NeuroImage 202 (2019) 116137

Contents lists available at ScienceDirect

NeuroImage

journal homepage: www.elsevier.com/locate/neuroimage

# *MRtrix3*: A fast, flexible and open software framework for medical image processing and visualisation

J-Donald Tournier[a,b,*], Robert Smith[c,d], David Raffelt[c], Rami Tabbara[c], Thijs Dhollander[c,d], Maximilian Pietsch[a,b], Daan Christiaens[a,b], Ben Jeurissen[e], Chun-Hung Yeh[c,d], Alan Connelly[c,d]

[a] *Centre for the Developing Brain, School of Biomedical Engineering & Imaging Sciences, King's College London, London, United Kingdom*
[b] *Department of Biomedical Engineering, School of Biomedical Engineering & Imaging Sciences, King's College London, London, United Kingdom*
[c] *The Florey Institute of Neuroscience and Mental Health, Melbourne, Australia*
[d] *The Florey Department of Neuroscience and Mental Health, University of Melbourne, Melbourne, Australia*
[e] *Imec-Vision Lab, Department of Physics, University of Antwerp, Antwerp, Belgium*

ARTICLE INFO

*Keywords:*
MRI
Software
Image
Processing
Visualisation

ABSTRACT

*MRtrix3* is an open-source, cross-platform software package for medical image processing, analysis and visualisation, with a particular emphasis on the investigation of the brain using diffusion MRI. It is implemented using a fast, modular and flexible general-purpose code framework for image data access and manipulation, enabling efficient development of new applications, whilst retaining high computational performance and a consistent command-line interface between applications. In this article, we provide a high-level overview of the features of the *MRtrix3* framework and general-purpose image processing applications provided with the software.

## 1. Introduction

The use of medical imaging technologies such as Magnetic Resonance Imaging (MRI) has rapidly expanded over the last decades, sparking the development of dedicated digital image analysis techniques tailored to these often large imaging datasets (3 and even higher dimensional) (Dhawan, 2011). These methods include artefact correction, anatomy segmentation, quantitative feature extraction, and spatial image registration which allows for comparison of features in subjects over time, between (groups of) subjects and across populations. Development of medical image analysis techniques is largely driven by the academic research community, and many such packages are now freely available, including: FSL (Jenkinson et al., 2012), FreeSurfer (Fischl, 2012), SPM (Ashburner, 2012), AFNI (Cox, 1996), DTI Studio (Jiang et al., 2006), DSI Studio ("DSI Studio" 2019), Camino (Cook et al., 2006), ExploreDTI (Leemans et al., 2009), MITK (Nolden et al., 2013), ITK (Yoo et al., 2002), ANTs (Avants et al., 2009), MIRTK ("MIRTK" 2019), DiPy (Garyfallidis et al., 2014), and many others. Such open software platforms have the potential to enable the translation of novel method developments to, e.g., MRI hardware vendors, clinicians and the biomedical research community.

*MRtrix3* has been designed as an open-source, modular software platform for medical image analysis and visualisation. This includes a lightweight framework for application development that facilitates efficient image access and manipulation, convenient multi-threading primitives, and support for a broad range of image file formats. End-user applications provided with *MRtrix3* include general-purpose image conversion and manipulation tools and the visualisation tool *MRView* tailored to multi-dimensional (3D+) medical image data. All components are designed with a strong focus on performance and memory efficiency, which are essential features for dealing with these often very large datasets. Computationally demanding quantitative imaging algorithms often require run times on the order of minutes (or hours), which may easily become days (or weeks) if performance is not optimised to deal with this specifically. In addition, *MRtrix3* offers support for external modules that can be built and distributed independently.

