconstruction in MPI

Tobias Knopp[a,b,*] · Patryk Szwargulski[a,b] · Florian Griese[a,b] · Mirco Grosser[a,b] · Marija Boberg[a,b] · Martin Möddel[a,b]

[a]Section for Biomedical Imaging, University Medical Center Hamburg-Eppendorf, Hamburg, Germany
[b]Institute for Biomedical Imaging, Hamburg University of Technology, Hamburg, Germany
[*]Corresponding author: t.knopp@uke.de

Received 07 May 2019; Accepted 19 June 2019; Published online 09 July 2019

Abstract

Image reconstruction plays an important role for the tomographic imaging technique magnetic particle imaging (MPI) since the measured raw data cannot be directly interpreted. Instead, one needs to invert the image formation process, which involves the solution of an ill-conditioned linear system of equations. Currently, most MPI researchers have implemented custom reconstruction algorithms that cannot be directly compared since the source code is not openly available. The software package *MPIReco.jl* aims to change this situation by providing a reference implementation for a variety of reconstruction algorithms. With the recently proposed magnetic particle imaging data format and its reference implementation *MPIFiles.jl* we have taken the first steps towards standardised data exchange. With *MPIReco.jl* we complement these initiatives to standardise the reconstruction algorithms and to facilitate reproducible research. We chose to implement the algorithms in the programming language Julia, which provides a high level syntax making the software accessible even for non-professional software developers. On the other hand Julia code has a high run-time performance comparable to low-level C code. In the present paper, we outline some of the design principles of *MPIReco.jl* and give an overview of the software package.

I. Introduction

Magnetic nanoparticles (MNP) are useful tracers for the diagnosis of a variety of diseases. While the particles enhance the contrast in magnetic resonance imaging by affecting the T_2^* relaxation time of the measured tissue, the tomographic imaging method magnetic particle imaging (MPI) measures the particle concentration directly and thus provides a positive contrast. Potential applications of MPI can be found in the field of vascular imaging [1–4], in the field of targeted imaging [5–7], and in the field of neuroimaging [8–11]. When medical instruments are labeled with particles, MPI is also suited for interventional imaging [12–14].

Since its invention in 2001, and its first publication in 2005 [15], numerous improvements and variations of the original idea have been developed in the field of instrumentation, field sequences, and particle synthesis. Still, the basic principle remains: The particles are excited with a set of excitation coils, and the change of the particle magnetization is recorded with inductive receiving coils. The crucial step for obtaining the final image based on the recorded signal is to apply a reconstruction step that inverts the physical process happening during the MPI experiment. Based on Faradays law of induction one can show that the relation between the induced signals and the particle concentration is linear and reconstruction thus involves the solution of a linear system of equa-

tions. Since the linear system is ill-conditioned [16] it requires an appropriate regularization technique to cope with the noise in the measurements. Most researchers apply Tikhonov regularization while recent works [17, 18] also use more sophisticated yet computationally expensive priors. The use of iterative solvers was initially proposed in [19].

One increasingly important topic in the field of image reconstruction is the question how to handle multi-patch data, where a large region of interest is partitioned into small patches sampled sequentially. In [20] and [21] an algorithm for reconstructing multi-patch data in a joint fashion ensuring consistency at patch boundaries has been developed. A time and memory efficient implementation of the algorithm has been proposed in [22]. Another important field that has been established in [23] is the field of multi-contrast reconstruction. Instead of calculating the distribution of a single tracer, multi-contrast reconstruction is capable of determining the distribution of several different tracers. The same technique can also be used to determine environment parameters such as viscosity [24], temperature [25], or particle size [26]. Nevertheless, whether for a multi-patch or a multi-contrast reconstruction, the system matrices and, above all, the linear systems of equations to be solved can grow rapidly and thus drastically prolong the reconstruction time. A method to counter this problem is the use of matrix compression techniques proposed in [27–29].

While there is growing stack of innovative reconstruction approaches, one increasing challenge is to apply and validate the methods under comparable conditions. This is due to the fact that research articles commonly only provide the algorithmic description of the methods, while no runnable code is provided. One exception is the system matrix viewer published in [30]. The software package *MPIReco.jl* aims at changing this situation by providing an open software platform for a variety of MPI reconstruction algorithms. *MPIReco.jl* is scanner agnostic and focuses on system matrix based reconstruction algorithms. It implements a processing chain that was originally developed in [19]. Beside baseline image reconstruction, *MPIReco.jl* also provides algorithms for more sophisticated reconstruction schemes, such as efficient joint multi-patch reconstruction [22], multi-gradient reconstruction [31], and multi-contrast reconstruction [23]. For accelerated image reconstruction *MPIReco.jl* implements matrix-compression techniques developed in [27–29]. The present paper gives an overview of *MPIReco.jl* and discusses its main design principles. Whenever we use programming code in this work, we indicate it by using a `typewriter` font.

