Stefan Kroboth

Fast Regularized Reconstruction for PatLoc MR Imaging using Total Generalized Variation and Graphics Cards

Master's Thesis

Graz University of Technology

Institute of Medical Engineering
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Rudolf Stollberger

Supervisors:
Dipl-Ing. Dr.techn. Florian Knoll
Univ.-Prof. Dr. Kristian Bredies

Graz, December 2012

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

PatLoc MR Imaging uses two nonlinear, nonbijective encoding fields in addition to the conventional three linear gradients for image encoding. This leads to new ways to perform image encoding. However, iterative reconstruction of PatLoc data is a computationally challenging task due to the fact that Fourier encoding does not apply anymore. This work aims at implementing a GPU-accelerated reconstruction framework for PatLoc MR imaging based on two discretization schemes of the forward model. Further, TGV regularization is performed to improve image quality. To improve convergence, a new method for numerically solving the TGV method is proposed, called TGV-CG. The reconstruction is evaluated on in-vivo and phantom data acquired with the PatLoc hardware. It is shown that GPU-acceleration leads to significantly inproved performance which renders the investigated methods practical. In addition, TGV improves image quality even for undersampled data. TGV-CG leads to faster convergence in some cases which further decreases reconstruction time.

Keywords: Nonlinear Encoding, PatLoc, Image Reconstruction, Total Generalized Variation, GPU

Kurzfassung

Die PatLoc MR Bildgebung nutzt zusätzlich zu den konventionellen linearen Gradienten zwei nichtlineare, nichtbijective Gradientenfelder für die Bildkodierung. Dieses Konzept ermöglicht neue Wege in der Bildkodierung. Jedoch bringt es aufgrund der höheren Komplexität auch längere Rekonstruktionszeiten mit sich. Diese Arbeit hat das Ziel eine GPU beschleunigte Rekonstruktion basierend auf zwei Diskretisierungsmodellen des Vorwärtsmodells zu implementieren. Weiters wurde TGV Regularisierung implementiert um die Bildqualität zu verbessern. Da TGV in manchen Fällen sehr langsam konvergiert wird weiters ein neues Konzept für die numerische Lösung des Problems mit dem Namen TGV-CG vorgeschlagen. Die Rekonstruktion wird an In-Vivo- und Phantom-Daten, die mit der PatLoc Hardware aufgenommen wurde, evaluiert. Es zeigt sich dass die GPU beschleunigte Implementierung die Bildrekonstruktion stark beschleunigt und somit die Methode brauchbar macht. Mit der TGV Regularisierung wird die Bildqualität selbst für unterabgetastete Daten erhöht. In manchen Fällen verbessert TGV-CG die Konvergenzgeschwindigkeit sehr stark und verringert somit die Rekonstruktionszeit für regularisierte Rekonstruktion.

Keywords: Nichtlineare Bildkodierung, PatLoc, Bildrekonstruktion, Total Generalized Variation, GPU

Thanks

First of all I'd like to thank my parents Elisabeth and Erich for their continuing support, both personally and financially. Certainly my girlfriend Conny deserves thanks for making the tedious work much more bearable.

Many thanks to my supervisor Florian Knoll for introducing me to the world of science and for giving me opportunities to set foot in the scientific community. Also for his patient and kind manner which helped a lot to motivate myself while working on this project.

My second supervisor Kristian Bredies also deserves many thanks because of his – although being a genious mathematician – capability of talking to non-mathematicians without causing total confusion and self-doubt.

Thanks to the Freiburg-bunch, a group of very kind, helpful, fun and interesting people who are responsible for turning a two weeks stay in Freibug into a true experience that I wouldn't want to miss. Thanks Gerrit Schultz, Dan Gallichan, Sebastian Littin and his wife Steffi, Chris Cocosco, Maxim Zaitsev, Frederik Testud, Hans Weber, Anna Welz, Stathis Hadjidemetriou and many more.

And most importantly I'd like to thank my mind for staying at least partially sane.

Contents

List of Figures

1. Introduction

In the history of magnetic resonance (MR) imaging, the focus in image encoding has been on the use of linear gradient fields. Linear gradient fields exhibit several positive properties, like constant image resolution and field of view (FOV) across the entire image. However, sufficiantly linear gradient fields are hard to obtain. Further, the switching rate of the gradient fields is limited due to physiological limitations like peripheral nerve stimulation (PNS). This puts a physiological limit on fast imaging techniques that require high switching rates of strong gradient fields. Also, there are cases where constant image resolution is not a necessary property, for instance when only certain areas of the object need to be resolved.

To relax the constraint of having linear gradient fields, concepts like O-Space imaging [1] and PatLoc imaging [2] have been introduced recently. These methods use nonlinear gradient fields in addition to or as replacement of the linear gradients. In PatLoc imaging, which this work is based on, two nonlinear, nonbijective gradient fields in the form of hyperbolic paraboloids (Fig. 1.1) are used in conjunction with the standard linear gradients. Due to the nonlinearity of the fields, image resolution becomes a local property with having low resolution in areas of low gradients like the saddle points of the fields and high resolution towards the border of the image. The nonbijectiveness of the fields leads to ambiguities in the encoding which are resolved with use of coil sensitivities similar to SENSE [4].

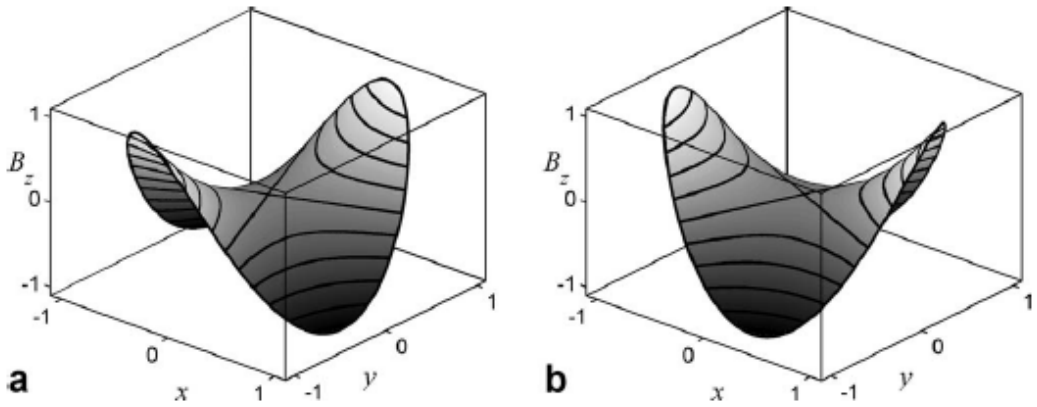


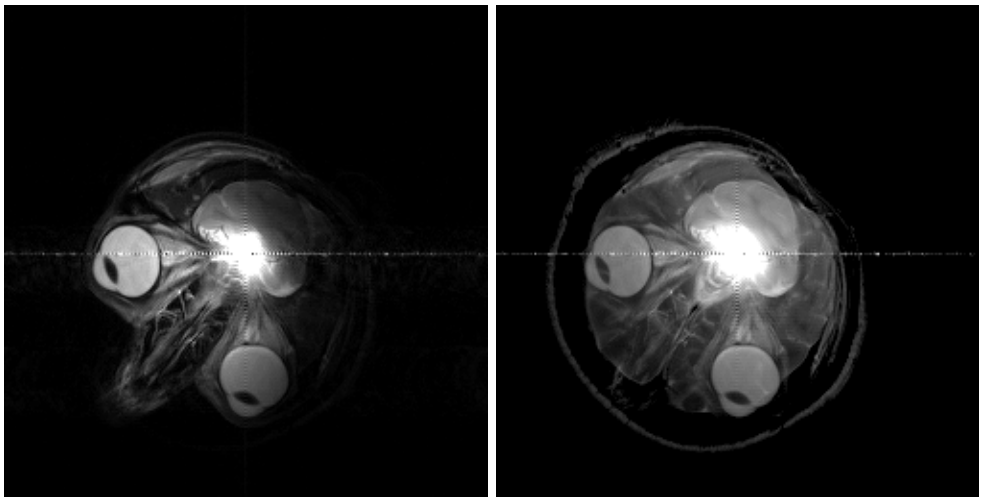
Figure 1.1.: Both curvilinear encoding fields SEM_a and SEM_b which are a close approximation to an ideal hyperbolic paraboloid, and are rotated 45° to each other. This image is taken from [3] with kind permission from the authors.

The PatLoc approach aims at the reduction of the maximal dB/dt by making the gradient fields more local. This allows higher switching rates of the gradient fields without peripheral nerve stimulation, therefore reducing scan time. Furthermore the encoding fields can be designed in a way to best fit the geometry of the anatomy.

All these properties have to be considered in the reconstruction, rendering reconstruction techniques used in conventional magnetic resonance imaging with linear gradients, like the *inverse discrete Fourier transform* (IDFT) useless because conventional Fourier encoding does not apply anymore. This is illustrated in Fig. 1.2 where PatLoc data aquired with the two nonlinear PatLoc gradient fields and a Cartesian sampling pattern is reconstructed with an IFFT.

These images show the PatLoc image space, where the image is highly distorted, leading to signal accumulation in the center of the image.

A lot of thought has been put into reconstruction techniques for PatLoc



(a) Reconstruction of one coil

(b) Reconstruction of all coils combined with sum of squares

Figure 1.2.: Images aquired by reconstructing Cartesian PatLoc data with a conventional IFFT. The reader is assured that these images do not represent the subjects anatomy. The evaluation of the diagnostic value of these images is up to the reader. The intensity range has been cropped in order to reduce the dominating effect of the signal accumulation in the center of the images.

imaging [5]. These techniques all suffer from high computational load causing long reconstruction times. Fortunately theses techniques are well suited for parallelization using the power of graphics cards (Section 2.2).

This work focuses on the implementation of fast regularized reconstruction algorithms with use of graphics cards (GPUs). Two operators (Section 2.5) derived from the signal equation of PatLoc imaging (Eq. 2.1) are investigated. These operators are used in (regularized) reconstruction techniques, namely the conjugate gradient (CG) method [6] and Total Generalized Variation (TGV) [7, 8, 9, 10, 11] (Section 2.6.2). Further a new approach to numerically solving the optimization problem defined by the TGV model is introduced in this work: Total Generalized Variation - Conjugate Gradient (TGV-CG) (Section 2.6.4).

2. Methods

2.1. PatLoc Imaging

This section introduces the basics of PatLoc imaging and follows the presentation in [3, 12]. Only an overview of PatLoc can be given here. For more detailed insights into PatLoc imaging, the reader is referred to [5] which also served as an important source for this section.

Schultz et al. [12] have shown that, when neglecting relaxation effects, the signal s from RF receive channel α can be generalized to include encoding fields beyond simple linear gradients:

$$s_\alpha(\boldsymbol{k}) = \int_V m(\boldsymbol{x}) c_\alpha(\boldsymbol{x}) e^{i\boldsymbol{k}^T \boldsymbol{\psi}(x)} d\boldsymbol{x} \tag{2.1}$$

where α represents the coil index, $m(\boldsymbol{x})$ is the magnetization at position $\boldsymbol{x}$, $c_\alpha(\boldsymbol{x})$ is the RF sensitivity of coil α at position $\boldsymbol{x}$ and $\boldsymbol{\psi}(\boldsymbol{x})$ is a multidimensional function representing all the gradient encoding fields. $\boldsymbol{k}$ is the sampling trajectory, describing the net gradient moment of each field.

In the experiments in [3], four fields are used for the encoding (and a z gradient for slice selection), namely two conventional linear x and y gradients and two additional curvilinear fields called SEM_a and SEM_b (Spatial

Encoding Magnetic fields). Although this theory works for arbitrary SEMs, the PatLoc hardware (Section 2.1.3) is capable of creating two hyperbolic paraboloids. SEM_a and SEM_b have the same form, but are rotated by 45° with respect to each other to be orthogonal, as shown in Fig. 1.1.

The samples of $\boldsymbol{k}$ are ordered with respect to time t, therefore the phase term of Eq. 2.1 can be rewritten as:

$$\boldsymbol{k}^T(t)\boldsymbol{\psi}(\boldsymbol{x}) = k_x(t)x + k_y(t)y + k_a(t)\psi_a(\boldsymbol{x}) + k_b(t)\psi_b(\boldsymbol{x}) = \sum_{l=1}^{L} k_l(t)\psi_l(\boldsymbol{x}) \tag{2.2}$$

where $k_x(t)$ and $k_y(t)$ are equivalent to the familiar k-space coordinates in 2D (Fourier encoding) and $k_a(t)$ and $k_b(t)$ are new k-space coordinates describing the amount of phase encoding due to the fields SEM_a and SEM_b at time t. $k_a(t)$ and $k_b(t)$ are defined by the currents running through the windings of the corresponding coils and $\psi_a(\boldsymbol{x})$ and $\psi_b(\boldsymbol{x})$ are appropriately scaled versions of the fields SEM_a and SEM_b.

To summarize, $k_l(t)$ are the trajectories, $\psi_l(\boldsymbol{x})$ describe the geometry of the applied magnetic fields and L is the number of fields.

Mathematically, the field information $\boldsymbol{\psi}(\boldsymbol{x})$ is defined as follows:

$$\boldsymbol{\psi}(\boldsymbol{x}) = \begin{pmatrix} x \\ y \\ x^2 - y^2 \\ 2xy \end{pmatrix} \tag{2.3}$$

Although the current PatLoc hardware is built to drive four gradient coils, there are cases where more SEMs are needed for reconstruction but not for acquiring the data. This is the case when the real trajectory is measured with use of dynamic field cameras as in [13]. It is necessary to model the

real trajectory as a set of base functions which are represented as SEMs and the corresponding net gradient moment in the reconstruction.

2.1.1. Local k-Space

In conventional MR imaging with linear gradients, the image resolution and FOV are constant across the object because the spatial derivative of the accumulated phase is independent of location. This does not apply to curvilinear SEMs, therefore the resolution and FOV are not spatially independent. This fact led to the introduction of the concept of "local k-space" in [3] to describe the regions of enhanced or reduced resolution.

In the case of encoding with linear gradients only, the accumulated phase ϕ at location $\boldsymbol{x}$ and time t can be written as a function of the gradient history:

$$\phi(\boldsymbol{x}, t) = \gamma \int_0^t \boldsymbol{x} \cdot \boldsymbol{G}(t')dt' \tag{2.4}$$

where $\boldsymbol{G}(t)$ is the vector describing the applied linear gradient field at time t. This can lead to an alternative expression for $\boldsymbol{k}(t)$:

$$\boldsymbol{k}(t) = \nabla\phi(\boldsymbol{x}, t) \tag{2.5}$$

This is an alternative way to describe k-space as the local spatial derivative of the accumulated phase at location $\boldsymbol{x}$. When using nonlinear fields, the local spatial derivative of the phase becomes location dependent as it is shown in Eq. 2.5. The local k-vector field can therefore be defined as:

$$\boldsymbol{k}_{\text{loc}}(\boldsymbol{x}, t) = \nabla\phi(\boldsymbol{x}, t) \tag{2.6}$$

2.1.2. Encoding Trajectories

PatLoc MR imaging opens up a myriad of possible encoding trajectories by using up to four encoding fields, two of them being linear, the other two hyperbolic paraboloids (Fig. 1.1). Any combination of these fields can be used for image encoding, with the possibly most interesting and challenging one being the use of all four fields. Gallichan et. al. [3] introduced and evaluated several different encoding schemes. The most promising of the above mentioned publication is 4D-RIO (4-dimensional Radial In/Out), a 4-dimensional trajectory where both the linear and the nonlinear fields follow separate radial trajectories. The advantage of this trajectory is that the saddle point of the nonlinear fields is moved in each encoding step, hence leading to the effect that the center of the image can be resolved as well, which is a known problem when using only the nonlinear fields. However, imaging with multidimensional trajectories is very prone to miscalibrations of the SEMs, exhibiting serious artifacts across the entire image if not calibrated properly. These artifacts due to miscalibrations have been investigated in detail by the author in [14]. Due to these difficulties the results of this work focus on 2-dimensional image encoding with two nonlinear quadratic SEMs. Results for both Cartesian and radial sampling patterns are shown in Section 3.

However, depending on the used forward model, the reconstruction algorithm is not limited to two encoding fields. On the contrary, even if imaging is performed with two SEMs, it might be necessary to reconstruct with a higher number of encoding fields, for instance if measured trajectories are modeled as a set of base functions represented as SEMs and their corresponding net gradient moment (k-vector) [13].

2.1.3. Hardware

PatLoc requires an additional gradient coil capable of producing the two nonlinear fields (Fig. 1.1). The coil is inserted into the bore of the scanner and offers enough space for acquiring in-vivo images of the head as shown in Fig. 2.1.