*MRtrix3* was historically developed with a particular emphasis on investigating brain white matter using diffusion-weighted MRI (dMRI), a medical imaging modality that can act as an indirect probe of neural microstructure, by measuring local hindrance of water diffusion (Johansen-Berg and Behrens, 2013; Jones, 2010). It derives from the now deprecated MRtrix 0.2.x software package (Tournier et al., 2012), which was itself started circa 2003, when few packages existed for the

* Corresponding author. Centre for the Developing Brain, School of Biomedical Engineering & Imaging Sciences, King's College London, London, United Kingdom.
*E-mail address:* jacques-donald.tournier@kcl.ac.uk (J.-D. Tournier).

https://doi.org/10.1016/j.neuroimage.2019.116137
Received 21 January 2019; Received in revised form 5 August 2019; Accepted 27 August 2019
Available online 29 August 2019

analysis of diffusion MRI data. It has evolved massively since, both in terms of the underlying codebase, and in terms of the scope of the functionality provided. The platform incorporates a number of features tailored to the orientational nature of dMRI data within each imaging voxel, such as a representation using spherical harmonics basis functions, and a bespoke file format for 3D streamlines. *MRtrix3* also provides a wide range of dMRI analysis methods, outlined in the *Diffusion analyses* section below. Thanks to these features, *MRtrix3* has primarily gained support in the dMRI community. However, its architecture facilitates image processing in general, and *MRtrix3* also offers image analysis methods less specific to dMRI such as image filtering and calculation, statistics, image denoising (Veraart et al., 2016) and Gibbs ringing suppression (Kellner et al., 2016), intensity normalisation (Raffelt et al., 2017a) and diffeomorphic image registration (Raffelt et al., 2011).

In this paper, we describe the guiding principles underlying the design and implementation of *MRtrix3*. We outline the main aspects of the software architecture, particularly the image access and multi-threading primitives, as well as the build system and related developer tools. Finally, we showcase the visualisation capabilities in *MRView* and give a brief overview of the currently supported tools for general-purpose image processing and dMRI analysis.

## 2. Guiding principles

### 2.1. Reproducible neuroscience

With a growing number of neuroimaging studies and the increasing prevalence of advanced data analysis in radiology practice, reproducibility becomes ever more important. Publishing open and reproducible analysis pipelines enables neuroscientific research to replicate pilot results on larger datasets, and also facilitates comparing sensitivity and specificity of various methods. In addition, publicly available software can help the clinical translation of novel analysis techniques.

The *MRtrix3* collective believes that reproducible neuroscience relies first and foremost on open source software. Publishing source code, tracking versions, and documenting improvements and bug fixes, is vital to ensure full transparency on what was done in a study. We are therefore fully committed to keeping *MRtrix3* open source and tracking the version history using Git. The software is published under the terms of the open source Mozilla Public Licence 2.0,[1] which permits third-party modification, distribution and commercial use as long as the source code is disclosed (although individual commands may be published under different copyright licences as required by the original authors). Furthermore, *MRtrix3* also incorporates continuous integration testing to detect and avoid adverse effects of code changes between versions.

In addition, *MRtrix3* provides the requisite functionalities for handling data in the Brain Imaging Data Structure (BIDS),[2] a recently proposed standard for organising and sharing neuroimaging data (Gorgolewski et al., 2016). These include importing and exporting dMRI gradient encoding and other metadata in JSON and *FSL* bvecs / bvals format, ultimately enabling integrating *MRtrix3* commands in BIDS Apps (Gorgolewski et al., 2017).

Finally, *MRtrix3* also offers automatic history tracking in the output file header, archiving the pipeline of operations used to generate each output image, as well as the exact version tag of each invoked command.

[1] http://mozilla.org/MPL/2.0/.
[2] http://bids.neuroimaging.io.

### 2.2. Documentation & support

User documentation is available in the inline help page of each command and also on a dedicated documentation website[3] that includes additional background information, tutorials and pipelines. The documentation is kept up to date with the current release and can be retrieved for older versions. Major updates and new software releases are also announced on the main website and blog.[4]

Another key benefit to open development is the interaction between users and developers as one community. To foster these interactions *MRtrix3* provides an online forum,[5] where a growing number of users post questions, receive support, and have scientific discussions. Furthermore, users and developers can report and track technical issues through Github,[6] and suggest new features for future releases. Github is also the platform where both internal and external developer contributions are discussed and managed.