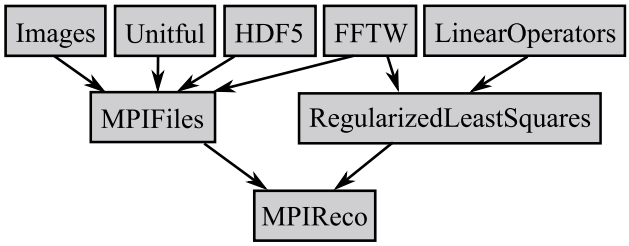

Figure 1: Dependency graph of *MPIReco.jl* including the most important direct and indirect dependencies.

II. Overview

MPIReco.jl is implemented in the programming language Julia [32]. Julia is a high-level multi-purpose programming language that is easy to use and allows to implement algorithms with a high degree of abstraction while still generating efficient machine code. The language fully supports generic programming and dispatches on the type of function arguments (multiple dispatch). This allows to have common function interfaces implemented by different concrete types. Julia is equipped with a powerful package manager that facilitates a proper modularization of independent functionality. The complete documentation and the description of how to install Julia can be found online.[1] *MPIReco.jl* leverages this flexibility by using several existing Julia packages, which can be found in the dependency graph shown in Figure 1. The two main dependencies are *MPIFiles.jl* [33] and *RegularizedLeastSquares.jl*. The former provides a unified access to different MPI raw data files such as the files measured with the MPI scanner from Bruker or vendor-independent magnetic particle imaging data format (MDF) files [34]. *RegularizedLeastSquares.jl* provides different algorithms for solving regularized least squares problems such as the Kaczmarz algorithm [35], which is commonly used in MPI. In addition, more sophisticated solvers like the fused lasso solver [17] are available.

MPIReco.jl is developed within a public Git repository hosted at Github.[2] The project is part of the *MagneticParticleImaging* organization and contains online documentation that can be accessed from the project homepage. Bug reports, feature requests, and comments can be made using an issue tracker. Any commit made to *MPIReco.jl* is tested using continuous integration services. *MPIReco.jl* is supposed to run on any operating system and platform that Julia itself supports. Currently, the test suite runs successfully on Linux, OS X, and Windows.

The software package can be installed from Julia itself by opening the package mode (by entering] within Julia) and then entering the following command:

[1] https://julialang.org/
[2] https://github.com/MagneticParticleImaging/MPIReco.jl

```
(v1.1) pkg> add MPIReco
```

This will install the latest released version of *MPIReco.jl*. If you want to make changes to the package you can checkout the package for development by entering the following command:

```
(v1.1) pkg> dev MPIReco
```

The files then can be edit within the following folder:
`~/.julia/dev/MPIReco/`
In this paper we consider version 0.1.1 of *MPIReco.jl*. The package works on any Julia version greater than 1.0. We recommend to always use the newest stable Julia version, which is version 1.1 at the time this paper has been published.

II.I. Background

The system matrix based reconstruction approach considers the linear system of equations

$$\boldsymbol{S}\boldsymbol{c} = \boldsymbol{u}, \tag{1}$$

where $\boldsymbol{S} \in \mathbb{C}^{M\times N}$ is the complex-valued system matrix, $\boldsymbol{u} \in \mathbb{C}^{M}$ are the Fourier coefficients of the induced signals and $\boldsymbol{c} \in \mathbb{R}^{N}$ is the particle concentration vector. The first dimension of the matrix $\boldsymbol{S}$ does usually not only encode the number of frequencies, but also the number of receive channels and the number of patches. The second dimension encodes the number of image voxels and – in the case of multi-contrast MPI – the number of contrasts (discussed in Section IV).

The linear system is usually not setup completely. Instead one first applies a filter according to specific rules and loads only a subset of the matrix rows. The corresponding reduced system of equations then reads

$$\boldsymbol{S}^{\text{red}}\boldsymbol{c} = \boldsymbol{u}^{\text{red}}, \tag{2}$$

where $\boldsymbol{S}^{\text{red}} \in \mathbb{C}^{\bar{M}\times N}$ and $\boldsymbol{u} \in \mathbb{C}^{\bar{M}}$. For 3D system matrices that can be larger than 30 GB, this data reduction is crucial to avoid exceeding the main memory of the reconstruction computer. After filtering of the matrix rows the linear system (2) is usually solved in a least-squares fashion by calculating

$$\boldsymbol{c}_{\text{Reco}} = \underset{\boldsymbol{c}\in\mathbb{R}^{N},\boldsymbol{c}\geq 0}{\operatorname{argmin}} \|\boldsymbol{S}^{\text{red}}\boldsymbol{c} - \boldsymbol{u}^{\text{red}}\|_2^2 + R(\boldsymbol{c}), \tag{3}$$

where R is a regularization term that penalizes implausible solutions. For instance R can be the L_1 or the L_2 norm. The optimization problem (3) is commonly solved using iterative solvers [19].

MPIReco.jl provides different reconstruction levels ranging from high level, where the user only has to hand over the paths of the measurements to low level, where the user has all parameter freedoms and handles the data as arrays. All of these reconstruction routines are called `reconstruction` and the dispatch is done based on the input types. Here, we exploit Julia's powerful multiple-dispatch mechanism. In the following we discuss each level starting with the high-level reconstruction.