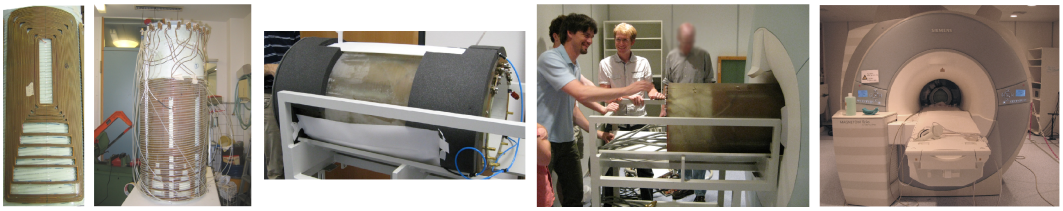


Figure 2.1.: Image shows the design and the insertion of the gradient coil which creates the nonlinear SEMs. The image on the right shows the space available for head imaging. Images are taken from [5] with friendly permission from the author.

2.2. GPGPU Programming

GPGPU (General Purpose Graphics Processing Unit) programming is a relatively new field in scientific computing where graphics cards (GPUs) are used for general purpose computations. Historically, due to the needs of computer games, GPUs are built to process lots of data in parallel on a set of massively parallel processors which can run thousands of threads simultaneously. The design of GPUs follows the SIMD[1] concept (Single Instruction, Multiple Data). This basically means that the same operation is performed on different data in parallel. Therefore data parallelism is essential for maximum performance and this property clearly separates it from CPUs, which

[1]There is some controversy around this term when used to describe GPGPU programming, which probably is the reason that led NVIDIA to calling it SIMT (Single Instruction, Multiple Threads) instead.

are getting more and more parallel themselves but do not exhibit strong limitations on the computations as GPUs do. However, GPU programming opens up new possiblities to solve problems like matrix algebra, image processing, simulations or computer graphics applications very fast.

In most cases in GPGPU programming, the clock speed of the processors is not the limiting part in terms of performance. In contrast to CPUs, memory access is the bottle neck in most applications. Therefore it is essential to limit memory access as much as possible and to follow memory access patterns which can be handled well by the hardware, like fetching data in certain junks.

The programming interface used in this work is NVIDIAs Compute Unified Device Architecture (CUDA) [15], therefore the following information may not apply to graphics cards of manufacturers other than NVIDIA.

For processing on the GPU, the problem needs to be divided into smaller subproblems called blocks. Each block consists of several threads. CUDA assures that all threads of a block are executed on the same streamline processor. The choice on how to divide the problem into blocks and threads depends on the character of the problem and most importantly on the memory needs of the application.

CUDA devices offer several types of memory with different properties that can improve performance significantly if used appropriately. The main memory, called *global memory*, is the biggest but also slowest and can be accessed by all threads in all blocks. Each thread has a *local memory* and *registers* which are very fast to access but also small and can only be accessed by the corresponding thread. In the middle is the *shared memory*, which is accessible by all threads of a block. This is advantageous if passing of data between threads is needed. If data is needed for all threads of a block, it

can be prefetched from the *global memory* to the *shared memory*, leading to more efficient memory access, as the *shared memory* is significantly faster than the *global memory*. Another memory is the *texture memory*, which is commonly used in computer graphics for textures, but can also be used as a fast to access memory for general purpose computations.

2.3. Software and Hardware Environment

The reconstruction algorithms were implemented using Python 2 [16], Scientific Tools for Python (SciPy) [17], Numerical Python (NumPy) [18] and PyCUDA [19]. In contrast to pure CUDA C [15], Python offers easy-to-use libraries for improving the user experience. PyCUDA is a wrapper around CUDA functionality which makes it possible to run CUDA C code from Python. CUDA C code is written as a Python string and compiled and executed by PyCUDA. Therefore it is possible to achieve the same speed as in a pure CUDA C implementation, but due to the fact that CUDA code is a string in Python, there is the possibility to modify or even assemble CUDA code at runtime which can be advantageous if just-in-time (JIT) compilation is needed.

2.3.1. Environment

- Intel Core 2 Duo 6600, 2.4GHz, Dual Core
- 8GB RAM
- Ubuntu 10.10
- Matlab (Mathworks Inc., Natick, USA) R2010b (64bit Version)
- NVIDIA GeForce GTX 480 1536MB, 480 Cores, 384bit Memory interface

- CUDA Toolkit 4.2 [15]
- Python 2.6.6 [16]
- PyCUDA 2012.1 [19]
- scikits.cuda 0.042 [20]
- PyFFT 0.3.6 [21]
- SciPy 0.11.0 [17]
- NumPy 1.6.2 [18]

2.4. Datasets

Fortunately it was possible to acquire a set of datasets measured with the actual PatLoc hardware shown in Section 2.1.3. Both phantom- and in-vivo-measurements were performed with the following settings:

1. Cartesian Spin Echo (SE), TR = 50, TE = 5.5
2. Cartesian Gradient Echo (GRE), TR = 300, TE = 12
3. Radial Spin Echo (SE), TR = 500, TE = 12
4. Radial Gradient Echo (GRE), TR = 50, TE = 5.5
5. Cartesian Turbo Spin Echo (TSE) FIXXME

more?

The slice thickness for each dataset was 5mm. The measurements were performed with linear gradients only and with nonlinear PatLoc gradients only. In the course of this work, only the datasets (3) and (5) of the above list will be investigated, as they exhibited the best tissue contrast in the reconstructed images. Dataset (3) was acquired with a k-space grid of 410 radial spokes with 256 sample points each. The k-space in (5) was acquired with 252 phase encoding steps and 256 samples on each readout line.

2.4.1. Single Shot North West EPI (NW-EPI)

Single shot North-West Echo Planar Imaging (NW-EPI) is a multidimensional trajectory designed to improve the resolution in the top-left region of the image by exploiting the spatially varying resolution characteristic of nonlinear encoding fields [22]. A dynamic field camera [23] is used to measure the encoding fields up to 3rd order (Fig. 2.2).

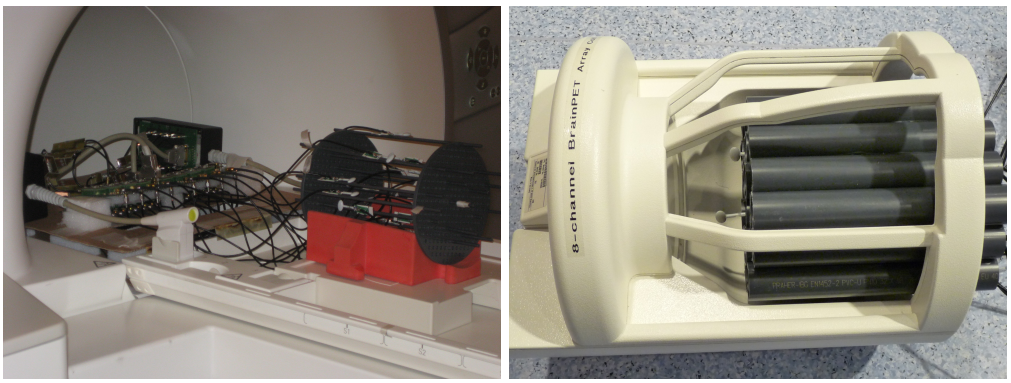


Figure 2.2.: Dynamic field camera used to measure the encoding fields up to third order. The setup is build on MR compatible cardboard and involves 16 ^{1}H fieldprobes approximatively distributed on a spheroids' surface.

The gradient waveforms of the NW-EPI were computed by solving an optimization problem to match the local k-space trajectory in the region of interest to a target trajectory. The target trajectory in this work was an EPI sequence covering twice the k-space extent of that achievable in given time using linear gradients alone. The target trajectory consists of 64 lines with 64 readout points each line for a total time of 41.6ms. The optimization was solved subject to peak gradient constraints and slew rate constraints. The image is reconstructed on a pixel grid that is dense enough (e.g. 192×192) to ensure that also the high local image resolution in the top-left region of the image is fully reflected.

The estimated trajectories are modeled as a set of base functions (represented in the reconstruction as SEMs) and the corresponding net gradient moment ($\boldsymbol{k}$ vector), leading to a total of 16 fields for the reconstruction. Because of the limited image quality due to the low k-space resolution and the fact that the trajectory is prone to Gibbs ringing, regularized reconstruction proved to be an interesting topic to improve the overall quality of the images. This was investigated by the author in [24].

2.5. Forward Operators

In conventional Cartesian MR imaging with linear gradients the reconstruction is performed by using the *fast Fourier transform* (FFT). Reconstructing PatLoc data would require a four-dimensional FFT where only one curved plane is actually needed and the other data being highly redundant, which is too time-consuming for practical use. Although the current hardware is not built to drive more than four fields (and the z gradient for slice selection), there is a case where even more fields are needed for reconstruction. This is the case for reconstruction with trajectories measured by dynamic field cameras as in [13] where 16 base functions used as SEMs are not uncommon. It is not possible to handle the computational load in cases like this. Therefore the reconstruction is interpreted as an inverse problem which is solved by inversion of the forward operator.

The forward operator can be seen as discretization of the signal equation for PatLoc imaging Eq. 2.1. The discretization alway resembles an approximation to the actual signal equation and can be implemented in several ways.

This section introduces the two forward operators used in this work. The first is based on the *encoding matrix* (Section 2.5.1), whereas the second uses the *nonuniform fast Fourier transform* (NUFFT) (Section 2.5.2).

2.5.1. Encoding Matrix

Because of the multidimensional k-vector from Eq. 2.5 which encodes a 2D object, the Fourier transform can no longer be used for the reconstruction of the image. Instead, Eq. 2.1 is discretized and rewritten in matrix form:

$$\boldsymbol{s} = \boldsymbol{E}\boldsymbol{m} \tag{2.7}$$

with $\boldsymbol{E}$ as the discrete forward operator

$$\boldsymbol{E}_{(\alpha,\kappa),\rho} = c_\alpha\left(\boldsymbol{x}_\rho\right) e^{-i\phi(\boldsymbol{x},t)} = c_\alpha\left(\boldsymbol{x}_\rho\right) e^{-i\boldsymbol{k}(t_\kappa)\boldsymbol{\psi}\left(\boldsymbol{x}_\rho\right)} \tag{2.8}$$

where $\boldsymbol{s}$ is the vector of data samples, $\boldsymbol{m}$ is the vector of magnetization values which represents the image of the object and $\boldsymbol{E}$ is the encoding matrix, incorporating both the phase terms resulting from the k-vector and the field information $\boldsymbol{\psi}(\boldsymbol{x})$, and the RF coil sensitivity values represented in Eq. 2.1 by $c_\alpha(\boldsymbol{x})$. Due to the ambiguity of the curvilinear and nonbijective SEMS, a generalized form of SENSE [4, 25] is needed for the reconstruction.

At this point the dimensions are defined as:

N_c Number of coils

N_κ Number of measured data samples

N_ρ Number of samples in the reconstructed image (resolution in x $\times$ resolution in y).

This leads to the following dimensions for the matrix and the vectors mentioned above:

$\boldsymbol{s}$ $\quad N_c N_\kappa$

$\boldsymbol{E}$ $\quad N_c N_\kappa \times N_\rho$

$\boldsymbol{m}$ $\quad N_\rho$

Section 2.6.1 illustrates how the encoding matrix $\boldsymbol{E}$ is constructed on the GPU.

2.5.2. Nonuniform Fast Fourier Transform (NUFFT)

In conventional MR imaging with linear gradients, the *discrete Fourier transform* (DFT) is used for transforming discrete data points measured in the frequency domain to the image domain. Usually a fast implementation of the DFT, called *fast Fourier transform* (FFT) is used. However, the DFT and the FFT require that the sampled data points as well as the points on the image grid lie on grids with equidistant spacing. If these requirements are not met, it is necessary to use a *nonuniform* DFT or its fast equivalent, the *nonuniform fast Fourier transform* (NUFFT). The NUFFT essentially performs an interpolation on a equidistant grid before applying the FFT. Therefore it can be seen as an approximation to the encoding matrix mentioned in Section 2.5.1. The quality of the NUFFT depends on the chosen interpolation method. Fessler et al. introduced a NUFFT framework that is optimal in the min-max sense of minimizing the worst-case approximation error over all signals of unit norm [26].

There are three types of NUFFT: *type 1* NUFFT describes the transformation from a non-equispaced grid to a equispaced grid (known as *gridding*), *type 2*

NUFFT transforms from a equispaced to a non-equispaced grid (known as *inverse gridding*). Transforming from one non-equispaced grid to another non-equispaced grid is known as the *type 3* NUFFT (Fig. 2.3).

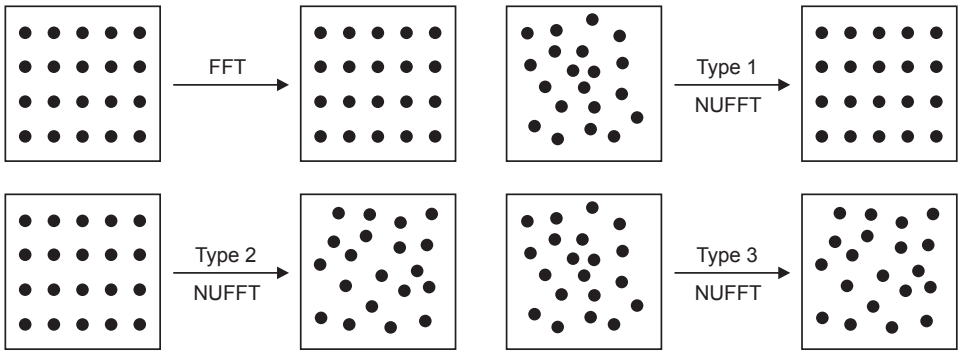


Figure 2.3.: Comparison of FFT and the different NUFFT types. While the FFT transforms from an equispaced grid to another equispaced grid, the *type 1* NUFFT transforms from a non-equispaced grid to a equispaced grid. The *type 2* NUFFT performs the transformation in the other direction. From a non-equispaced grid to another non-equispaced grid is done with a *type 3* NUFFT.

Following the presentation in [11], the *PatLoc* NUFFT 1+2 *Operator* essentially requires a type 3 NUFFT for which at the time of writing no efficient implementation exists. Therefore the type 3 NUFFT is built with use of a type 1 and a type 2 NUFFT with an additional inverse FFT operation, called NUFFT 1+2, as illustrated in Fig. 2.4.

The PatLoc NUFFT 1+2 operator $\mathcal{F}$ can be written channelwise as

$$\mathcal{F}_n = \mathrm{NUFFT}_{\Phi} \circ \mathrm{FT}^{-1} \circ \mathrm{NUFFT}^*_{-\Psi} \circ M_n \tag{2.9}$$

where NUFFT_{Φ} and $\mathrm{NUFFT}^*_{-\Psi}$ are the type 1 and type 2 NUFFT with the corresponding changes of coordinates Φ and $-\Psi$, defined by the trajectory and the field geometries, respectively. FT^{-1} is the conventional inverse Fourier transform and M_n denotes the point wise multiplication with the

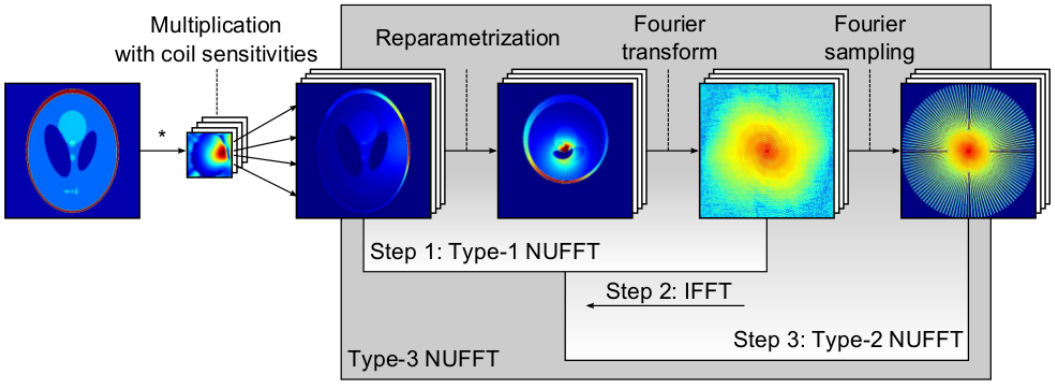


Figure 2.4.: Illustration of the NUFFT 1+2 forward operator $\mathcal{F}$ (Eq. 2.9). First, the image of the object is multiplied with the coil sensitivities of the receive channels, followed by a type 1 NUFFT, an inverse FFT and a type 2 NUFFT. This resembles the implementation of a type 3 NUFFT. Illustration is included from [11] with kind permission of the authors.

coil sensitivity c_n of receive channel n. For a more detailed presentation of the PatLoc NUFFT 1+2 operator please see the above mentioned reference by Knoll et al. [11].