## 3. Design aspects

*MRtrix3* has been designed from the outset to facilitate the development of high-performance applications concerned with the processing and analysis of medical image datasets, and to offer a consistent and simple command-line interface. It focuses primarily on command-line applications to provide a wide range of functionalities that can be combined and scripted to form complex automated workflows. It relies on modern standards and technologies, particularly C++11[7] and *OpenGL*[8] 3.3, yet strives to limit the number of external dependencies to a small and well-supported set, available across a broad range of platforms; currently these include Python[9] (for building the software and using scripts), *zlib*[10] (for access to compressed images), *Eigen 3*[11] (for high-performance linear algebra support), and *Qt*[12] (version 4 or 5, for graphical user interface support). All of these dependencies are available across a wide range of platforms, allowing *MRtrix3* to run natively on most flavours of Unix, including GNU/Linux and macOS as well as a range of high-performance computing (HPC) environments, and Microsoft Windows via the *MSYS2*[13] project or Windows Subsystem for Linux (*WSL*).[14]

Most of the functionality is written using C++ templates, providing high runtime performance by allowing the compiler to optimise away any redundant overhead. Motivated by a heavy focus on multidimensional image access, simple yet powerful constructs for multi-threaded operations are provided. The framework provides native access to and seamless interaction between a range of image file formats commonly used in the medical imaging community, including native support for DICOM[15] (read-only), Analyze,[16] NIfTI1,[17] NIfTI2,[18] MGH,[19] and *MRtrix3*'s own dedicated image formats[20], as well as compressed

[3] https://mrtrix.readthedocs.io.
[4] http://www.mrtrix.org.
[5] http://community.mrtrix.org.
[6] https://github.com/mrtrix3.
[7] http://www.open-std.org/jtc1/sc22/wg21/docs/standards.
[8] https://www.opengl.org/.
[9] https://www.python.org/.
[10] https://zlib.net/.
[11] http://eigen.tuxfamily.org/.
[12] https://www.qt.io/.
[13] https://www.msys2.org/.
[14] https://docs.microsoft.com/en-us/windows/wsl/about.
[15] https://www.dicomstandard.org/.
[16] https://rportal.mayo.edu/bir/ANALYZE75.pdf.
[17] https://nifti.nimh.nih.gov/nifti-1/.
[18] https://nifti.nimh.nih.gov/nifti-2/.
[19] https://surfer.nmr.mgh.harvard.edu/fswiki/FsTutorial/MghFormat.
[20] https://mrtrix.readthedocs.io/en/3.0_rc3/getting_started/image_data.html#mrtrix-image-formats-mih-mif.

versions thereof.

*MRtrix* introduces a few additional file formats to handle various types of data, notably image data, and streamlines data. These *MRtrix*-specific formats were designed to address limitations in the formats available at the time and provide additional benefits including:

- Ability to store arbitrary key-value entries in the header, which enables storage of diffusion gradient and phase encoding schemes or any other metadata deemed of interest, avoiding a potential source of error due to mixing data stored across multiple files (e.g. bvecs/bvals files);
- Tracking of the command history;
- Ability to store data with arbitrary strides (discussed below and in Appendix A);
- Streamlines data stored in scanner space, removing any dependencies on external reference images.

We note that the benefits listed above are still relevant today, and for this reason, we still advocate the use of these formats to our users. Both formats are documented with reference MATLAB[21] code available for use. *MRtrix3* also provides the *tckconvert* conversion tool to import & export streamlines data between formats.