II.II. High-level reconstruction

For the high-level reconstruction one first defines a structured dictionary `recoParams` which has the mandatory parameter `:measPath`. The colon here is a special Julia syntax for defining a symbol, which can be used as a key in a dictionary. An example parameter set could look as follows:

```
recoParams = Dict{Symbol,Any}()
recoParams[:measPath] = fnMeas
recoParams[:SFPath] = fnSF
recoParams[:frames] = 1:100
recoParams[:numAverages] = 10
recoParams[:SNRThresh] = 4.0
recoParams[:lambda] = 0.001
recoParams[:iterations] = 10
```

Here, `fnMeas` and `fnSF` are the paths of the measurement and the system matrix, respectively. Having defined the reconstruction parameters one can pass the `recoParams` to the reconstruction function, which has the following function signature:

```
reconstruction(recoParams::Dict)
```

Instead of putting all parameters into a dictionary one can instead call `reconstruction` like this:

```
reconstruction(fnSF::AbstractString,
               fnMeas::AbstractString;
               kargs...)
```

Here, `kargs` are keyword arguments. The `kargs...` syntax indicates that there can be various keyword arguments. The previous example would for instance translate into the following:

```
reconstruction(fnSF,
               fnMeas;
               frames = 1:100,
               numAverages = 10,
               SNRThresh = 4.0,
               lambda = 0.001,
               iterations = 10)
```

Keyword arguments play an important role in the design of *MPIReco.jl*. The idea is that each reconstruction layer

has a certain set of parameters that it will use while the remaining parameters in `kargs` will be passed to the next layer. This allows to make changes in the low-level code without any necessity to make changes in one of the higher levels.

The advantage of using a dictionary is that one can keep all parameters in a single structure, which can be copied, compared, saved in a file and so on. One common pattern is to perform reconstructions with varying parameters, which can be done by first defining a dictionary with all static parameters and than loop over the parameters that are to be varied. An example where the regularization parameter is changed looks like this:

```
c = Any[]
for lambda in [0.001, 0.01, 0.1]
  recoParams[:lambda] = lambda
  cl = reconstruction(recoParams)
  push!(c, cl)
end
```

Instead of passing the filename as a string one can also pass an `MPIFile`, which is an abstract type describing an MPI data file including all important information about the scans [34]. Different implementations of an `MPIFile` exist in *MPIFiles.jl* such as `BrukerFile` and `MDFFile`. The function signature of the reconstruction function then looks like this:

```
reconstruction(fSF::MPIFile,
               fMeas::MPIFile;
               kargs...)
```

The high-level reconstruction checks if the dataset is a single-patch or a multi-patch dataset and then calls the corresponding specialized functions. Additionally, the function calls `filterFrequencies` from the *MPIFiles.jl* package to select only a subset of all frequency components as described in (2). The most important frequency selection parameters are outlined in Section II.VI. The high-level reconstruction is mainly suitable for users who do not want to develop new methods themselves but want to reconstruct their MPI data with the established tools.

II.III. Middle Level Reconstruction

The next layer of the reconstruction function looks like this:

```
reconstruction(fSF::MPIFile,
               fMeas::MPIFile,
               freq::Vector;
               kargs...)
```

One can see that the frequency index is passed to this function as the third argument, so that the user can easily integrate own frequency selection algorithms and test them in the reconstruction. The function is responsible for loading the system matrix, the measurement data, and potential background data that is subtracted from the measurement. For any frame to be reconstructed, the low-level reconstruction routine is called.

II.IV. low-level reconstruction

Finally, we arrived at the low-level reconstruction routine that has the following signature:

```
reconstruction(S::Any,
               u::Array;
               kargs...)
```

One can see that it requires the system matrix `S` and the measurements `u` to be already loaded. On purpose, the matrix `S` is not restricted in the type. For a regular reconstruction one will basically feed in a `Matrix{ComplexF32}`, although more precisely it will be a `transposed` version of that type if the `"kaczmarz"` algorithm is being used. However, in case that matrix compression is applied `S` will be of type `SparseMatrixCSC`. And for multi-patch reconstruction `S` will be a dedicated type as well. The low-level reconstruction method calls one of the solvers from `RegularizedLeastSquares.jl`. At this level, the user has all freedom and has to make sure that the data is available in the correct format and dimension. This also allows users to reconstruct data from other scanners without having to convert it to a specific file format like the MDF.