GPU Implementation

For the implementation presented in this work, the NUFFT *Toolbox* by Fessler et al. [26] was ported to PYTHON and all time-consuming parts were ported to GPU code with use of PYCUDA. Essentially the most time-consuming parts can be broken down to a sparse matrix vector multiplication and $N_c + 1$ fast Fourier transforms for each evaluation of the PatLoc NUFFT operator or its adjoint. The sparse matrix-vector multiplication is based on undocumented code available in PYCUDA. The sparse operation uses the coordinate format (each element in a sparse matrix is represented by its value and the indices of its position within the matrix) and it was necessary to adapt it in order to be able to handle complex valued data, as this was

not possible with the code provided by PyCUDA. The code makes heavy use of the texture memory which is limited in the kind of data it can store. It was therefore necessary to fool the texture memory into storing two 32bit floating point values while actually storing a 64bit complex.

The FFTs needed in the operator are implemented in two ways based on different libraries. The faster implementation is based on NVIDIAs CUFFT library [15] and the corresponding Python wrapper *scikits.cuda* [20]. The disadvantage of CUFFT is that it stores several intermediate arrays which has the effect of filling the GPU memory very rapidly in case of big reconstruction sizes. As the FFTs are oversampled[2], this is an issue that needs consideration. This can be a problem on consumer GPUs with less memory than high-end GPUs for technical computing. To overcome this limitation, a second implementation based on the FFT library PyFFT [21] was done. PyFFT dynamically creates FFT kernels based on the size of the problem. It its significantly slower (about factor 2) than CUFFT and can only handle grids with sizes of power of 2. However, it requires less memory and can therefore handle larger reconstruction sizes on cheaper GPUs.

To avoid time-consuming transfers of data between CPU and GPU, all intermediate steps are also implemented as GPU code but will not be mentioned here in detail, as they are less interesting in terms of performance compared to the sparse matrix vector multiplication and the FFTs.

[2]Usually four times the actual reconstruction size. For instance, reconstructing on a 512×512 image grid requires FFTs on data of size 2048×2048.

2.6. Reconstruction

In this section iterative reconstruction methods which are applied to the above mentioned operators are introduced.

2.6.1. Inversion of the Encoding Matrix

Because of the computationally challenging task of inverting $\boldsymbol{E}$ in Eq. 2.7 to get the desired m-vector, a conjugate gradient method [6] is used instead of inverting $\boldsymbol{E}$ directly. This also has the benefit of a "built-in" regularization which prevents overamplification of noise terms as long as the number of iterations is chosen carefully [12].

Equation 2.7 can now be solved by computing the pseudoinverse of $\boldsymbol{E}$:

$$\boldsymbol{m} = \boldsymbol{E}^{+}\boldsymbol{s} = (\boldsymbol{E}^{H}\boldsymbol{E})^{-1}\boldsymbol{E}^{H}\boldsymbol{s} \tag{2.10}$$

Rewriting Eq. 2.10 leads to:

$$\boldsymbol{E}^{H}\boldsymbol{E}\boldsymbol{m} = \boldsymbol{E}^{H}\boldsymbol{s} \tag{2.11}$$

Equation 2.11 can now be solved using the conjugate gradient method [6] on the normal equations.

GPU Implementation

The conjugate gradient method requires the application of $\boldsymbol{E}^{H}$ and $\boldsymbol{E}$ to vectors in each iteration. These steps consume the most time in the reconstruc-

tion due to the size of $\boldsymbol{E}^H$ and $\boldsymbol{E}$. Therefore it is necessary to optimize these matrix-vector multiplications accordingly for maximum performance.

Several problems arise when implementing these operations. The problems and their solutions will be discussed in this section.

The main problem is the size of the matrix $\boldsymbol{E}$ and its adjoint, not just because the size is responsible for the long reconstruction times, but also because it is not possible to fit the matrices into the memory, particularly not in GPU RAM, in almost all practical cases. For example, a measurement consisting of 256×256 k-space data from eight coils, which should be reconstructed onto a 512×512 image grid would require 1 Terabyte ($N_c \times N_k \times N_\rho \times \text{sizeof(complex float)}$) to keep $\boldsymbol{E}$ in the memory – far too much for current hardware which is usually limited to around 6-8 Gigabytes of RAM on high-end GPUs. Therefore it is generally not possible to precompute $\boldsymbol{E}$ or $\boldsymbol{E}^H\boldsymbol{E}$ and it is necessary to compute at least parts of $\boldsymbol{E}$ on the fly in each iteration.

A first approach in reducing the memory usage would be to precompute parts of $\boldsymbol{E}$ instead of the full matrix. Well suited for this approach is the precomputation of the phase term matrix $e^{-i\phi(\boldsymbol{x},t)}$. In the encoding matrix, the phase term matrix is repeated for each coil, therefore the memory consumption can be reduced by the factor N_c (number of coils). Considering the example above, the memory consumption would still be at 127 Gigabytes – still far to much for current hardware.

The most memory efficient approach is to compute each element of $\boldsymbol{E}$ whenever it is needed. This approach is also known as matrix-free computation. However, this is very time-consuming because the same computations have to be performed for each application of the operator or its adjoint. On the other hand it also allows to utilize the characteristics of

the encoding matrix by prefetching data that is to be used several times. The following will explain how this is done when applying $\boldsymbol{E}$ and $\boldsymbol{E}^H$ to a vector.

Construction of E and Multiplication with a Vector The matrix $\boldsymbol{E}$ can be seen as a consecutive collocation of N_c equal $e^{-i\phi(\boldsymbol{x},t)}$ matrices, where N_c denotes the number of coils. Each of these $e^{-i\phi(\boldsymbol{x},t)}$ matrices is multiplied row by row by its corresponding coil sensitivity $c_\alpha(\boldsymbol{x})$.

When performing the multiplication of $\boldsymbol{E}$ with the vector $\boldsymbol{m}$, special care has to be taken considering the characteristics of GPGPU programming using CUDA. In particular the problem has to be divided into blocks which are further divided into threads. Threads share certain properties within a block, i.e. the so called shared memory. For details on GPGPU programming see Section 2.2. Choosing parameters like block alignment and the number of threads per block considerately can lead to a major boost in performance.

In this case, the scalar product of each row of $\boldsymbol{E}$ with the vector $\boldsymbol{m}$ is computed by one block, therefore the number of blocks equals the number of rows of $\boldsymbol{E}$. The number of threads per block is either equal to the number of columns of $\boldsymbol{E}$ or equal to the maximum number of threads allowed on a particular graphics card model, depending on which one is lower. In nearly all practical cases the number of threads per block is equal to the maximum number of threads.

For the sake of simplicity and to make the explanation of the principle easier, it is assumed that each block consists of only four threads. The following explanation is supported by Fig. 2.5.

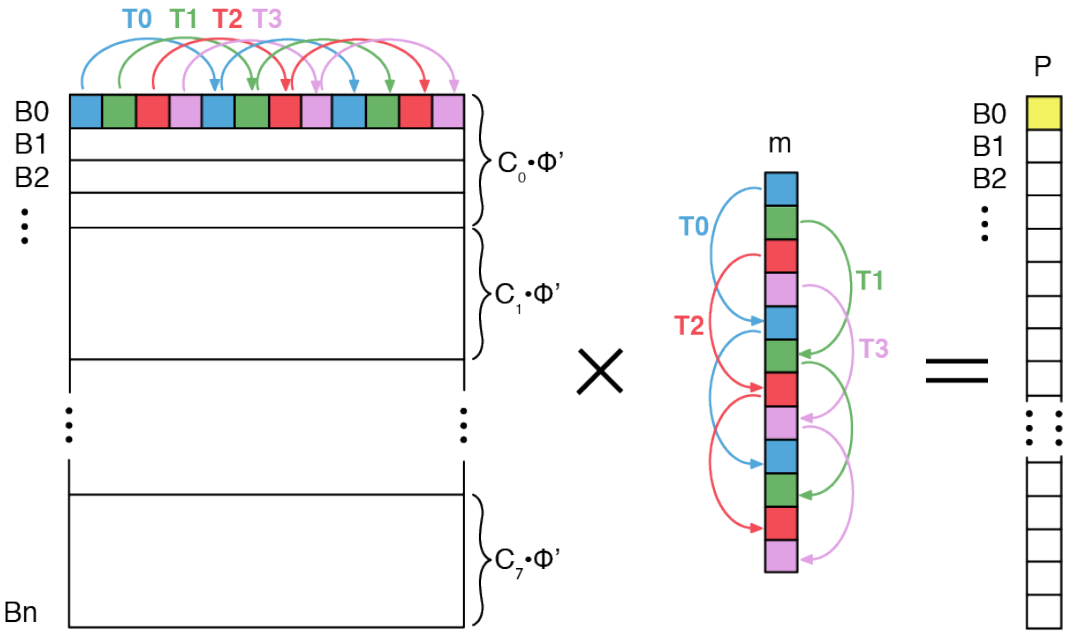


Figure 2.5.: Multiplication of the encoding matrix $\boldsymbol{E}$ with the vector $\boldsymbol{m}$. The encoding matrix consists of N_c phase term matrices $\Phi' = e^{-i\phi(x,t)}$ stacked together vertically, each multiplied row-wise and point wise with the corresponding coil sensitivity c_n. Due to memory limitations, $\boldsymbol{E}$ is not kept in the memory of the GPU, but each element of $\boldsymbol{E}$ is computed when it is needed. Illustration also shows how this is implemented on the GPU. See text for a detailed explanation.

Thread `T0` of block `B0` takes the first element of $\boldsymbol{E}$ and $\boldsymbol{m}$ and stores the product of these two elements in the first element of a temporary vector `tmp0` (same length as number of threads per block). Thread `T0` then jumps to the index 4 of the vector $\boldsymbol{m}$ and the first row of the matrix and performs the same action. This step is repeated until the position of the thread exceeds the dimensions of $\boldsymbol{m}$. The other Threads `T1` - `T3` perform the same computations, but shifted to the right according to their thread index.

It should be noted that in the actual implementation every element of $\boldsymbol{E}$ is only constructed when needed by taking the respective values of c_n, $\boldsymbol{k}$ and

$\boldsymbol{\psi}$, according to Eq. 2.8.

When all threads of the corresponding block have finished their element wise multiplication, the temporary vector `tmp0` is completely filled with values which have to be summed up to complete the computation. The values are now summed up by the first half of the threads (in this example `T0` and `T1`). `T1` sums up element 0 and element 2, whereas `T1` sums up element 1 and element 3. The threads `T2` and `T3` are finished at this point. The results are once again saved in another temporary vector `tmp0'` which is only half as long as `tmp0`. This step is repeated until all elements of the initial `tmp0` are summed up. This scheme assures that *warps*[3] are finished as fast as possible, therefore leading to better performance. Figure 2.6 illustrates how the elements of `tmp0` are summed up.

This principle is the same for every block. Apart from this specific way of computing the matrix-vector product, no further optimizations have been implemented. Due to the structure of $\boldsymbol{E}$ it is not possible to prefetch certain elements of the matrix or the vector to minimize the number of memory accesses.

This concept works for small datasets which do not hit certain CUDA limitations. One of the limitations is that at the time of writing the number of blocks is limited to 65535 per kernel call, it is therefore essential that $N_\kappa \times N_c \leq 65535$. Assuming that in most measurements with the PatLoc hardware eight coils are used, the k-space resolution is limited to a total number of ≈ 8191 points. This could be achieved by a 90×90 k-space grid, which is obviously not enough for serious imaging applications. There are

[3] *Warps* are a set of threads that are processed at once on the GPU. It should be avoided to execute *warps* with less threads than possible because it hinders other threads to run, therefore leading to situations where threads have to wait to get a slot although processor cores are free.

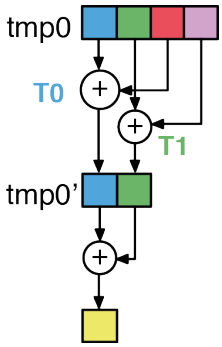


Figure 2.6.: Efficient implementation of summing up all elements of a vector on the GPU, illustrated for four threads. In each step the number of threads is reduced by half. As shown here, the threads `T0` and `T1` each sum up two elements of vector `tmp0`, leading to a new vector `tmp0'` half the size of `tmp0`. This is repeated until all elements are summed up. This scheme assures that *warps* finish as fast as possible.

two ways to overcome this limitation. First of all it is not necessary to multiply each of the consecutive phase term matrices with the corresponding coil sensitivity, but instead the $\boldsymbol{m}$ vector is multiplied by the coil sensitivity vectors of each coil, leading to N_c vectors $m(\boldsymbol{x})c_\alpha(\boldsymbol{x})$. As illustrated in Fig. 2.7, this decreases the number of blocks by a factor of N_c. Additionally, due to the fact that each coil sensitivity vector does not need to be multiplied with each row of the phase term matrix $e^{-i\phi(\boldsymbol{x},t)}$ and each element of the phase term matrix has to be computed only once, there is less computational load resulting in a performance boost. Unfortunately, there still is a limitation to $N_\kappa \leq 65535$. To overcome this limitation completely, the algorithm has to be made more "serial" which is achieved by processing several rows of the matrix per block as also shown in Fig. 2.7. The number of rows each block "jumps" in the reconstruction is defined by the *hop* parameter.

Consumer graphics cards exhibit another limitation, namely that kernel calls are aborted if they consume too much time. Therefore it is possible

to divide the problem into subproblems where several kernel calls are performed on different parts of the data, hence loosing some of the parallelism of the implementation.

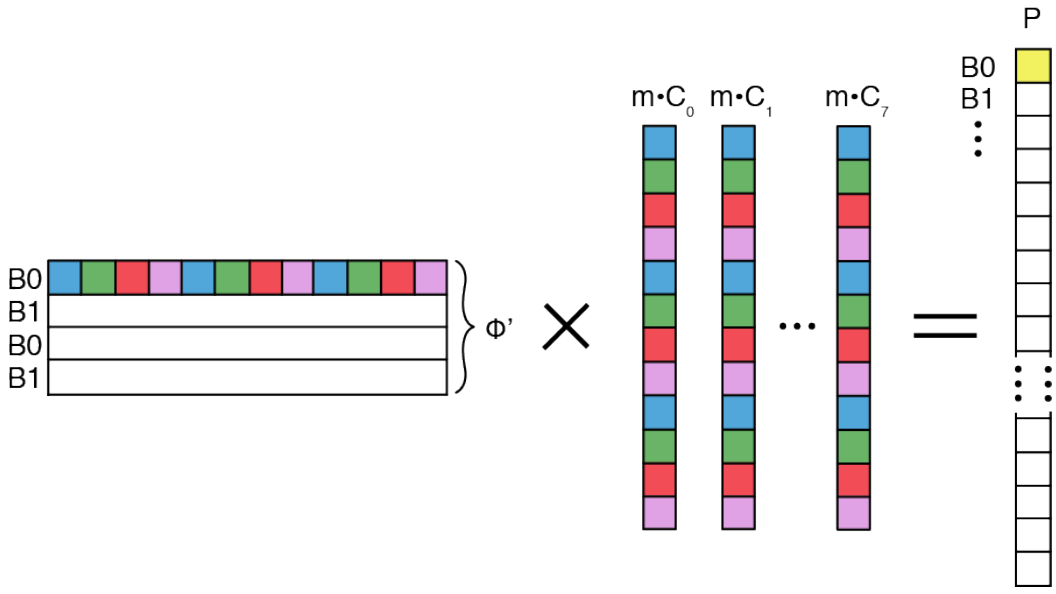


Figure 2.7.: More efficient implementation of the principle outlined in Fig. 2.5. The coil sensitivities are not multiplied element wise with each row of the corresponding phase term matrix Φ' but rather multiplied point wise with the vector $\boldsymbol{m}$, leading to N_c vectors. This reduces both the number of blocks and the computational load, because the size of the matrix is reduced by N_c and less point wise multiplications are needed as well as smart prefetching and reusing of already computed elements of Φ' can be performed. To reduce the number of blocks even further, each block processes several rows of the phase term matrix Φ (defined by the *hop* parameter). For a detailed explanation see the text.

Construction of E^H and Multiplication with a Vector The basic principles for the matrix vector product $\boldsymbol{E}^H\boldsymbol{p}$ are the same as for $\boldsymbol{Em}$, with the difference that $\boldsymbol{E}$ is transposed. Due to the structure of $\boldsymbol{E}^H$ it is possible

to implement prefetching routines to minimize the number of memory accesses.