While most *MRtrix3* commands consist of compiled binary executables, some are written in Python, and typically invoke other *MRtrix3* executables to perform more complex tasks. Moreover, a number of scripts invoke commands from 3rd party software packages such as FSL (Jenkinson et al., 2012) or ANTs (Avants et al., 2009), providing convenience wrappers to perform specialised tasks (see the *Image Processing tools and analyses* section below for details).

## 3.1. Image access

Functionality for image input/output is provided by the core *MRtrix3* library, so that all *MRtrix3* applications can read and write all supported image formats directly: there is no requirement to convert the data to and from specific image formats. All *MRtrix3* applications consistently use the same coordinate system, which is identical to the scanner reference frame in the NIfTI standard. *MRtrix3* handles coordinates consistently across file formats and in addition, the native *MRtrix3* image format allows automated handling and storage of coordinate system dependent information such as the dMRI gradient table in the image header to ensure this information remains consistent throughout the analysis pipeline.

We do however note that while *MRtrix3* supports these formats, there can still be issues regarding interoperability between software packages, relating primarily to the conventions assumed regarding the *interpretation* of the information stored, for example: orientations, tensor components, spherical harmonics coefficients, and other types of information (e.g. what coordinate system is assumed when storing eigenvectors, etc.). Until common standards are agreed by the community, users will need to handle any issues arising from these differences manually.

### 3.1.1. Optimised data access

Access to image data is automatically optimised using a variety of techniques to ensure near-optimal performance across a range of operating conditions. The various strategies employed are outlined in this section.

By default, image access is achieved via memory-mapping with on-the-fly data type conversion. However, when the stored data type matches the data type requested by the application, *MRtrix3* commands will automatically use “direct IO” access, where data are accessed as a native array in memory, thus avoiding the overhead of the function calls otherwise necessary for the conversion. When desired, applications can also explicitly request “direct IO” access to the data (e.g. as shown in Fig. 1): this is particularly beneficial for applications that will access image data repeatedly, for example, multi-pass algorithms, patch-based operations, interpolation, etc. To achieve this, the data are loaded directly into RAM in the desired data type, unless they are already stored as such (in which case memory-mapping already provides “direct IO” access). Furthermore, while image writing is also typically done using memory mapping, there are cases where doing so incurs a loss of performance. This is particularly relevant when operating on networked file systems where random writes to memory-mapped regions can result in heavy network usage. In this case, *MRtrix3* uses an alternative *delayed write-back* strategy, with the backend holding the data in RAM and automatically committing the data to storage when the image is closed.

*MRtrix3* also has a flexible concept of *strides*, allowing the multi-dimensional image data to be stored in RAM (or file when the image format supports it) in any reasonable order (see Appendix A for details). It is for example entirely possible to have the data stored in a volume-contiguous manner (for 4D files), meaning that the data for all volumes of a given imaging voxel are co-located in RAM (see Fig. A1). This allows for efficient access and maximises CPU cache reuse, in some cases improving performance considerably. Developers can specify these strides when accessing images with a particular purpose in mind (e.g., voxel-wise processing of 4D files), and the *MRtrix3* backend will load and re-order the image data in RAM if necessary, but still use “direct IO” access via memory-mapping where possible to avoid needless data copies. Images can be stored in any stride ordering supported by the chosen file format. Of the supported formats, only *MRtrix3*'s own formats support arbitrary stride ordering, a feature that offers distinct performance advantages when processing different types of data, particularly by allowing volume-contiguous storage.

### 3.1.2. DICOM handling

*MRtrix3* includes its own fast DICOM import handling, allowing all *MRtrix3* applications to seamlessly support DICOM images as input. The DICOM import code handles data stored using the recent DICOM multi-frame standard, as well as the standard multi-file storage, including Siemens mosaics. If present, it will also extract the diffusion MRI encoding information for 3 major vendors (Siemens, GE, Philips), and make it available to the application as part of the image header. Other types of information are also extracted if found, including the EPI phase encoding direction and slice timings.