II.V. Solver

The most important parameters for choosing the linear solver are outlined in the following table:

Parameter	Description
`solver`	Solver used for reconstruction
`lambda` or λ	Regularization parameter Can be a scalar or a vector in case of multiple regularizers
`shuffleRows`	Shuffle rows in the Kaczmarz algorithm
`enforceReal`	Enforce a real solution
`enforcePositive`	Enforce a positive solution

The solver is chosen based on the `solver` keyword that accepts a String, which in Julia is written using quotation marks. Currently, `solver` can be any of `"kaczmarz"`, `"cgnr"`, `"fusedlasso"`. Depending on the solver,

10.18416/ijmpi.2019.1907001

there can be additional keyword arguments. For instance all iterative algorithms have an argument `iterations` that controls the number of iterations being used. All solvers providing Tikhonov regularization have a parameter `lambda` that controls the regularization parameter. When using a single regularization term, like L_2 regularization, `lambda` can be a scalar `Float64` value. In case of multiple regularizers, `lambda` has to be a vector of floats. The `"fusedlasso"` solver, for instance, implements L_1 and TV regularization. Therefore, `lambda` needs to be of length two. The `"kaczmarz"` solver has an additional boolean parameter `shuffleRows` that allows to use a randomized version of the Kaczmarz algorithm. The parameters `enforceReal` and `enforcePositive` are also boolean and can be used to enforce the solution to be real and positive, respectively.

II.VI. Frequency Selection Parameters

The most important parameters that can be used to control the frequency selection are listed in the following table:

Parameter	Description
`minFreq`	Minimum frequency threshold
`maxFreq`	Maximum frequency threshold
`SNRThresh`	SNR threshold
`recChannels`	Selected receive channels Can be a vector like `[1,2,3]` or a range like `1:2`
`maxMixingOrder`	Maximum mixing order

It is very common to remove frequency components that are below the noise floor, which can be done by choosing `SNRThresh` > 1. The restriction of the receive channel can be important when investigating the influence of the receive channels on the reconstruction result [36] or when using a custom receiver as was done in [3]. The parameters `minFreq` and `maxFreq` can be used to cut off all frequencies below or above the specialised borders as for example to cut off the drive field frequencies. For the Bruker scanner used in e.g. [37] having excitation frequencies between 24.510 kHz and 26.042 kHz, one can remove the excitation frequencies by setting `minFreq` to 27000. Since the analog filter used to damp the excitation frequencies is not sharp, one may in practise use a higher frequency cut-off.

As an alternative to the SNR threshold, frequency selection can also be based in mixing orders [38] by specifying the parameter `maxMixingOrder` putting an upper bound on the mixing order of the frequency components. The mixing order was introduced in [39] and there is a monotonic dependency between mixing factors and SNR [19].

II.VII. General Parameters

Beside the frequency selection parameters, following parameters are important to control the reconstruction process:

Parameter	Description
`frames`	Frames to be reconstructed
`numAverages`	Number of block averages
`spectralLeakage-Correction`	Flag for enabling spectral leakage correction
`emptyMeas`	Empty measurement, has to be an `MPIFile`. Otherwise specify `emptyMeasPath`
`bgFrames`	Frames within the empty measurement that are taken for background estimation
`bgCorrection-Internal`	Flag if internal background frames are used for background subtraction

The most important parameter here is `frames`, which is used to specify the frames that should be reconstructed. When setting for instance `numAverages = 100` and `frames = 1:1000` one will obtain 10 reconstructed images where for each image a block of 100 raw data frames has been averaged prior to reconstruction.

The flag `spectralLeakageCorrection` controls whether a spectral leakage correction is applied when the data is loaded. For this purpose three neighbouring frames are windowed with a Hann window and the average of the three frames is taken.

Background subtraction can be done by specifying `emptyMeas`, which can either be another file that has been measured with an empty bore or the same file as is used for reconstruction. The later is useful in case that the measurement is initiated with an empty scanner bore. The frames taken for background subtraction are specified by `bgFrames`. All frames are averaged before they are subtracted. Instead of using a dedicated empty measurement file, one can also use internal empty measurements, if the file to be reconstructed contains background frames. For instance, the MDF data within the Open MPI Data[3] contain empty measurements within the files itself.

II.VIII. Reconstruction Result

The return value `c` of `reconstruction` except for the low-level variant is not a simple `Array` but a high level wrapper around the data. In particular two wrappers are applied. The first is an `AxisArray`, which gives the dimensions of `c` a name, a range, and units. The second is an `ImageMeta`, which is basically just the regular array

[3] https://magneticparticleimaging.github.io/OpenMPIData.jl/latest/

plus a dictionary that can hold arbitrary metadata. In order to get rid of the wrappers one can call the following:

```
c.data.data
```

The first `.data` will take of the `ImageMeta` wrapper, while the second `.data` will take of the `AxisArray` wrapper.

But there is reason to work with the `ImageMeta` type. For instance, the reconstruction result can be stored by calling the following function:

```
saveRecoData("reco.mdf", c)
```

If we inspect the stored data using an HDF5 viewer, we can see that all acquisition parameters are stored correctly and that even the reconstruction parameters are stored. This was only possible since this data was present in the `ImageMeta` wrapper. If we want to load the stored image we can call the following function:

```
cmdf = loadRecoData("reco.mdf")
```

The object `cmdf` is of the same type and content as the originally reconstructed object `c`.

We note that the storage of reconstruction data into an MDF has the advantage that it is lossless. The disadvantage is that common image viewers such as *ImageJ* will not be capable of loading the data. For this purpose one can use the *NIfTI.jl* or the *DICOM.jl* package both of which are under development.