As already explained, $\boldsymbol{E}$ consists of N_c matrices $e^{-i\phi(\boldsymbol{x},t)}$ which are row by row multiplied by the corresponding coil sensitivity. Transposing $\boldsymbol{E}$ leads to a structure where each row consists of several equal values. For example, each row needs only N_c values of the coil sensitivity matrix. A similar setting applies for the transposed $e^{-i\phi(\boldsymbol{x},t)}$ term. In each row of $\boldsymbol{E}^H$ one column of $e^{-i\phi(\boldsymbol{x},t)}$ is repeated N_c times. Figure 2.8 illustrates the process.

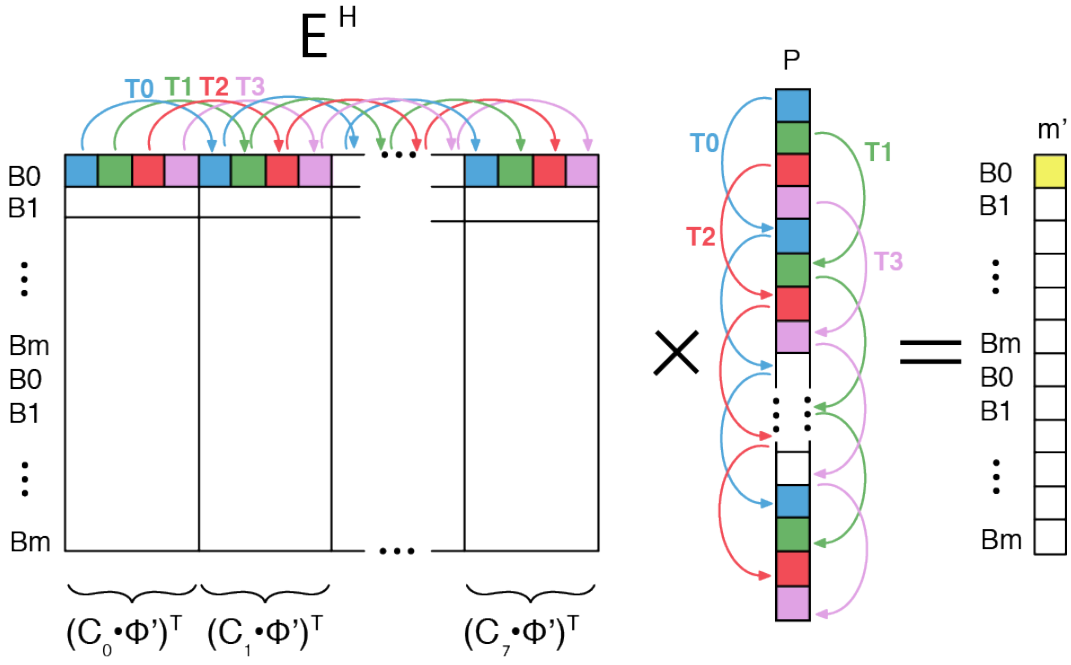


Figure 2.8.: Implementation of the multiplication of the adjoint encoding matrix $\boldsymbol{E}^H$ with vector p. In contrast to the multiplication of the encoding matrix with a vector, it is not possible to reduce the number of blocks needed by N_c because of the transposing of the phase term matrices Φ'. Therefore it is also not possible to multiply p with the coil sensitivities c_n. But the multiplication with the adjoint offers other possibilities for optimization which are explained in the text.

The CUDA algorithm takes advantage of these properties by prefetching all

needed values only once, thereby providing an extremely effective solution for this computationally intensive task.

Further the prefetched coil sensitivity values are stored in the shared memory. The shared memory is a very fast part of the GPU memory which is accessible by all threads of a block. Since each value of the coil sensitivity is needed by several threads, the memory access times are reduced.

Similar to the multiplication of $\boldsymbol{E}$ with a vector, the limitation on the number of blocks is an issue that has to be addressed accordingly. Again, each block processes several rows consecutively if the number of rows exceeds the maximum number of blocks, as illustrated in Fig. 2.8.

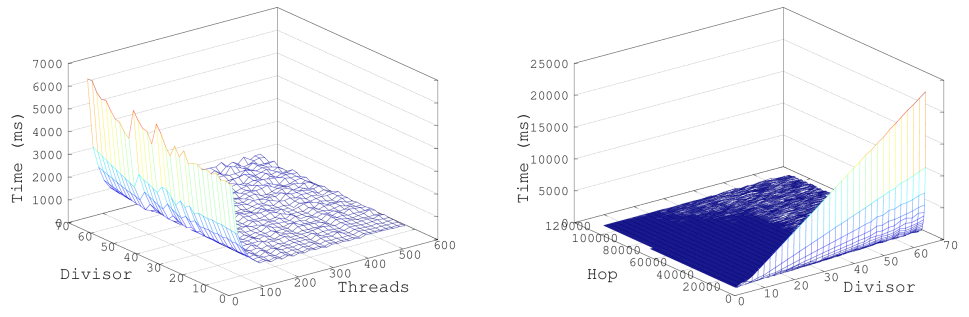
Choice of Parameters

The illustrated implementation and CUDA itself offer various parameters which need to be tuned in order to achieve maximum performance. This leads to the issue of finding parameter sets that work well with the given data. The fact that the parameters depend on the datasets, especially the size of the datasets, makes the optimization even more difficult.

One of most obvious parameters that needs to be chosen is the number of threads per block. Although a high number of threads is desirable, too many threads can lead to a drop in performance as it can cause memory issues, which has the effect that threads have to wait for their execution. Further, NVIDIAs consumer graphics cards limit the execution time of one kernel. Therefore the kernel execution has to be subdivided into subproblems which are executed one after the other. This is represented by the *divisor* parameter in the implementation. However, dividing the problem has the effect of serializing a problem which actually could be run in parallel. Hence performance drops if the problem is divided into too many

subproblems. Similar effects can be observed for the *hop* parameter, which defines how far a block "jumps" when processing several rows of the matrix. A low *hop* value means that each block has to process many rows in each call, again turning a parallel problem into a more serial one, causing a performance drop.

To assess how important these issues are, an exhaustive search was performed for several different datasets. Figure 2.9 shows two plots for an example dataset. In Fig. 2.9(a) the *hop* parameter was fixed and the execution time is plotted depending on the *divisor* and *threads* parameters, whereas in Fig. 2.9(b) the *threads* parameter was fixed.



(a) *hop* parameter fixed

(b) *threads* fixed

Figure 2.9.: Example plots of an exhaustive search of the parameter space, whereby in (a) the *hop* parameter and in (b) the number of threads is fixed. The execution time is plotted over the remaining parameters. Plots show that there are sets of parameters that should be avoided for good performance.

It turned out that it is actually harder to find a bad set of parameters rather than finding a good one. A low *divisor* parameter definitely leads to better performance as long as it is high enough to not cause problems with execution itself. The *hop* value can be chosen as high as possible, only limited by the maximum number of blocks. For the *threads* parameter, 512 is a good

choice for all datasets investigated.

2.6.2. Total Generalized Variation (TGV)

In this chapter the regularization technique known as *Total Generalized Variation* (TGV) as introduced in Section 1 will be explained in detail.

Inverse problems are often expressed as minimization problems in the form of

$$\min_u \|\mathcal{F}(u) - g\|_2^2 + \mathcal{R}(u) \tag{2.12}$$

where $\|\mathcal{F}(u) - g\|_2^2$ is known as the data fidelity term, consisting of the forward operator $\mathcal{F}$, the noisy data g and a norm $\|\cdot\|$. $\mathcal{R}$ is the regularization term, which can be defined in many ways, one of the easiest being

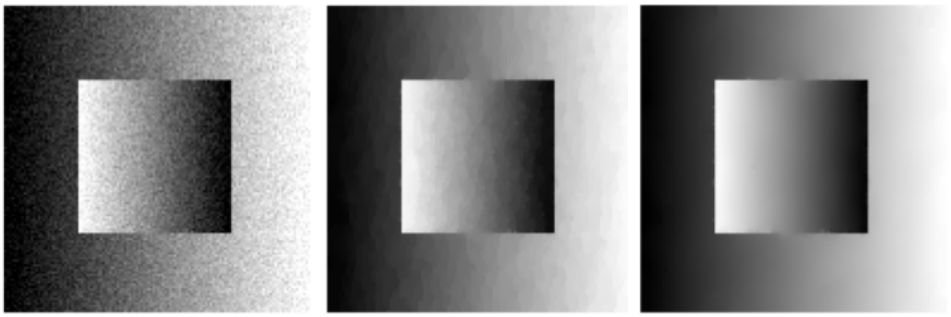
$$\mathcal{R}(u) = \alpha\|u\|_2^2 \tag{2.13}$$

where α is the regularization parameter. Although fast to compute, it heavily penalizes outliers, a property that can be disadvantageous in many cases. A better choice is Total Variation (TV) as regularization term $\mathcal{R}$, which was introduced in [27] for image denoising:

$$\mathcal{R}(u) = \mathrm{TV}(u) = \int_\Omega |\nabla u| \mathrm{d}x \tag{2.14}$$

If the reconstructed image consists of piece-wise constant areas, TV regularization leads to good results as it preserves edges and corners. However, TV is not capable of handling smooth areas where it leads to additional edges, known as staircasing artifacts (Fig. 2.10(b)).

To overcome the disadvantages of Total Variation, higher order derivatives can be incorporated into the image model, as it is done in the concept of



(a) Noisy image (b) TV (c) TGV

Figure 2.10.: Denoising of (a) a noisy image with (b) Total Variation and (c) Total Generalized Variation. TV introduces artificial edges in areas with smooth transitions of the gray values, known as staircasing artifact, whereas TGV is capable of reconstructing the smooth transitions in the image artifact-free.

Total Generalized Variation (TGV) which was first introduced by Bredies et al. in [7]. The second-order Total Generalized Variation is defined as

$$\begin{aligned} \mathcal{R}(u) &= \mathrm{TGV}^2_\alpha(u) = \\ &= \min_v \alpha_1 \int_\Omega |\nabla u - v| \mathrm{d}x + \alpha_0 \int_\Omega \left| \frac{1}{2} \left(\nabla v + \nabla v^T \right) \right| \mathrm{d}x \end{aligned} \quad (2.15)$$

where $\frac{1}{2}\left(\nabla v + \nabla v^T\right)$ denotes the symmetrized derivative which is also denoted as $\varepsilon(v)$ and α_0 and α_1 are the regularization parameters. The TGV regularization was recently successfully applied to MR image reconstruction of undersampled data by Knoll et al. in [8, 10]. It also proved to lead to significant improvements in PatLoc imaging for undersampled data in [11].

Figure 2.10 illustrates the differences between Total Variation and Total Generalized Variation in case of denoising a noisy image (Fig. 2.10(a)). Total Variation (Fig. 2.10(b) clearly exhibits a staircasing artifact by introducing additional edges which are not part of the underlying noise-free image.

Total Generalized Variation on the other hand is able to reconstruct the smooth transitions without introducing artifacts (Fig. 2.10(c)).

2.6.3. Numerical Solution for TGV

To solve the inverse problem regularized with TGV, many numerical algorithms for solving convex-concave saddle point problems can be used. In this work, the primal-dual ascent-descent method with primal extragradient introduced in [28] by Chambolle and Pock is used.

As shown in chapter 3 of [9], the algorithm is implemented as follows:

1. Choose $\tau_p > 0$, $\tau_d > 0$ such that
   $\tau_d \tau_p \frac{1}{2} \left(\sqrt{\left(\|\mathcal{F}\|^2 - 1 \right)^2 + 32} + \|\mathcal{F}\|^2 + 17 \right) \leq 1$
2. Choose $(u^0, p^0) \in U \times V$, $(v^0, w^0, \lambda^0) \in V \times W \times \Lambda$ and set $\bar{u}^0 = u^0$, $\bar{p}^0 = p^0$.
3. For $n = 0, 1, 2, \ldots$ iterate according to
   $$\begin{aligned} v^{n+1} &= \mathcal{P}_{\alpha_0} \left(v^n + \tau_p \left(\nabla \bar{u}^n - \bar{p}^n \right) \right) \\ w^{n+1} &= \mathcal{P}_{\alpha_1} \left(w^n + \tau_p \mathcal{E} \left(\bar{p}^n \right) \right) \\ \lambda^{n+1} &= \frac{\lambda^n + \tau_p (\mathcal{F} \bar{u}^n - g)}{1 + \tau_p} \\ u^{n+1} &= u^n + \tau_d \left(\nabla \circ v^{n+1} - \mathcal{F}^* \lambda^{n+1} \right) \\ p^{n+1} &= p^n + \tau_d \left(v^{n+1} + \nabla \circ w^{n+1} \right) \\ \bar{u}^{n+1} &= 2u^{n+1} - u^n \\ \bar{p}^{n+1} &= 2p^{n+1} - p^n \end{aligned}$$
4. Return u^N for some large N.

To be in accordance with the MATLAB reference implementation from [11], the parameters τ_p and τ_d are set to 0.0625 and 0.125, respectively.

Estimation of $\|\mathcal{F}\|$

The norm of the operator $\mathcal{F}$ is an important factor for the convergence speed. The algorithm shown in Section 2.6.3 assumes that $\|\mathcal{F}\| \leq 1$. Since this is not the case for the operators investigated in this work, each application of $\mathcal{F}$ or $\mathcal{F}^*$ needs to be normalized by $\|\mathcal{F}\|$ to meet this constraint.

Usually a good estimate can be obtained with the following iterative algorithm:

1. Choose $u^0 = \vec{0}$ or $u^0 = \mathcal{F}^* g$ where g is the data.
2. For $i = 0, 1, 2, \ldots$ iterate according to

$$u^{i+1} = \frac{\mathcal{F}^* \mathcal{F} u^i}{\sqrt{|\langle u^i, u^i \rangle|}}$$

3. Return $\|\mathcal{F}\|_{\text{est}} = \sqrt{|\langle u^N, u^N \rangle|}$ for $N = 10$ which usually is sufficiently high.

Increasing the step width by modifying $\|\mathcal{F}\|_{\text{est}}$ In some cases it can be advantageous to increase the step width by decreasing the estimated norm in the algorithm by a factor d to speed up convergence. However, this has to be done with care, considering the fact that setting this value too high can lead to oscillating effects in the reconstruction, resulting in useless images. For the cases investigated in this work if not noted otherwise, setting $d = 1000$ improves convergence significantly while still preventing oscillations.

Regularization parameters α_0 and α_1

In accordance with the implementation from [11], the regularization parameters α_0 and α_1 are not set directly, but instead depend on the current iteration. The regularization parameters are reduced exponentially in each iteration, hence the parameters to set are α_{10} and α_{00}. The latter is usually set to $l \cdot \alpha_{10}$ where $l = 2$. The parameters α_{00} and α_{10} are then multiplied by a reduction factor r, usually 2^{-8}, leading to α_{01} and α_{11}, respectively. In each iteration k of a total of N iterations, the current α_0 and α_1 are acquired by

$$\alpha_0 = \exp\left(\frac{k}{N}\log\alpha_{01} + \frac{N-k}{N}\log\alpha_{00}\right) \tag{2.16}$$

$$\alpha_1 = \exp\left(\frac{k}{N}\log\alpha_{11} + \frac{N-k}{N}\log\alpha_{10}\right) \tag{2.17}$$

Figure 2.11 illustrates the adaptive regularization parameters.

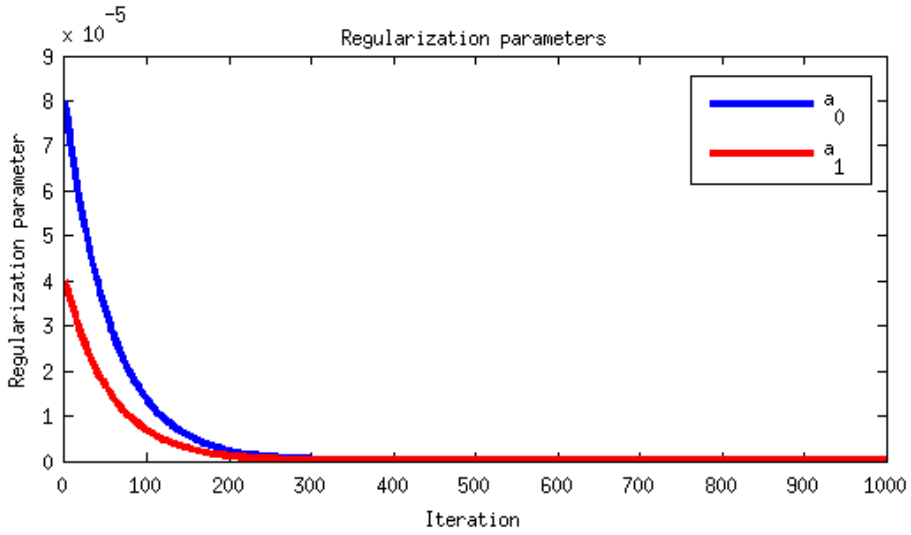


Figure 2.11.: Development of the regularization parameters α_0 and α_1 for $\alpha_{01} = 4 \cdot 10^{-5}$, $l = 2$, $r = 2^{-8}$ and $N = 1000$.