DICOM images can be provided as a single .dcm file, or more commonly, as a folder containing the multiple DICOM files that make up the full DICOM series. The import code will scan through all files in the folder, recursing into any subfolders encountered, and collate all the information into a hierarchy by patient/study/series/images. If the folder contains a single DICOM series, the data are loaded as a single dataset directly with no further interaction required; otherwise, the user will be presented with a listing of the contents and required to select the desired series.

## 3.2. Performance and piping

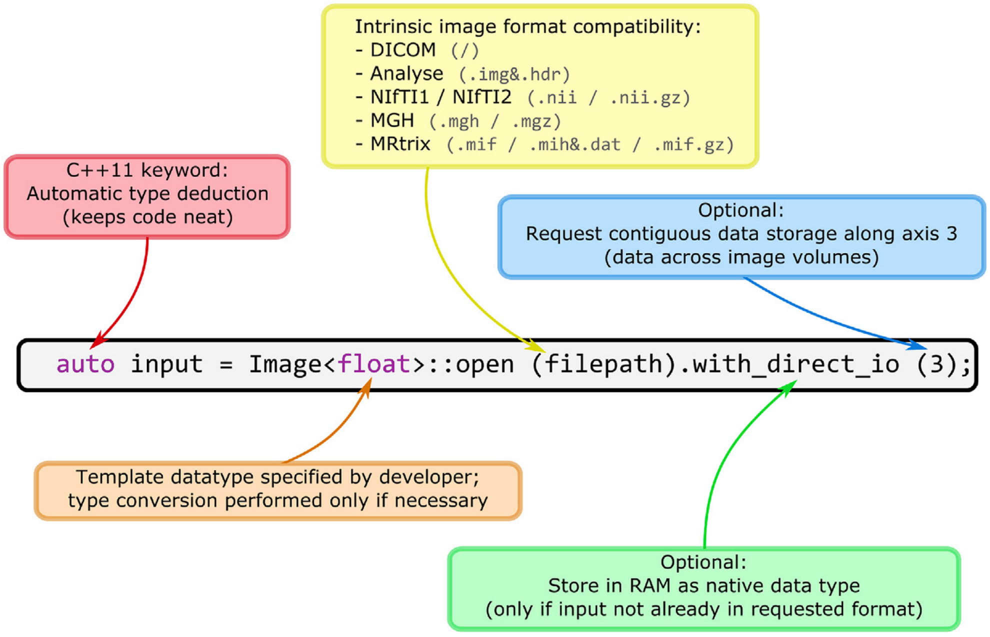
### 3.2.1. Unix pipes

*MRtrix3* allows temporary images to be created in RAM for use as scratch buffers within applications and also passed between applications via Unix pipes. This allows the software to be used in flexible, powerful combinations, for example:

[21] MATLAB, The MathWorks, Natick, https://mathworks.com/.

```
$ mrthreshold input.mif -abs 0.5 - | maskfilter - erode - | mrview input.mif -overlay.load -
```

Listing 1: illustration of the use of Unix pipes in *MRtrix*. The command above will threshold image "input.mif" at an absolute value of 0.5; erode the resulting binary mask by one voxel; visualise the original input image, with the mask calculated in the previous step overlaid on top



```
auto input = Image<float>::open (filepath).with_direct_io (3);
```

Intrinsic image format compatibility:
- DICOM (/)
- Analyse (.img&.hdr)
- NIfTI1 / NIfTI2 (.nii / .nii.gz)
- MGH (.mgh / .mgz)
- MRtrix (.mif / .mih&.dat / .mif.gz)
C++11 keyword: Automatic type deduction (keeps code neat)
Optional: Request contiguous data storage along axis 3 (data across image volumes)
Template datatype specified by developer; type conversion performed only if necessary
Optional: Store in RAM as native data type (only if input not already in requested format)

**Fig. 1.** Demonstration of features provided within a single line of code when using the *MRtrix3* software library to access an image from the filesystem.