The reconstructed image `c` has always five dimensions. Keeping the number of dimensions static has the advantage that the user knows exactly what to expect from the `reconstruction` method, which is the key for writing type-stable code. The first dimension of `c` is the channel dimension. The dimension will be a singleton dimension unless a multi-contrast reconstruction is performed. More details on that will follow in Section IV. The 2nd to 4th dimension of `c` are the three spatial dimensions N_x, N_y, N_z of which some can again be singleton dimensions in case that 1D or 2D reconstructions are performed. The 5th dimension of `c` is the frame dimension. It encodes the number of (averaged) frames being reconstructed.

III. Multi-Patch

For multi-patch reconstruction a generalized version of the method proposed in [22] is integrated into *MPIReco.jl*. The algorithm integrates the data from different patches into a single linear system of equations and then performs a joint reconstruction [20]. In the sequel, we consider a multi-patch experiment with L patches.

In principle, *MPIReco.jl* will automatically detect that the provided measurement file `fMeas` contains multiple patches and will then invoke the multi-patch reconstruction. In case that the patches are measured in separate measurements one can create a multi-patch file using the following constructor:

```
fMultiPatch = MultiMPIFile([fnPatch1, ...,
                            fnPatchL])
```

Here, `fnPatch1, ..., fnPatchL` are the filenames of the single-patch measurements.

If only a single system matrix is passed to the reconstruction, *MPIReco.jl* will assume shift-invariance and use the provided system matrix for all patches. Due to field imperfections, it can be advantageous to use a dedicated system matrix per patch. This can be done by passing several system matrices to the reconstruction:

```
fSF = MultiMPIFile([fnSFPatch1, ...,
                    fnSFPatchL])

c = reconstruction(fSF, fMultiPatch;
                   kargs...)
```

Here, `fnSFPatch1, ..., fnSFPatchL` are the filenames of the system matrices. It is also possible to use less than L but more than one system matrix. In that case, one has to pass an index vector of length L to the `reconstruction` function as the keyword argument `mapping`. The vector specifies, which system matrix within the `MultiMPIFile` is to be used for a certain patch. As an example where two system matrices should be used to reconstruct four patches, the following mapping could be used:

```
mapping = [1,1,2,2]
```

With this, patches one and two are reconstructed with the first system matrix and patches three and four with the second system matrix.

IV. Multi-Contrast

MPIReco.jl also implements algorithms that are commonly known as multi-contrast or multi-colored reconstruction. Let D be the number of colors or channels. Then, the multi-contrast reconstruction solves the linear system of equations

$$\begin{pmatrix} \boldsymbol{S}_1 & \cdots & \boldsymbol{S}_D \end{pmatrix} \begin{pmatrix} \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_D \end{pmatrix} = \boldsymbol{u}, \tag{4}$$

10.18416/ijmpi.2019.1907001

where $\boldsymbol{c}_d$, $d = 1,\dots,D$ are the individual channels being reconstructed and $\boldsymbol{S}_d$, $d = 1,\dots,D$ are the system matrices required for reconstruction. For instance, in the initial publication by Rahmer et al. [23], D was 3 and the three system matrices were measured with different tracer materials.

To perform a multi-contrast reconstruction using *MPIReco.jl* one needs the filenames of the different system matrices `fnS1, ..., fnSD`. Then, one can create a multi-contrast file handle by calling the following constructor:

```
fSF = MultiContrastFile([fnS1, ..., fnSD])
```

With that one can then call the regular reconstruction method:

```
c = reconstruction(fSF, fMeas; kargs...)
```

MPIReco.jl will automatically load the system matrices in a way that they are combined into the composed matrix outlined in (4). For the frequency selection an intersection is performed ensuring that the same matrix rows are available for each system matrix.

We note that `reconstruction` will return an array that encodes the color channel in its first dimension. If `c` is the return value of `reconstruction`, then the d-th channel can be accessed by `c[d,:,:,:,:]`.

V. Matrix-Compression

The reconstruction can be accelerated by applying matrix compression. To this end, the system matrix $\boldsymbol{S}$ is transformed into a different domain by applying a basis transformation on the rows of the system matrix. In *MPIReco.jl*, matrix compression can be enabled by specifying `sparseTrafo`, which can be `"DCT-IV"` or `"FFT"`.

The transformations can be restricted to the drive-field field-of-view by setting `useDFFoV = true`. The compression factor that controls how many coefficients are dropped after application of the transformation is controlled by the parameter `redFactor`. For instance a reduction factor of `redFactor = 0.01` will drop 99 % of the data.

VI. Discussion

This paper gave an overview on the functionality and design principles of the Julia package *MPIReco.jl*. By providing access to the individual layers of the reconstruction chain from low level until high level, the user can change individual parts of the reconstruction chain, without the need to change the code of *MPIReco.jl*. For instance one might implement a custom frequency selection algorithm and then call the corresponding reconstruction layer that gets a frequency index as argument. This shows that *MPIReco.jl* targets both the user that just wants to perform standard reconstruction by employing the high-level interface and the MPI reconstruction expert that might want to make changes in the lower levels of the reconstruction stack.