GPU Implementation

The operations mentioned in the algorithm of Section 2.6.3 are executed on the GPU. From a performance point of view this is not essential as these operations are rather fast compared to the evaluation of the operators $\mathcal{F}$ and $\mathcal{F}^*$. However, mixing GPU and CPU code requires the movement of data between both architectures, resulting in a significant performance drop.

Since pixels can mostly be handled individually, all the operations are well suited for parallel execution.

2.6.4. TGV - Conjugate Gradient (TGV-CG)

TGV-CG is a modification of the numerical algorithm used to solve the TGV regularized inverse problem shown in Section 2.6.3. This leads to the loss of the variable λ but instead requires evaluating the equation

$$u^{n+1} = (\mathrm{id} + \tau_d \mathcal{F}^* \mathcal{F})^{-1} \left(u^n + \tau_d \left(\nabla \circ v^{n+1} + \mathcal{F}^* g \right) \right) \tag{2.18}$$

for the update of u. The inversion of $(\mathrm{id} + \tau_d \mathcal{F}^* \mathcal{F})$ is due to the size of the operator $\mathcal{F}$ computationally too challenging. Therefore it is solved with an additional conjugate gradient method in each TGV iteration, hence giving the method the name TGV-CG.

The algorithm is implemented the following way:

1. Choose $\tau_p > 0$, $\tau_d > 0$ such that $\tau_d \tau_p \frac{1}{2} \left(17 + \sqrt{33} \right) \leq 1$
2. Choose $(u^0, p^0) \in U \times V$, $(v^0, w^0) \in V \times W$ and set $\bar{u}^0 = u^0$, $\bar{p}^0 = p^0$.
3. For $n = 0, 1, 2, \ldots$ iterate according to

$$v^{n+1} = \mathcal{P}_{\alpha_0} \left(v^n + \tau_p \left(\nabla \bar{u}^n - \bar{p}^n \right) \right)$$

$$
\begin{aligned}
w^{n+1} &= \mathcal{P}_{\alpha_1}\left(w^n + \tau_p \varepsilon\left(\bar{p}^n\right)\right) \\
u^{n+1} &= \left(\mathrm{id} + \tau_d \mathcal{F}^* \mathcal{F}\right)^{-1}\left(u^n + \tau_d\left(\nabla \circ v^{n+1} + \mathcal{F}^* g\right)\right) \\
p^{n+1} &= p^n + \tau_d\left(v^{n+1} + \nabla \circ w^{n+1}\right) \\
\bar{u}^{n+1} &= 2u^{n+1} - u^n \\
\bar{p}^{n+1} &= 2p^{n+1} - p^n
\end{aligned}
$$

4. Return u^N for some large N.

This modification is expected to converge faster than TGV with the primal-dual algorithm. However, for the reconstruction time, the iterations of the conjugate gradient method have to be considered in addition to the number of iterations N, because the operators $\mathcal{F}$ and $\mathcal{F}^*$ have to be applied in each CG iteration. Therefore it may result in increased reconstruction time compared to solving the problem with the primal-dual algorithm. In Chapter 3 it will be shown that it depends on the problem and especially on the chosen operator whether this method is advantageous compared to the primal-dual implementation or not.

It proved well to set τ_d and τ_p depending on $\|\mathcal{F}\|_{\text{est}}$ (Section 2.6.3). Choosing

$$
\tau_p = \frac{1}{\|\mathcal{F}\|} \tag{2.19}
$$

leads to

$$
\tau_d = \frac{1}{\tau_p \frac{1}{2}\left(17 + \sqrt{33}\right)} \tag{2.20}
$$

in order to meet the constraint

$$
\tau_d \tau_p \frac{1}{2}\left(17 + \sqrt{33}\right) \leq 1 \tag{2.21}
$$

GPU Implementation

Most of the operations shown in the algorithm above are equal to the ones in Section 2.6.3. Additionally a conjugate gradient method was implemented taking care of the requirements of Eq. 2.18. However, in principle the implementation of the CG method follows standard implementations.

2.6.5. Reconstruction Methods

The numerical solution of TGV (Section 2.6.3) and TGV-CG (Section 2.6.4) as well as the conjugate gradient method have been applied to data with use of the encoding matrix operator (Section 2.5.1) and the PatLoc NUFFT 1+2 (2.5.2) operator. This effectively leads to the following methods:

- Encoding Matrix CG
- Encoding Matrix TGV
- Encoding Matrix TGV-CG
- PatLoc NUFFT 1+2 CG
- PatLoc NUFFT 1+2 TGV
- PatLoc NUFFT 1+2 TGV-CG

These methods are analyzed in Section 3.

2.7. Performance Analysis

For the evaluation of the performance, the execution time of the GPU code based on the encoding matrix (Section 2.5.1) was compared to a MATLAB implementation provided by the Department of Radiology, Medical

Physics of the University Hospital Freiburg, Germany. The code based on the NUFFT 1+2 (Section 2.5.2) was also compared to a MATLAB implementation by Florian Knoll from the Institute of Medical Engineering, Graz, Austria.

The time measurements were performed in PYTHON by the `time` module. To avoid parallel execution of the time measurement and the computations on the GPU, care has been taken to appropriately synchronize between CPU and GPU code before calling time-measurement functions. The MATLAB solutions use `tic` and `toc`.

The time has been measured per iteration which is sufficient because the GPU based implementations converge at the same speed as the corrensponding CPU based implementation.

3. Results

The image quality has been assessed with several datasets acquired for this work (Section 2.4). A Cartesian sampling pattern was used as well as a radial sampling pattern for both in-vivo and phantom data.

In Figs. 3.1 (in-vivo) and 3.3 (phantom) the reconstructions for the Cartesian sampling pattern using the encoding matrix forward operator are shown. The same dataset has also been reconstructed with the PatLoc NUFFT 1+2 operator as shown in Figs. 3.2 and 3.4.

The second dataset was acquired using a radial trajectory. Figures 3.5 and 3.7 show the reconstructions using the encoding matrix operator for both the in-vivo data as well as the phantom data. The datasets have also been reconstructed with the PatLoc NUFFT 1+2 operator in Figs. 3.6 and 3.8.

CG and TGV reconstructions for undersampled Cartesian data (Fig. 3.9) and undersampled radial data (Fig. 3.10) are shown as well.

Further, a dataset acquired with the NorthWest EPI (Section 2.4.1) trajectory (Fig. 3.11) was used to assess the performance of the algorithm on multi-dimensional trajectories.

Finally, plots depicting the speedup (Fig. 3.12) and a plot showing the convergence of the inner CG of TGV-CG (Fig. 3.13).

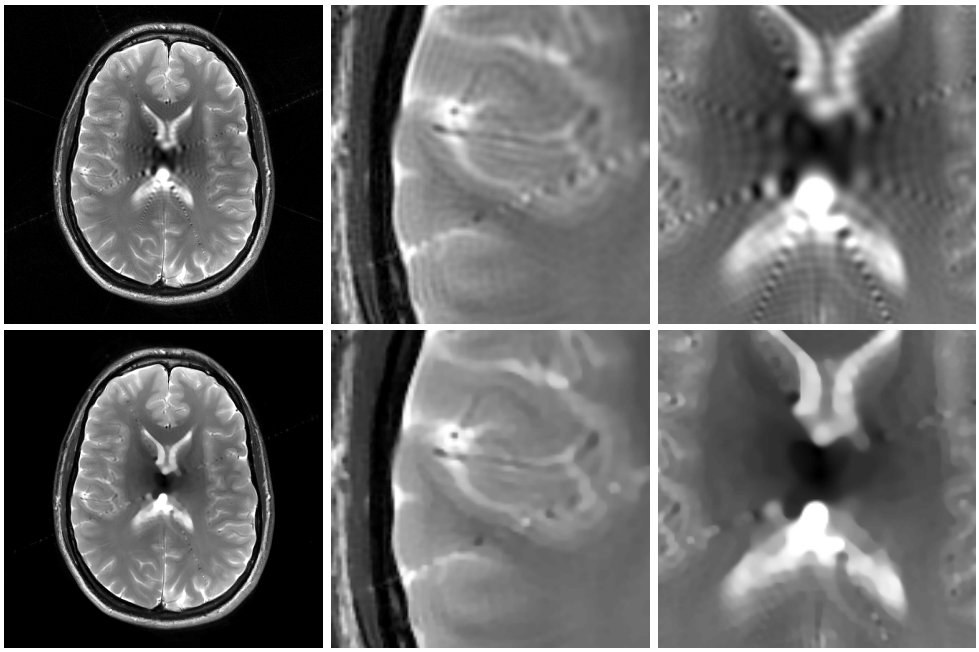


Figure 3.1.: Reconstruction of in-vivo human data acquired with a Cartesian trajectory and a Turbo Spin Echo (TSE) sequence (Section 2.4). The top row shows the image reconstructed with a conventional CG method (50 iterations) and the encoding matrix operator as well as two detailed views. The second row shows the results for the TGV-CG regularized reconstruction. The regularization parameter has been set to $\alpha = 1 \cdot 10^{-1}$. TGV-CG required 200 PD iterations.

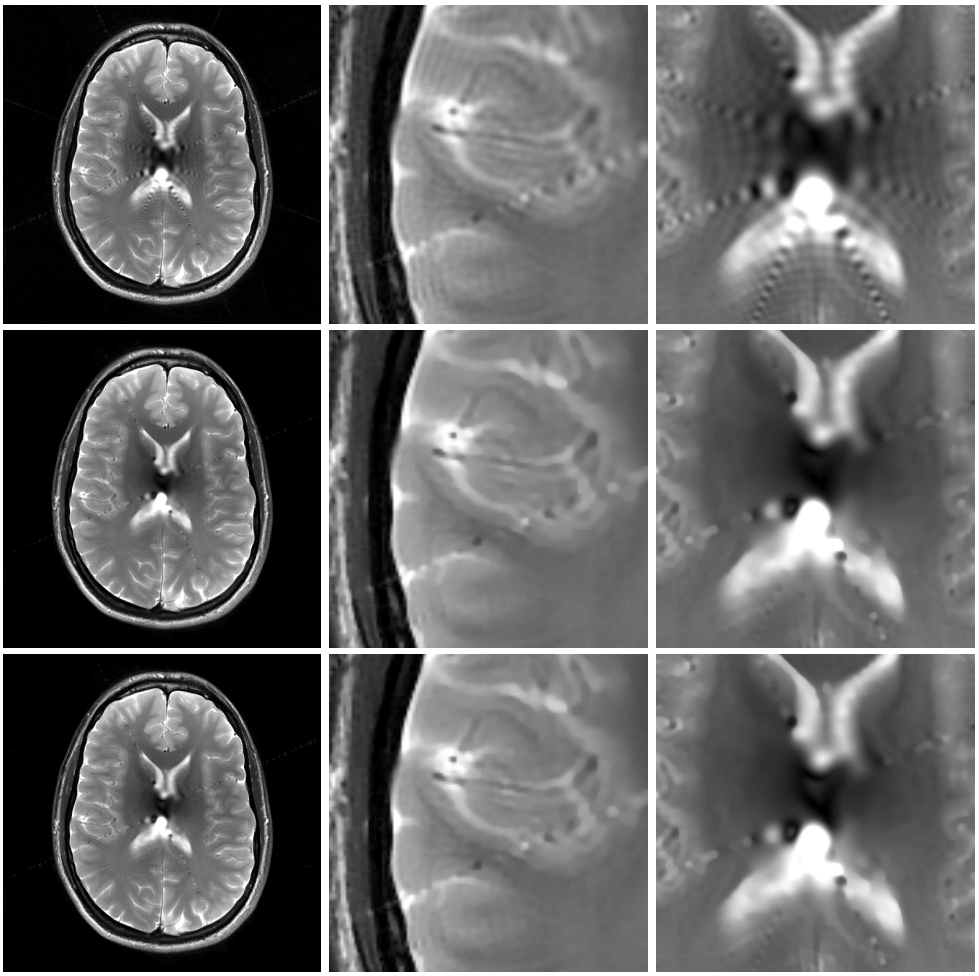


Figure 3.2.: Reconstruction of in-vivo human data acquired with a Cartesian trajectory and a Turbo Spin Echo (TSE) sequence (Section 2.4). The top row shows the image reconstructed with a conventional CG method (50 iterations) and the PatLoc NUFFT 1+2 operator as well as two detailed views. The rows two and three show the results for TGV and TGV-CG regularized reconstruction, respectively. The regularization parameter has been set to $\alpha = 4 \cdot 10^{-5}$ in both cases. TGV required 2000 PD iterations, whereas TGV-CG required 200 PD iterations.

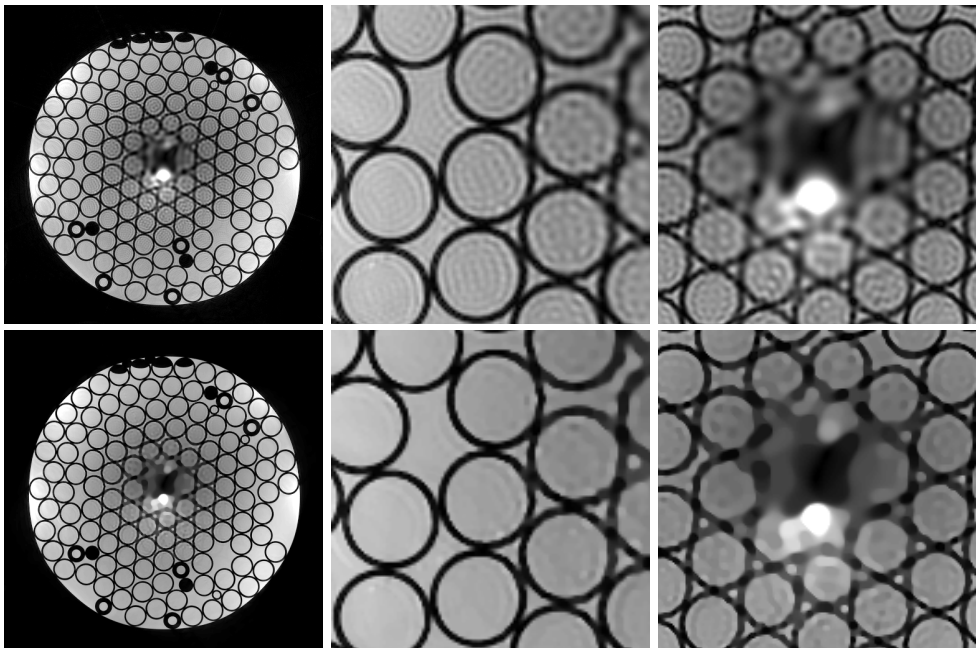


Figure 3.3.: Reconstruction of phantom data acquired with a Cartesian trajectory and a Turbo Spin Echo (TSE) sequence (Section 2.4). The top row shows the image reconstructed with a conventional CG method (50 iterations) and the encoding matrix operator as well as two detailed views. The second row shows the results for the TGV-CG regularized reconstruction. The regularization parameter has been set to $\alpha = 1 \cdot 10^{-1}$. TGV-CG required 200 PD iterations.