Rather than passing the full image data through the pipe, temporary images are created to hold the data, and only the filename is passed through the pipe. This helps to ensure optimal performance since the data can be accessed via memory-mapping at both ends of the pipe, eliminating any needless data copies, and keeping RAM usage to a minimum.

### 3.2.2. *Multi-threading*

Multi-threading is a type of computer execution technique supported by modern hardware that allows a program to run across multiple threads of execution concurrently. Each thread executes independently while sharing its resources and coordinating with the other threads in the process. In *MRtrix3*, almost all commands have been multi-threaded, by default running with the maximum utilization of CPU processors to achieve high computing performance.

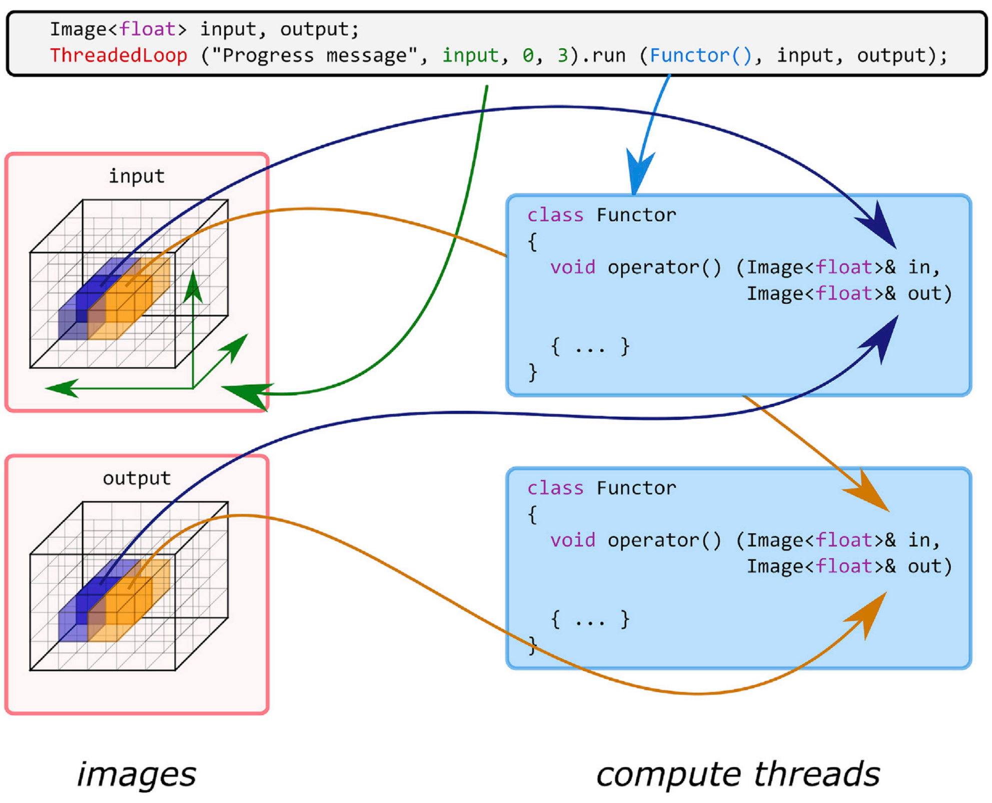
In *MRtrix3*, multi-threaded processing is principally achieved through the use of one of two Application Programming Interfaces (APIs):