MPIReco.jl is an active open source project and will evolve from the state that is described in the current paper. But the function signatures and parameter names listed in this paper are not expected to change anymore. We have omitted several more experimental parameters that affect functionality that is still in flux. Nevertheless, further changes can be retrieved and tracked on the project website.[4]

MPIReco.jl is also used by the Open MPI Data initiative.[5] The latter was created to provide experimental MPI data to other researchers who do not have direct access to an MPI scanner. Together with *MPIReco.jl*, the Open MPI Data initiative enables comparison against state of the art algorithms on standardized datasets, when researchers develop new reconstruction algorithms. By integrating new algorithms into *MPIReco.jl*, the reproducibility of studies can be significantly increased.

While currently *MPIReco.jl* is restricted to the reconstruction based on a system matrix, the framework could in-principle be enhanced to also implement x-space reconstruction [40]. This will require a new parameter responsible for switching between a system matrix based reconstruction and an x-space reconstruction.

VII. Conclusion

In conclusion, *MPIReco.jl* is a versatile software package for the reconstruction of magnetic particle imaging data. It focuses on system matrix based reconstruction, can handle single-patch, multi-patch, and multi-contrast data, and implements matrix compression techniques for accelerated reconstruction. *MPIReco.jl* is implemented in Julia and optimized in runtime performance. It makes state-of-the-art reconstruction algorithms accessible for other researchers and complements the Open MPI data initiative, which provides users without an MPI scanner experimental measurement data that can be used for validation purposes.

[4] https://github.com/MagneticParticleImaging/MPIReco.jl
[5] https://github.com/MagneticParticleImaging/OpenMPIData.jl

10.18416/ijmpi.2019.1907001

References

[1] J. Weizenecker, J. Borgert, and B. Gleich. A simulation study on the resolution and sensitivity of magnetic particle imaging. *Physics in Medicine and Biology*, 52(21):6363–6374, 2007, doi:10.1088/0031-9155/52/21/001.

[2] P. Vogel, M. A. Rückert, P. Klauer, W. H. Kullmann, P. M. Jakob, and V. C. Behr. First in vivo traveling wave magnetic particle imaging of a beating mouse heart. *Physics in Medicine and Biology*, 61(18):6620–6634, 2016, doi:10.1088/0031-9155/61/18/6620.

[3] M. Graeser, T. Knopp, P. Szwargulski, T. Friedrich, A. von Gladiss, M. Kaul, K. M. Krishnan, H. Ittrich, G. Adam, and T. M. Buzug. Towards Picogram Detection of Superparamagnetic Iron-Oxide Particles Using a Gradiometric Receive Coil. *Scientific Reports*, 7(1):6872, 2017, doi:10.1038/s41598-017-06992-5.

[4] S. Vaalma, J. Rahmer, N. Panagiotopoulos, R. L. Duschka, J. Borgert, J. Barkhausen, F. M. Vogt, and J. Haegele. Magnetic Particle Imaging (MPI): Experimental Quantification of Vascular Stenosis Using Stationary Stenosis Phantoms. *PLOS ONE*, 12(1):e0168902B. Xu, Ed., 2017, doi:10.1371/journal.pone.0168902.

[5] J. W. M. Bulte, P. Walczak, M. Janowski, K. M. Krishnan, H. Arami, A. Halkola, B. Gleich, and J. Rahmer. Quantitative "Hot-Spot" Imaging of Transplanted Stem Cells Using Superparamagnetic Tracers and Magnetic Particle Imaging. *Tomography*, 1(2):91–97, 2015, doi:10.18383/j.tom.2015.00172.

[6] J. Dieckhoff, M. G. Kaul, T. Mummert, C. Jung, J. Salamon, G. Adam, T. Knopp, D. Schwinge, and H. Ittrich. Magnetic Particle Imaging of liver tumors in small animal models. *International Journal on Magnetic Particle Imaging*, 3(2), 2017, doi:10.18416/IJMPI.2017.1707003.

[7] E. Y. Yu, P. Chandrasekharan, R. Berzon, Z. W. Tay, X. Y. Zhou, A. P. Khandhar, R. M. Ferguson, S. J. Kemp, B. Zheng, P. W. Goodwill, M. F. Wendland, K. M. Krishnan, S. Behr, J. Carter, and S. M. Conolly. Magnetic Particle Imaging for Highly Sensitive, Quantitative, and Safe in Vivo Gut Bleed Detection in a Murine Model. *ACS Nano*, 11(12):12067–12076, 2017, doi:10.1021/acsnano.7b04844.

[8] E. E. Mason, C. Z. Cooley, S. F. Cauley, M. A. Griswold, S. M. Conolly, and L. L. Wald. Design analysis of an MPI human functional brain scanner. *International Journal on Magnetic Particle Imaging*, 3(1), 2017, doi:10.18416/IJMPI.2017.1703008.