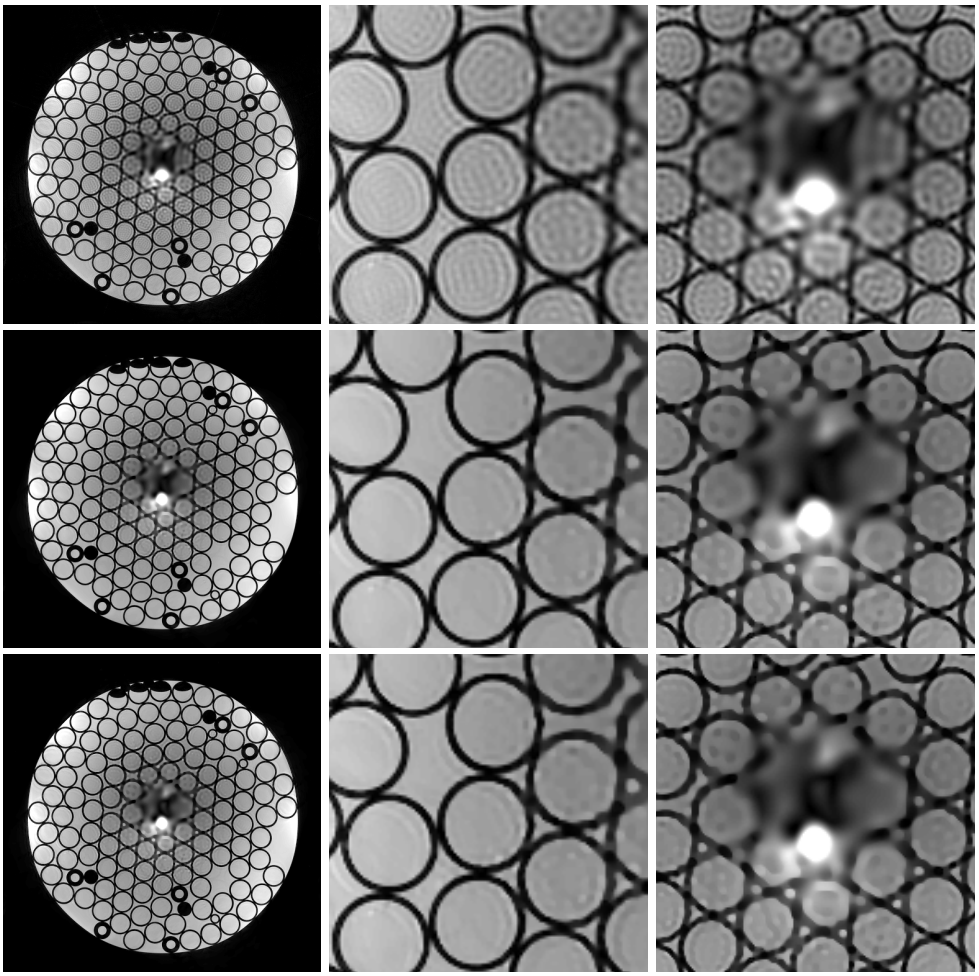


Figure 3.4.: Reconstruction of phantom data acquired with a Cartesian trajectory and a Turbo Spin Echo (TSE) sequence (Section 2.4). The top row shows the image reconstructed with a conventional CG method (50 iterations) and the PatLoc NUFFT 1+2 operator as well as two detailed views. The rows two and three show the results for TGV and TGV-CG regularized reconstruction, respectively. The regularization parameter has been set to $\alpha = 4 \cdot 10^{-5}$ in both cases. TGV required 2000 PD iterations, whereas TGV-CG required 200 PD iterations.

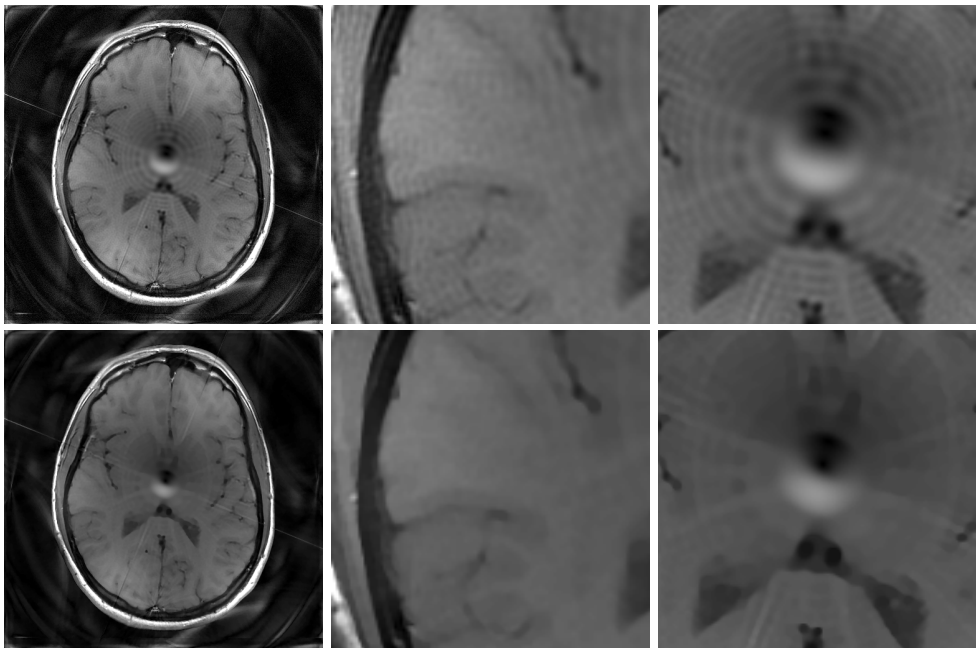


Figure 3.5.: Reconstruction of in-vivo human data acquired with a Cartesian trajectory and a Turbo Spin Echo (TSE) sequence (Section 2.4). The top row shows the image reconstructed with a conventional CG method (50 iterations) and the encoding matrix operator as well as two detailed views. The second row shows the results for the TGV-CG regularized reconstruction. The regularization parameter has been set to $\alpha = 1 \cdot 10^{-1}$. TGV-CG required 200 PD iterations.

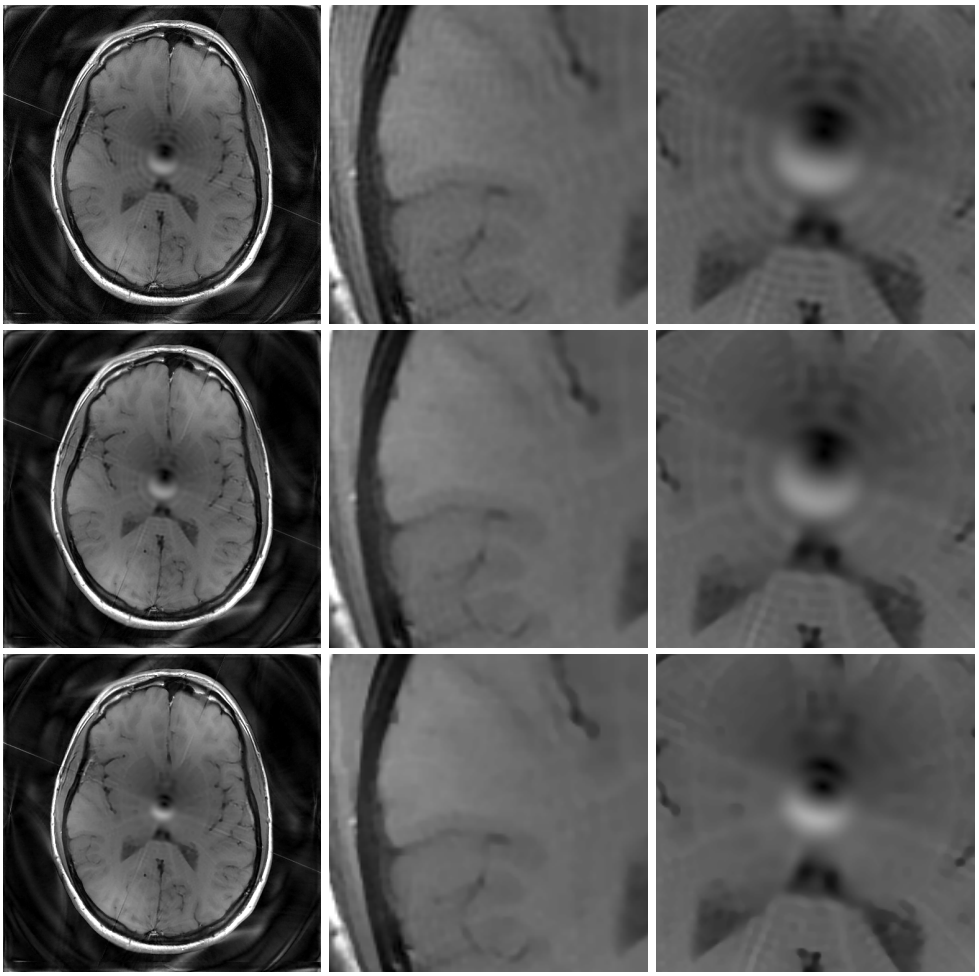


Figure 3.6.: Reconstruction of in-vivo human data acquired with a radial trajectory and a Spin Echo (SE) sequence (Section 2.4). The top row shows the image reconstructed with a conventional CG method (50 iterations) and the PatLoc NUFFT 1+2 operator as well as two detailed views. The rows two and three show the results for TGV and TGV-CG regularized reconstruction, respectively. The regularization parameter has been set to $\alpha = 10^{-6}$ in both cases. TGV required 2000 PD iterations, whereas TGV-CG required 200 PD iterations.

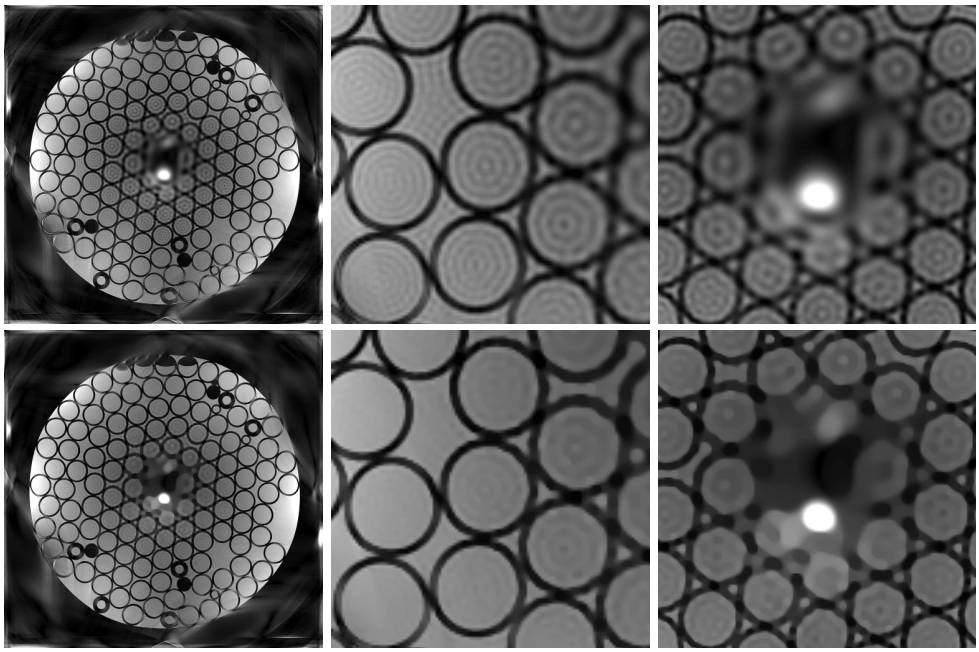


Figure 3.7.: Reconstruction of phantom data acquired with a Cartesian trajectory and a Turbo Spin Echo (TSE) sequence (Section 2.4). The top row shows the image reconstructed with a conventional CG method (50 iterations) and the encoding matrix operator as well as two detailed views. The second row shows the results for the TGV-CG regularized reconstruction. The regularization parameter has been set to $\alpha = 1 \cdot 10^{-1}$. TGV-CG required 200 PD iterations.

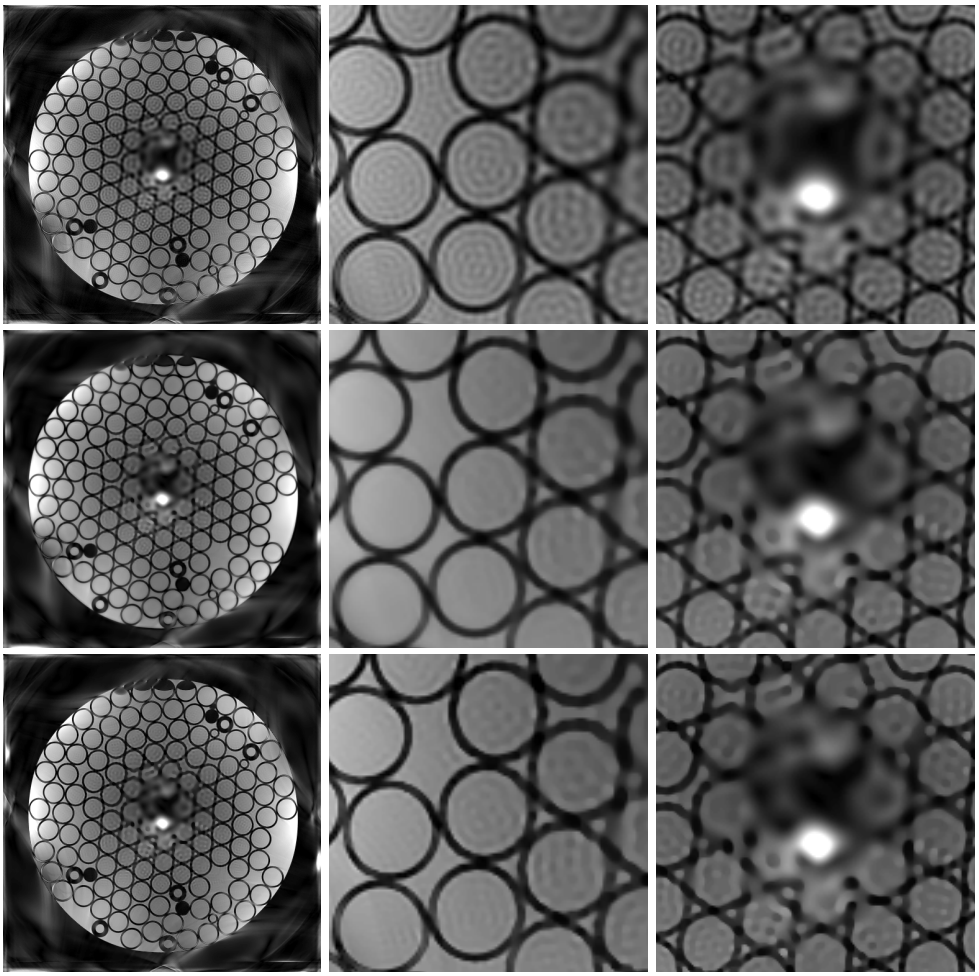
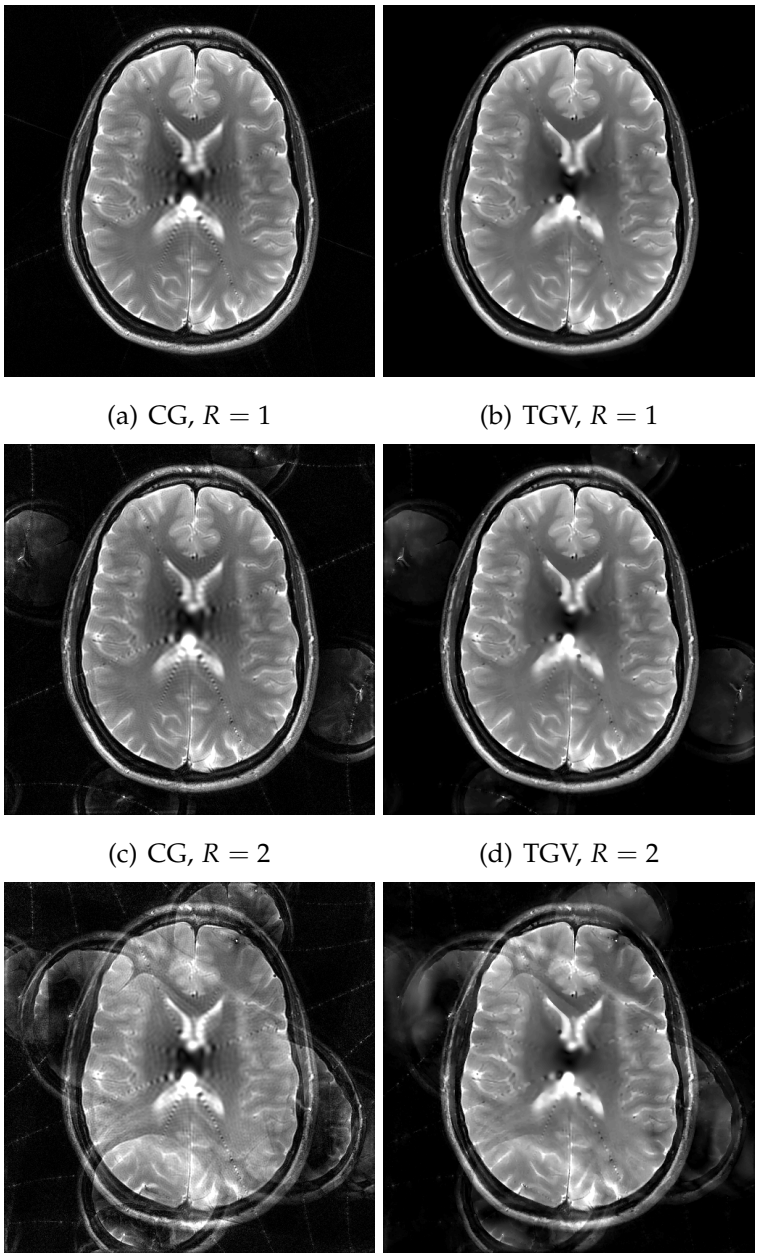


Figure 3.8.: Reconstruction of phantom data acquired with a radial trajectory and an Spin Echo (SE) sequence (Section 2.4). The top row shows the image reconstructed with a conventional CG method (50 iterations) and the PatLoc NUFFT 1+2 operator as well as two detailed views. The rows two and three show the results for TGV and TGV-CG regularized reconstruction, respectively. The regularization parameter has been set to $\alpha = 10^{-6}$ in both cases. TGV required 2000 PD iterations, whereas TGV-CG required 200 PD iterations.