1. *Image processing*: When subsets (e.g. voxels, slices, volumes) of an image (or set of images) can be processed independently, the ThreadedLoop construct performs a multi-threaded traversal of the image(s), with the only requisite that the programmer define the function to be executed for each image location; this is shown in Fig. 2 and utilised in Appendix B. Concurrent threads will typically process adjacent rows of data; this is done to ensure optimal performance by maximising cache reuse, minimising disk seek latency, and maintaining a sufficiently fine level of granularity to avoid idle CPU cores. By default, the order of traversal of the image axes is set by the strides of the image, although this can be specified explicitly by the developer if desired. Moreover, the developer can control which images are to be looped over simultaneously, which axes to iterate over, and of these, which are to be processed within each thread (a single row by default).
2. *General processing*: For tasks not native to an image grid, the more general run_queue () construct enables multi-threaded processing through the use of data queue structures; an example of this is shown in Fig. 3. Classes defined by the programmer read and/or write data from these queues via functors, with the requisite mutual exclusion locking between threads handled by the back-end of the API. The type(s) of data passed between these threads are also templated and therefore entirely under programmer control. For smaller data/processing tasks, instances of these data can be aggregated in batches so as to mitigate the overhead of inter-thread communication.

### 3.2.3. *Use in high-performance computing environments*

*MRtrix3* can run in HPC environments: it can be installed either system-wide by system administrators, or into individual user accounts by users themselves, and provides control over aspects such as the number of threads and temporary file locations at the system-wide, user-specific and command-specific level. To ease installation on older HPC systems with an outdated compiler or other dependencies, *MRtrix3* provides the option to compile static builds that can be produced on a separate, up to date workstation, and transferred across to HPC once compiled.

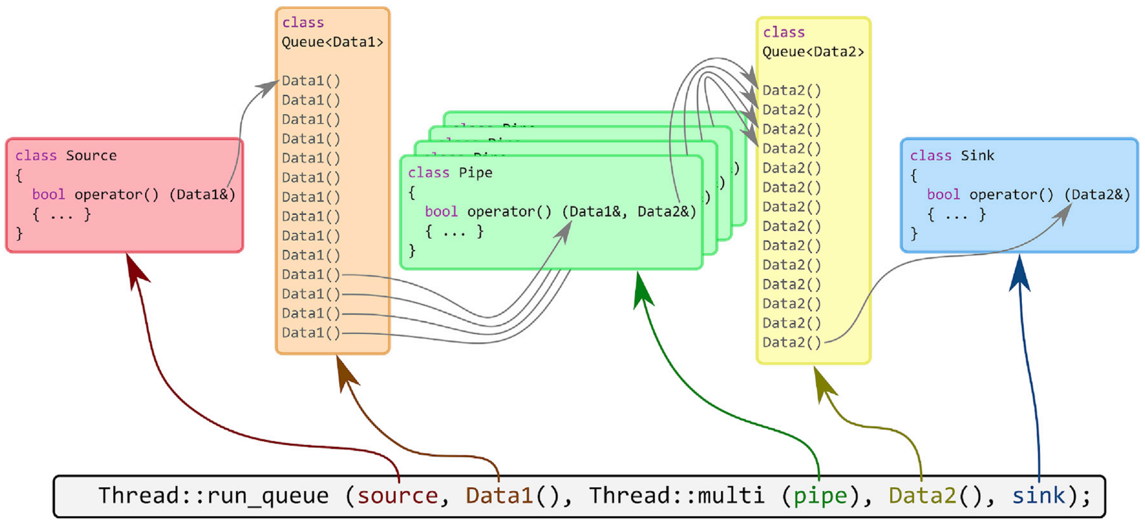
While support for virtualization via containers is outside the scope of


```
Image<float> input, output;
ThreadedLoop ("Progress message", input, 0, 3).run (Functor(), input, output);
```

```
class Functor
{
  void operator() (Image<float>& in,
                   Image<float>& out)

  { ... }
}
```

**Fig. 2.** Demonstration of ThreadedLoop construct for multi-threaded image processing. A reference image ("input" in this case) and/or axes selection determine(s) the image domain over which the loop will iterate (green vectors), optionally showing a progress bar with the text provided. The programmer writes a Functor class (blue box) which has an operator that is intended to be applied to each image element individually. The loop construct automatically duplicates instances of the Functor class as required, one for each executing thread (two in this case). Each of these threads will loop over a unique section of the image(s) (a row of voxels by default), with the loop construct setting the index position prior to each functor invocation, for all images involved in the loop ("input" and "output" in this example).


```
class Source
{
  bool operator() (Data1&)
  { ... }
}
```

```
class Pipe
{
  bool operator() (Data1&, Data2&)
  { ... }
}
```

```
class Sink
{
  bool operator() (Data2&)
  { ... }
}
```

```
Thread::run_queue (source, Data1(), Thread::multi (pipe), Data2(), sink);
```