[9] C. Z. Cooley, J. B. Mandeville, E. E. Mason, E. T. Mandeville, and L. L. Wald. Rodent Cerebral Blood Volume (CBV) changes during hypercapnia observed using Magnetic Particle Imaging (MPI) detection. *NeuroImage*, 178:713–720, 2018, doi:10.1016/j.neuroimage.2018.05.004.

[10] L. Wu, Y. Zhang, G. Steinberg, H. Qu, S. Huang, M. Cheng, T. Bliss, F. Du, J. Rao, G. Song, L. Pisani, T. Doyle, S. Conolly, K. Krishnan, G. Grant, and M. Wintermark. A Review of Magnetic Particle Imaging and Perspectives on Neuroimaging. *American Journal of Neuroradiology*, 40(2):206–212, 2019, doi:10.3174/ajnr.A5896.

[11] P. Ludewig, N. Gdaniec, J. Sedlacik, N. D. Forkert, P. Szwargulski, M. Graeser, G. Adam, M. G. Kaul, K. M. Krishnan, R. M. Ferguson, A. P. Khandhar, P. Walczak, J. Fiehler, G. Thomalla, C. Gerloff, T. Knopp, and T. Magnus. Magnetic Particle Imaging for Real-Time Perfusion Imaging in Acute Stroke. *ACS Nano*, 11(10):10480–10488, 2017, doi:10.1021/acsnano.7b05784.

[12] J. Haegele, N. Panagiotopoulos, S. Cremers, J. Rahmer, J. Franke, R. L. Duschka, S. Vaalma, M. Heidenreich, J. Borgert, P. Borm, J. Barkhausen, and F. M. Vogt. Magnetic Particle Imaging: A Resovist Based Marking Technology for Guide Wires and Catheters for Vascular Interventions. *IEEE Transactions on Medical Imaging*, 35(10):2312–2318, 2016, doi:10.1109/TMI.2016.2559538.

[13] J. Salamon, M. Hofmann, C. Jung, M. G. Kaul, F. Werner, K. Them, R. Reimer, P. Nielsen, A. vom Scheidt, G. Adam, T. Knopp, and H. Ittrich. Magnetic Particle / Magnetic Resonance Imaging: In-Vitro MPI-Guided Real Time Catheter Tracking and 4D Angioplasty Using a Road Map and Blood Pool Tracer Approach. *PLOS ONE*, 11(6):e0156899M. Yamamoto, Ed., 2016, doi:10.1371/journal.pone.0156899.

[14] S. Herz, P. Vogel, P. Dietrich, T. Kampf, M. A. Rückert, R. Kickuth, V. C. Behr, and T. A. Bley. Magnetic Particle Imaging Guided Real-Time Percutaneous Transluminal Angioplasty in a Phantom Model. *CardioVascular and Interventional Radiology*, 41(7):1100–1105, 2018, doi:10.1007/s00270-018-1955-7.

[15] B. Gleich and J. Weizenecker. Tomographic imaging using the nonlinear response of magnetic particles. *Nature*, 435(7046):1214–1217, 2005, doi:10.1038/nature03808.

[16] T. Knopp, S. Biederer, T. Sattel, and T. M. Buzug, Singular value analysis for Magnetic Particle Imaging, in *2008 IEEE Nuclear Science Symposium Conference Record*, 4525–4529, IEEE, 2008. doi:10.1109/NSSMIC.2008.4774296.

[17] M. Storath, C. Brandt, M. Hofmann, T. Knopp, J. Salamon, A. Weber, and A. Weinmann. Edge Preserving and Noise Reducing Reconstruction for Magnetic Particle Imaging. *IEEE Transactions on Medical Imaging*, 36(1):74–85, 2017, doi:10.1109/TMI.2016.2593954.

[18] C. Bathke, T. Kluth, C. Brandt, and P. Maaß. Improved image reconstruction in magnetic particle imaging using structural a priori information. *International Journal on Magnetic Particle Imaging*, 3(1), 2017, doi:10.18416/IJMPI.2017.1703015.

[19] T. Knopp, J. Rahmer, T. F. Sattel, S. Biederer, J. Weizenecker, B. Gleich, J. Borgert, and T. M. Buzug. Weighted iterative reconstruction for magnetic particle imaging. *Physics in Medicine and Biology*, 55(6):1577–1589, 2010, doi:10.1088/0031-9155/55/6/003.

[20] T. Knopp, K. Them, M. Kaul, and N. Gdaniec. Joint reconstruction of non-overlapping magnetic particle imaging focus-field data. *Physics in Medicine and Biology*, 60(8):L15–L21, 2015, doi:10.1088/0031-9155/60/8/L15.

[21] M. Ahlborg, C. Kaethner, T. Knopp, P. Szwargulski, and T. M. Buzug. Using data redundancy gained by patch overlaps to reduce truncation artifacts in magnetic particle imaging. *Physics in Medicine and Biology*, 61(12):4583–4598, 2016, doi:10.1088/0031-9155/61/12/4583.

[22] P. Szwargulski, M. Möddel, N. Gdaniec, and T. Knopp. Efficient Joint Image Reconstruction of Multi-Patch Data Reusing a Single System Matrix in Magnetic Particle Imaging. *IEEE Transactions on Medical Imaging*, 38(4):932–944, 2019, doi:10.1109/TMI.2018.2875829.