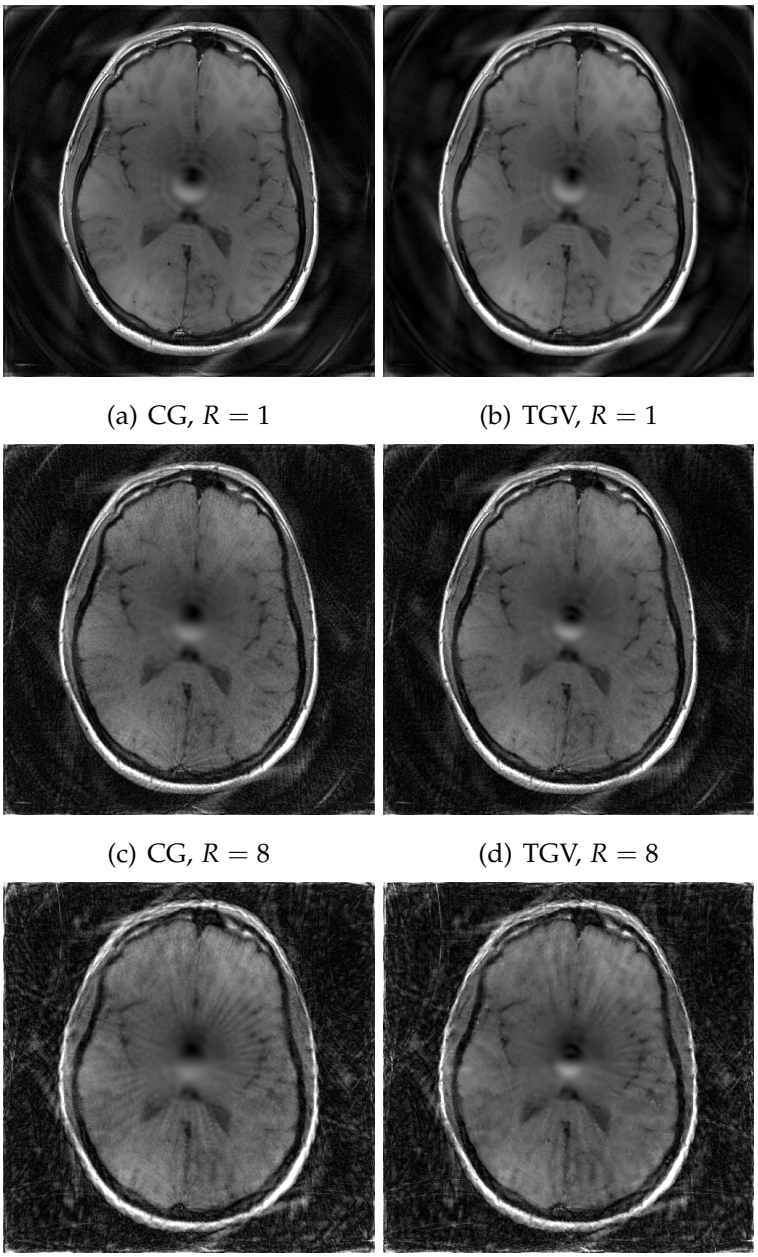


(a) CG, $R = 1$ (b) TGV, $R = 1$

(c) CG, $R = 2$ (d) TGV, $R = 2$

(e) CG, $R = 4$ (f) TGV, $R = 4$

Figure 3.9.: Reconstruction of a dataset acquired with a Cartesian sampling pattern with different levels of undersampling indicated by R.



(a) CG, $R = 1$ (b) TGV, $R = 1$

(c) CG, $R = 8$ (d) TGV, $R = 8$

(e) CG, $R = 16$ (f) TGV, $R = 16$

Figure 3.10.: TODO Reconstruction of a dataset acquired with a radial sampling pattern with different levels of undersampling indicated by R.

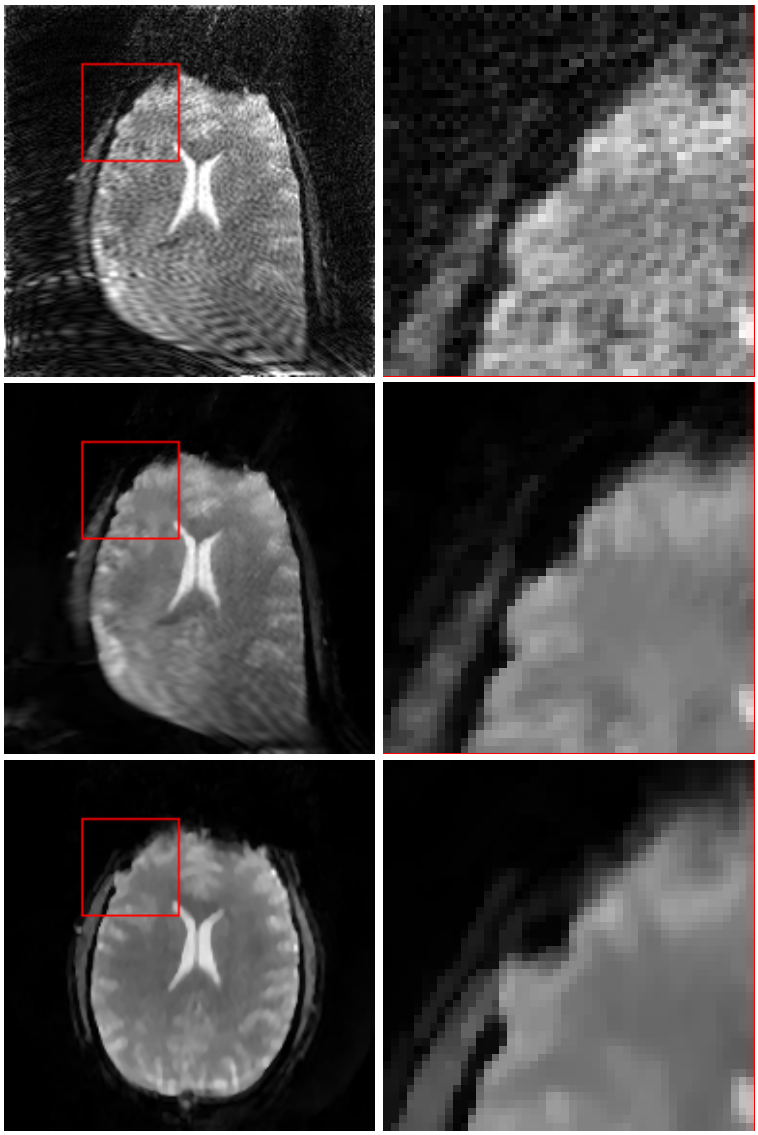
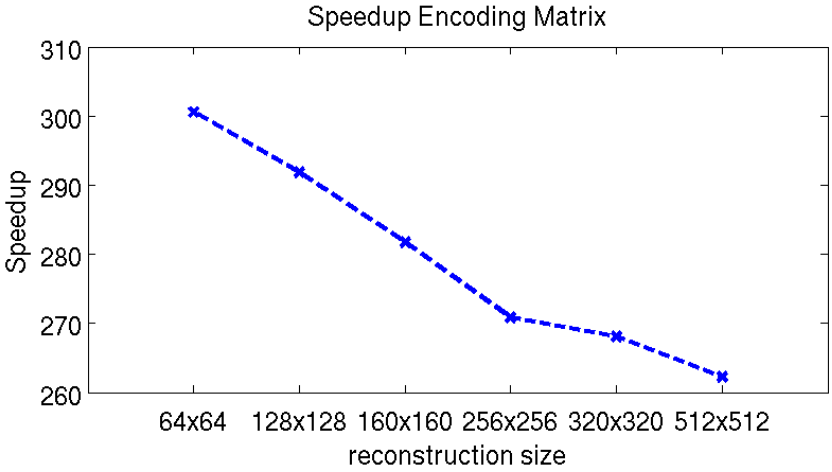
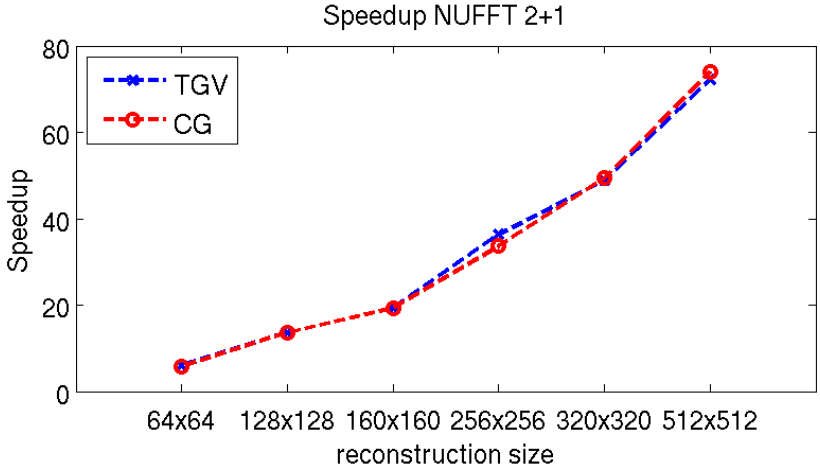


Figure 3.11.: Single-shot NorthWest EPI data (Section 2.4.1) reconstructed onto a 192×192 pixel grid with CG (top row) and TGV-CG (middle row, $\alpha = 1.2$). For comparison an image acquired with a conventional EPI sequence using linear gradients and also reconstructed using TGV-CG ($\alpha = 0.9$) is shown in the bottom row. The first column shows the entire image with the region of interest (area of highest resolution in case of NW-EPI) labeled with a red rectangle whereas the second column shows a zoomed version of the ROI.



(a) Speedup of the encoding matrix operator.



(b) Speedup of the PatLoc NUFFT 1+2 operator.

Figure 3.12.: Speedup of the GPU-accelerated implementation compared to a sequential MATLAB implementation for different reconstruction sizes. The dataset was simulated with a 256×256 k-space grid and eight coils.

	CG	TGV	TGV-CG	
Operator			PD	CG
Encoding Matrix	≈ 50	$>> 60000$	≈ 300	≈ 2200
PatLoc NUFFT 1+2	≈ 50	≈ 2000	≈ 200	≈ 4000

Table 3.1.: Iterations needed for each forward operator and reconstruction method.

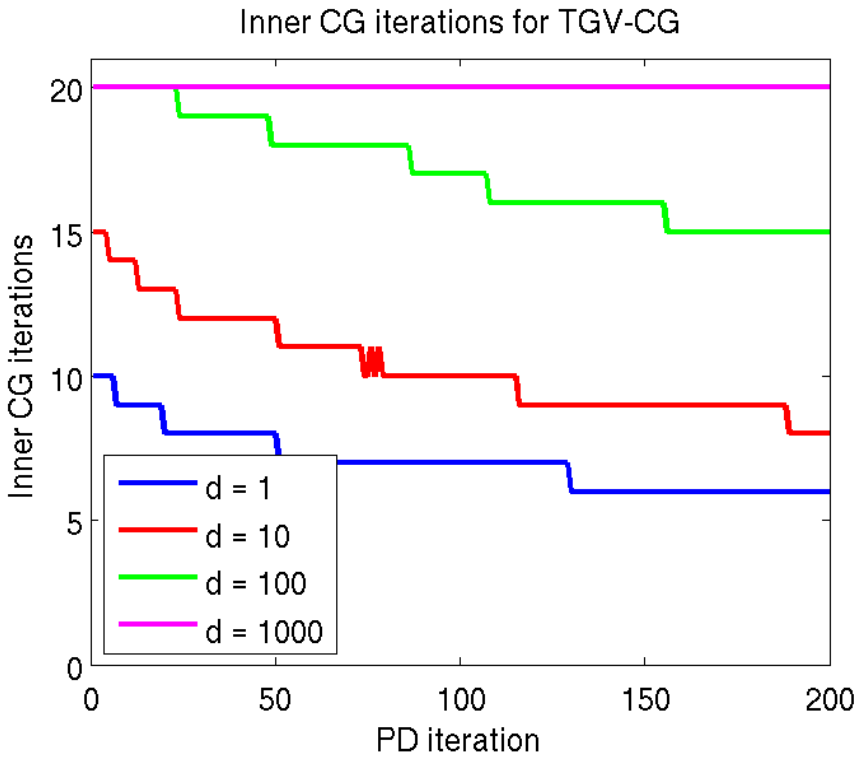


Figure 3.13.: Illustration of the iterations of the inner CG in TGV-CG regularized reconstruction. The factor d indicates the value by which the norm of the operator is divided to speed up convergence.

4. Discussion

Using linear gradients for image encoding in MR imaging has important properties like constant field of view (FOV) and constant image resolution. However, generalizing image encoding to arbitrarily shaped fields like in PatLoc imaging proved to be an interesting concept for new imaging techniques. One challenge of getting non-linear encoding schemes accepted for clinical use is overcoming the computationally demanding reconstruction. One approach to solve this are direct reconstruction concepts which are mostly tailored to a specific trajectory [12, 29]. These concepts are faster, but exhibit stronger artifacts, particularly due to the missing regularization. Other ideas of reducing the complexity of the reconstruction – like when using the PatLoc NUFFT 1+2 operator [11] – often involve interpolation which may not be desirable. The most general way to reconstruct is by brute-force inversion of the encoding matrix with iterative methods. Since this is also the most time-consuming way, a GPU-accelerated implementation is necessary to make the method practical and to eventually get it accepted in clinical practice. The aim of this work was the implementation of GPU-accelerated image reconstruction for PatLoc MR imaging, both regularized (TGV (Section 2.6.2) and TGV-CG (Section 2.6.4)) and not regularized (conjugate gradient (CG) method). This chapter will discuss the findings concerning image quality (Section 4.1), performance of the GPU-accelerated implementation (Section 4.2), the convergence of the implemented techniques (Section 4.3) and finally a conclusion will be drawn (Section 4.4).

4.1. Image Quality

In all of the shown images in Section 3 TGV and TGV-CG led to significantly better image quality compared to the images reconstructed with a conjugate gradient method. Due to reasons explained in Section 4.3, no results for TGV in combination with the encoding matrix are shown.

In case of the Cartesian sampling pattern (Figs. 3.1, 3.2, 3.3 and 3.4) the reconstruction exhibits strong streaking artifacts originating from the center of the image which is in PatLoc imaging the area of lowest resolution. The explanation of these artifacts lies in the *point spread function* (PSF) in the center. These artifacts can also be observed in the images shown in Fig. 1.2, however, due to the reparameterization performed in the reconstruction, the streaking artifacts are split up causing eight streaks instead of four. Further, miscalibration of the gradient fields during imaging is indicated by the bright spot in the center. This could also partially contribute to these artifacts. Nevertheless, in all cases TGV and TGV-CG are capable of reducing the effects of the PSF. Some of the streaks are even removed completely, while others remain still visible, possibly due to the miscalibration. Even the highly corrupted center shows better quality when TGV regularization is performed. However, it is still a region that cannot be trusted as TGV does not increase resolution in this area but rather reduces the effects of this region on the rest of the image. Another positive effect is the removal of Gibbs ringing which can be observed in the second zoom in each of the figures. This effect is particularly well depicted in the phantom images (Figs. 3.3, 3.4) where strong Gibbs ringing can be observed at the edges of the tubes.

In imaging with a radial trajectory (Figs. 3.5, 3.6, 3.7 and 3.8) the effects of the PSF do not result in streaking artifacts. However, the center still re-

sults in ringing artifacts which are reduced by TGV and TGV-CG. Due to the unfortunately low contrast in in-vivo images, the regularization parameter had to be chosen lower than in case of the Cartesian sampling pattern in order to not loose the image features. The phantom images (Figs. 3.7, 3.8) still show strong Gibbs ringing artifacts which can be overcome by TGV regularization.