[23] J. Rahmer, A. Halkola, B. Gleich, I. Schmale, and J. Borgert. First experimental evidence of the feasibility of multi-color magnetic particle imaging. *Physics in Medicine and Biology*, 60(5):1775–91, 2015, doi:10.1088/0031-9155/60/5/1775.

[24] M. Möddel, C. Meins, J. Dieckhoff, and T. Knopp. Viscosity quantification using multi-contrast magnetic particle imaging. *New Journal of Physics*, 20(8):083001, 2018, doi:10.1088/1367-2630/aad44b.

[25] C. Stehning, B. Gleich, and J. Rahmer. Simultaneous magnetic particle imaging (MPI) and temperature mapping using multi-color MPI. *International Journal on Magnetic Particle Imaging*, 2(2), 2016, doi:10.18416/IJMPI.2016.1612001.

[26] C. Shasha, E. Teeman, K. M. Krishnan, P. Szwargulski, T. Knopp, and M. Möddel. Discriminating nanoparticle core size using multi-contrast MPI. *Physics in Medicine & Biology*, 64(7):074001, 2019, doi:10.1088/1361-6560/ab0fc9.

[27] J. Lampe, C. Bassoy, J. Rahmer, J. Weizenecker, H. Voss, B. Gleich, and J. Borgert. Fast reconstruction in magnetic particle imaging. *Physics in Medicine and Biology*, 57(4):1113–1134, 2012, doi:10.1088/0031-9155/57/4/1113.

[28] T. Knopp and A. Weber. Local System Matrix Compression for Efficient Reconstruction in Magnetic Particle Imaging. *Advances in Mathematical Physics*, 2015:1–7, 2015, doi:10.1155/2015/472818.

[29] L. Schmiester, M. Möddel, W. Erb, and T. Knopp. Direct Image Reconstruction of Lissajous-Type Magnetic Particle Imaging Data Using Chebyshev-Based Matrix Compression. *IEEE Transactions on Computational Imaging*, 3(4):671–681, 2017, doi:10.1109/TCI.2017.2706058.

[30] U. Heinen, A. Weber, J. Franke, H. Lehr, and O. Kosch. A versatile MPI System Function Viewer. *International Journal on Magnetic Particle Imaging*, 3(2), 2017, doi:10.18416/IJMPI.2017.1706006.

[31] N. Gdaniec, P. Szwargulski, and T. Knopp. Fast multiresolution data acquisition for magnetic particle imaging using adaptive feature detection. *Medical Physics*, 44(12):6456–6460, 2017, doi:10.1002/mp.12628.

[32] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, 2017, doi:10.1137/141000671.

[33] T. Knopp, M. Möddel, F. Griese, F. Werner, P. Szwargulski, N. Gdaniec, and M. Boberg. MPIFiles.jl: A Julia Package for Magnetic Particle Imaging Files. *Journal of Open Source Software*, 4(38):1331, 2019, doi:10.21105/joss.01331.

[34] T. Knopp, T. Viereck, G. Bringout, M. Ahlborg, J. Rahmer, and M. Hofmann. MDF: Magnetic Particle Imaging Data Format. *ArXiv e-prints*, 2016. arXiv: 1602.06072v1. URL: https://arxiv.org/abs/1602.06072v2.

[35] S. Kaczmarz. Angenäherte Auflösung von Systemen linearer Gleichungen. *Bulletin International de l'Académie Polonaise des Sciences et des Lettres*, (35)pp. 355–357, 1937.

[36] P. Szwargulski and T. Knopp. Influence of the Receive Channel Number on the Spatial Resolution in Magnetic Particle Imaging. *International Journal on Magnetic Particle Imaging*, 3(1), 2017, doi:10.18416/IJMPI.2017.1703014.

[37] T. Knopp and M. Hofmann. Online reconstruction of 3D magnetic particle imaging data. *Physics in Medicine and Biology*, 61(11):N257–N267, 2016, doi:10.1088/0031-9155/61/11/N257.

[38] P. Szwargulski, J. Rahmer, M. Ahlborg, C. Kaethner, and T. M. Buzug. Experimental evaluation of different weighting schemes in magnetic particle imaging reconstruction. *Current Directions in Biomedical Engineering*, 1(1):206–209, 2015, doi:10.1515/cdbme-2015-0052.

[39] J. Rahmer, J. Weizenecker, B. Gleich, and J. Borgert. Analysis of a 3-D System Function Measured for Magnetic Particle Imaging. *IEEE Transactions on Medical Imaging*, 31(6):1289–1299, 2012, doi:10.1109/TMI.2012.2188639.

[40] J. J. Konkle, P. W. Goodwill, D. W. Hensley, R. D. Orendorff, M. Lustig, and S. M. Conolly. A Convex Formulation for Magnetic Particle Imaging X-Space Reconstruction. *PLOS ONE*, 10(10):e0140137J. Najbauer, Ed., 2015, doi:10.1371/journal.pone.0140137.