As expected, the differences in image quality between the reconstructions using the PatLoc NUFFT 1+2 operator and the encoding matrix are minimal.

Even for undersampled data as shown in Fig. 3.9 (Cartesian) and Fig. 3.10 (radial) TGV is capable of reducing the undersampling artifacts, leading to improved image quality compared to reconstructions with the conjugate gradient method. However, for a high undersampling factor of $R = 4$ the undersampling artifacts still corrupt the image (Fig. 3.10(f)).

The single-shot NorthWest EPI sequence exhibits strong artifacts within the object as well as outside. Figure 3.11 shows the image reconstructed with CG, TGV-CG and an image acquired with a conventional EPI sequence using linear gradients. The area of interest is illustrated with a red rectangle. A zoomed view of the area of interest can be seen in the right column. TGV is capable of reducing the artifacts significantly. To illustrate the higher resolution of the NW-EPI sequence a reference image acquired with linear gradients is shown as well.

4.2. Performance

For the performance analysis, the GPU implementation was compared to already existing MATLAB solutions as mentioned in Section 2.7. The hard-

ware used for the evaluation is presented in Section 2.3. In this section only a rough overview of the performance is given, as detailed comparisons tend to compare hardware architectures rather than illustrating the practical benefit of the implementation. Therefore the comparison is limited to one representative dataset which was simulated with a 256×256 k-space grid for eight coils. As depicted in Fig. 3.12, the speedup of the GPU implementation was compared to the given MATLAB implementations for different reconstruction grids. A comparison of the different reconstruction methods was not necessary, as the most time-consuming parts are the application of the forward operator and its adjoint which are the same for all methods.

Figure 3.12(a) displays the results for the encoding matrix as forward operator. It exhibits a negative trend where the practical cases (reconstruction grid ranging from 256×256 to 512×512) correspond to a speedup of approximately 260 to 270. The negative trend can be explained by the fact that due to the size of the encoding matrix and the CUDA related limitations which have to be overcome (Section 2.5.1) the implementation has to be made more and more sequential the bigger the reconstruction size is. Still, the speedup is high enough to make this method practical in the research context as this corresponds to about 27 seconds for one iteration on the GPU compared to approximately 2 hours on the CPU for a reconstruction size of 512×512.

The performance in Fig. 3.12(b) shows a more positive trend for the PatLoc NUFFT 1+2 operator. However, the speedup is much lower than for the encoding matrix which is due to the fact that the implementation mostly consists of FFTs and sparse matrix vector multiplications. For these parts very fast implementations on the CPU exist whereas the available GPU libraries still have room for improvement and are under heavy development. Nevertheless, for bigger reconstruction sizes the GPU implementation shows its

potential as can be seen from the positive trend, such that a speedup of ≈ 75 can be expected. For a reconstruction grid of 512×512 this corresponds to a reconstruction time of ≈ 0.27 seconds on the GPU for one iteration in contrast to ≈ 20 seconds for reconstructing on the CPU. The graph also illustrates the difference between CG and TGV on a per iteration basis. This emphasizes that the differences of different reconstruction methods (taking aside the number of iterations needed) have a minimal impact on the performance.

4.3. Convergence

Because of the similar reconstruction times for different reconstruction methods (Section 4.2) when compared on a per iteration basis, the convergence properties of the methods are the dominating factor in terms of overall speed.

Table 3.1 gives an overview of the number of iterations needed for each reconstruction method applied with one of the discussed forward operators. As can be seen from this table, CG is the fastest method because 50 iterations usually suffice. However, the actual number of iterations might differ according to the needed degree of regularization.

The number of iterations also indicates when TGV-CG should be favoured over TGV. For instance, in case of the encoding matrix operator, TGV does not converge in reasonable time as even after 60000 iterations no convergence can be observed, even for a very conservative regularization parameter α. TGV-CG on the other hand only needs 300 PD iterations. However, one has to consider the iterations of the inner CG method for solving the linear system. This corresponds to approximately 2200 iterations in total.

In case of the PatLoc NUFFT 1+2 operator the TGV-CG method again converges faster (fewer PD iterations), however, taking into account the inner CG method which involves the time-consuming application of the forward operator and its adjoint, it actually needs twice as long as conventional TGV.

The actual number of iterations of the inner CG depends on different factors, for instance the trajectory or the reconstruction size. But most importantly, the factor d used to speed up convergence (Section 2.6.3) plays a major role as illustrated in Fig. 3.13. The lower this factor is set, the faster the inner CG method converges. However, for some datasets it is necessary to speed up convergence with this factor to keep the number of iterations low. This is a tradeoff that has to be considered when choosing these parameters.

4.4. Conclusion

This work has shown that a significant speedup can be achieved with use of a GPU based implementation compared to a sequential algorithm. This even renders methods based on the brute-force inversion of the encoding matrix practical in a research and clinical context. However, in a clinical setting, reconstruction techniques using regularization such as TGV or TGV-CG, which require significantly more iterations than CG, profit from faster discretization methods of the signal equations like the PatLoc NUFFT 1+2 operator. This could even be improved by a direct implementation of the type 3 NUFFT on the GPU. This topic will be investigated in future work. Another topic of future investigations will be the use of more sophisticated solvers for convex-concave saddle point problems which are expected to lead to faster convergence and therefore to a faster TGV regularized reconstruction.

The regularization techniques TGV and TGV-CG exhibit significantly better image quality compared to reconstructions acquired with use of CG. The results in [11] could be confirmed for new in-vivo data. Even for highly undersampled data a good image quality could be obtained.

TGV-CG proved to be a good choice for regularized reconstruction in certain situations. With this new method it is possible to speed up convergence significantly when the encoding matrix is used as forward operator. In this case the convergence can be improved tremendously compared to the conventional primal-dual algorithm (Section 2.6.3). The method has been successfully applied to in-vivo data from a current research project which investigates multi-dimensional single-shot trajectories using non-linear encoding fields and a dynamic field camera (Section 2.4.1) and showed impressive results in terms of speed and image quality.

It offers different levels of complexity by giving the user the ability to adjust all parameters used in reconstruction while also providing sane defaults. This framework can easily be merged into existing scanner software due to its command line interface, which could eventually lead to a one-button solution.

Appendix

Appendix A.

Usage

This section describes the basic usage of the command line interface for the reconstruction algorithms. For a more detailed explanation, especially on installation, the reader is referred to the `README` file provided with the source code. All possible parameters of the command line interface can be seen in the help page (`plcli --help`).

The data needs to be passed to the algorithm in form of a MATLAB file which consists of the following struct:

- `S.a_res`: number of readout points
- `S.b_res`: number of phase encoding steps
- `S.nC`: number of coils
- `S.reconSize`: reconstruction size
- `S.numSEM`: number of SEMs
- `S.SEM`: stack of encoding fields
- `S.SEM_pat_a`: nonlinear encoding field A (NUFFT 1+2 only)
- `S.SEM_pat_b`: nonlinear encoding field B (NUFFT 1+2 only)
- `S.k`: sampling trajectory

- `S.traj_pat_a`: trajectory for nonlinear encoding field A (NUFFT 1+2 only)
- `S.traj_pat_b`: trajectory for nonlinear encoding field B (NUFFT 1+2 only)
- `S.Cmat`: RF coil sensitivities
- `S.recondata`: measured data

First the user has to choose one of the following reconstruction methods:

- `--cg`: CG method using the encoding matrix as forward operator
- `--nufftcg`: CG method using the PatLoc NUFFT 1+2 forward operator
- `--tgv`: TGV regularized reconstruction using the encoding matrix
- `--tgvnufft`: TGV regularized reconstruction using the PatLoc NUFFT 1+2 operator
- `--tgvcg`: TGV-CG reconstruction using the encoding matrix
- `--tgvnufftcg`: TGV-CG reconstruction using the PatLoc NUFFT 1+2 operator

Other important parameters are:

- `-i, --iters`: number of iterations
- `-ii, --inner-iters`: number of iterations of the inner CG (TGV-CG only)
- `-a, --alpha`: regularization parameter (TGV and TGV-CG only)
- `-r, --reconsize`: reconstruction size (if provided, value passed in data file is ignored)
- `--norm-div`: factor to speed up convergence
- `-o, --out`: output directory (defaults to `results` if not provided)
- `-si, --save-images`: save all intermediate images
- `-sm, --save-matlab`: save all intermediate results as `.mat` files

A.1. Examples

Here, some examples are shown to illustrate the usage of the command line interface.

1. Reconstructing a dataset `data.mat` on an image grid of the size 512×512 with the conjugate gradient method (60 iterations). Save all intermediate images:

   ```
   plcli --cg -r 512 -i 60 -si data.mat
   ```

2. Reconstructing a dataset `data.mat` with TGV (PatLoc NUFFT 1+2 operator), reconstruction size of 256, 2000 primal-dual iterations, speed up convergence with a `norm-div` parameter of 1000, regularization parameter `alpha` of 10^{-4}.

   ```
   plcli --tgvnufft -r 256 -i 2000 --norm-div 1000 -a 1e-4 data.mat
   ```

3. Same as 2., but using TGV-CG instead of TGV and without `norm-div` parameter. In each primal-dual iteration 20 CG iterations should be performed.

   ```
   plcli --tgvnufftcg -r 256 -i 2000 -ii 20 -a 1e-4 data.mat
   ```

Bibliography

[1] J.P. Stockmann, P.A. Ciris, G. Galiana, L. Tam, and R.T. Constable. "O-space imaging: Highly efficient parallel imaging using second-order nonlinear fields as encoding gradients with no phase encoding." In: *Magn. Reson. Med.* 64.2 (2010), pp. 447–456 (cit. on p. 1).

[2] J. Hennig, A. M. Welz, G. Schultz, J. Korvink, Z. Liu, O. Speck, and M. Zaitsev. "Parallel imaging in non-bijective, curvilinear magnetic field gradients: a concept study." In: *Magn. Reson. Mater. Phy.* 21 (2008), pp. 5–14 (cit. on p. 1).

[3] D. Gallichan, C. A. Cocosco, A. Dewdney, G. Schultz, A. Welz, J. Hennig, and M. Zaitsev. "Simultaneously driven linear and nonlinear spatial encoding fields in MRI." In: *Magn. Reson. Med.* 65.3 (2011), pp. 702–714 (cit. on pp. 2, 5, 7, 8).

[4] K. P. Pruessmann, M. Weiger, M. B. Scheidegger, and P. Boesiger. "SENSE: Sensitivity encoding for fast MRI." In: *Magn. Reson. Med.* 42.5 (1999), pp. 952–962 (cit. on pp. 1, 15).

[5] G. Schultz. "Magnetic Resonance Imaging with Nonlinear Gradient Fields: Signal Encoding and Image Reconstruction." PhD thesis. University of Freiburg, Germany, 2012 (cit. on pp. 3, 5, 9).

[6] M. R. Hestenes and E. Stiefel. "Methods of Conjugate Gradients for Solving Linear Systems." In: *J. Res. Nat. Bur. Stand.* 49.6 (1952), pp. 409–436 (cit. on pp. 3, 20).

[7] K. Bredies, K. Kunisch, and T. Pock. "Total generalized variation." In: *SIAM J. Imag. Sci.* 3.3 (2010), pp. 492–526 (cit. on pp. 3, 31).

[8] F. Knoll, K. Bredies, T. Pock, and R. Stollberger. "Second Order Total Generalized Variation (TGV) for MRI." In: *Magn. Reson. Med.* 65 (2011), pp. 480–491 (cit. on pp. 3, 31).

[9] K. Bredies. "Recovering piecewise smooth multichannel images by minimization of convex functionals with total generalized variation." In: *submitted for publication* (2012) (cit. on pp. 3, 32).

[10] F. Knoll. "Constrained MR Image Reconstruction of Undersampled Data from Multiple Coils." PhD thesis. Graz University of Technology, Austria, 2011 (cit. on pp. 3, 31).

[11] F. Knoll, G. Schultz, K. Bredies, D. Gallichan, M. Zaitsev, J. Hennig, and R. Stollberger. "Reconstruction of undersampled radial PatLoc imaging using total generalized variation." In: *Magn. Reson. Med.* (2012) (cit. on pp. 3, 17, 18, 31, 32, 34, 53, 59).

[12] G. Schultz, P. Ullmann, H. Lehr, A. M. Welz, J. Hennig, and M. Zaitsev. "Reconstruction of MRI data encoded with arbitrarily shaped, curvilinear, nonbijective magnetic fields." In: *Magn. Reson. Med.* 64.5 (2010), pp. 1390–1403 (cit. on pp. 5, 20, 53).

[13] K. J. Layton, D. Gallichan, F. Testud, C. A. Cocosco, A. M. Welz, C. Barmet, K. Pruessmann, and M. Zaitsev. "Region-specific trajectory design for single-shot imaging using linear and nonlinear magnetic encoding fields." In: *Proc. Intl. Soc. Mag. Reson. Med. 2012*. Melbourne, Australia, 2012 (cit. on pp. 6, 8, 14).

[14] S. Kroboth, D. Gallichan, F. Knoll, C. A. Cocosco, G. Schultz, and M. Zaitsev. "Effect of miscalibrations of gradient fields and coil sensitivities in PatLoc imaging." In: *ESMRMB Congress 2012*. Lisbon, Portugal, 2012 (cit. on p. 8).

[15] NVIDIA Corporation. *CUDA Toolkit 4.2*. URL: www.nvidia.com/getcuda (cit. on pp. 10–12, 19).

[16] Python Software Foundation. *Python programming language 2.6*. URL: www.python.org (cit. on pp. 11, 12).

[17] The SciPy Community. *Scientific tools for Python*. URL: www.scipy.org (cit. on pp. 11, 12).

[18] The SciPy Community. *Numerical Python*. URL: numpy.scipy.org (cit. on pp. 11, 12).

[19] A. Klöckner. *PyCUDA 2012.1*. URL: mathema.tician.de/software/pycuda (cit. on pp. 11, 12).

[20] L. Givon. *CUDA SciKit*. URL: lebedov.github.com/scikits.cuda/ (cit. on pp. 12, 19).

[21] B. Opanchuk. *FFT for PyCUDA and PyOpenCL*. URL: packages.python.org/pyfft (cit. on pp. 12, 19).

[22] K.J. Layton, D. Gallichan, Testud. F., C.A. Cocosco, A.M. Welz, C. Barmet, K.P. Pruessmann, J. Hennig, and Zaitsev M. "Single shot trajectory design for region-specific imaging using linear and nonlinear magnetic encoding fields." In: *Magn. Reson. Med.* (2012). DOI: 10.1002/mrm.24494 (cit. on p. 13).

[23] B.J. Wilm, C. Barmet, M. Pavan, and K.P. Pruessmann. "Higher order reconstruction for MRI in the presence of spatiotemporal field perturbations." In: *Magn. Reson. Med.* 65.6 (2011), pp. 1690–1701 (cit. on p. 13).

[24] S. Kroboth, F. Testud, K. Bredies, K.J. Layton, D. Gallichan, C.A. Cocosco, G. Schultz, F. Knoll, C. Barmet, K.P. Pruessmann, M. Zaitsev, and R. Stollberger. "Image Reconstruction of Single-Shot North West EPI Data acquired with PatLoc Gradients using Magnetic Field Monitoring and Total Generalized Variation - Conjugate Gradient." In: *submitted to ISMRM* (2013) (cit. on p. 14).

[25] K.P. Pruessmann, M. Weiger, P. Börnert, and P. Boesiger. "Advances in sensitivity encoding with arbitrary k-space trajectories." In: *Magn. Reson. Med.* 46.4 (2001), pp. 638–651 (cit. on p. 15).

[26] J. Fessler and B. Sutton. "Nonuniform Fast Fourier Transforms Using Min-Max Interpolation." In: *IEEE Trans. Signal Process.* 51.2 (2003), pp. 560–574 (cit. on pp. 16, 18).

[27] L. Rudin, S. Osher, and E. Fatemi. "Nonlinear total variation based noise removal algorithms." In: *Physica D* 60 (1992), pp. 259–268 (cit. on p. 30).

[28] A. Chambolle and T. Pock. "A first-oder primal-dual algorithm for convex problems with applications to imaging." In: *J. Math. Imaging Vision* 40 (2011), pp. 120–145 (cit. on p. 32).

[29] G. Schultz, H. Weber, D. Gallichan, W.R.T. Witschey, A.M. Welz, C.A. Cocosco, J. Hennig, and M. Zaitsev. "Radial Imaging with Multipolar Magnetic Encoding Fields." In: *IEEE Trans. Med. Imag.* 30 (2011), pp. 2134–2145 (cit. on p. 53).