**Fig. 3.** Demonstration of run_queue () construct for generalised multi-threaded processing. Each of the Source, Pipe and Sink classes defines a functor that reads from and/or writes to a specific data instance with a Queue structure that manages the transfer of data between threads. In this particular example, the Pipe class is requested to be multi-threaded, resulting in multiple instances of the Pipe class, each of which will be configured by the back-end to operate on unique instances of the Data 1 class from the first queue and write unique instances of the Data 2 class to the second queue. Multi-threaded processing can be terminated by any class at any time via the return value of the corresponding functor. Such queues are used extensively in *MRtrix3* to process non-image data, such as streamlines: for example, 'source' might read streamlines data from a file, 'pipe' might perform some operation on each streamline independently (e.g. resampling), and 'sink' might write them back to file.

the *MRtrix3* project, there are a number of external projects providing precisely this kind of functionality, for instance, the *MRtrix3_connectome* BIDS application.[22] Docker[23] and Singularity[24] containers are also available.

One outstanding issue with the use of *MRtrix3* in HPC or cloud environments is the lack of reliable remote display capability. This is due to *MRtrix3* using the OpenGL 3.3 specification, which is currently not (or at best poorly) supported for X11 remote rendering. At time of writing there are no simple solutions for this limitation.

[22] https://github.com/BIDS-Apps/MRtrix3_connectome.
[23] https://hub.docker.com/r/neurology/mrtrix/.
[24] https://singularity-hub.org/collections/729.

### 3.3. Developer tools

*MRtrix3* is developed using modern software practices with transparency in mind. Software version control is managed using Git, hosted openly on Github.[25] Issues are tracked and discussed publicly on GitHub.[26] Most *MRtrix3* commands have tests to ensure the output remains consistent with expectations throughout the development cycle. Test data are hosted on a separate Git repository[27] to keep the main source code repository small and lightweight. Continuous integration testing is managed by TravisCI[28] (for Linux and macOS) and AppVeyor[29] (for Microsoft Windows).

*MRtrix3* is designed to minimise the amount of effort required to develop complete image processing applications, as illustrated in Figs. 1–3. Developers have access to a range of classes for seamless access to images and other types of data, facilities for multi-threading operations over images and for parallel processing of streams of data, image filters and other convenience methods. The core API documentation is available online[30], and is generated directly from the code using Doxygen.[31]

The development tools and build system are demonstrated in Appendix B (other examples are available in the developer documentation[32]). The example binary command in B.1 also illustrates the image access and multi-threading primitives provided in the *MRtrix3* libraries.

#### 3.3.1. Integrated command-line interface and documentation

*MRtrix3* provides a framework for simultaneously documenting, specifying and retrieving command-line arguments and options, all centralised within a single function for each application. This includes authorship and copyright information, a general description of the command and specific descriptions of the arguments and options, and any relevant references. This information is used to generate the application's online help page, and at runtime to display its help page, to automatically check the validity of the arguments, and to retrieve these arguments and options at any point in the code. See the code example in Appendix B for an illustration of this framework.

#### 3.3.2. Build system

The software includes a bespoke system to manage configuring and building the software. It is comprised of two Python scripts: 'configure' and 'build'. The former detects various aspects of the operating environment (OS, availability of C++11 compliant compiler and required libraries, etc). The build script scans the code to build a full dependency tree based on simple rules about file location and naming conventions, identifies outdated targets, assembles a list of tasks required to bring the targets up to date, and then launches the relevant commands in the right order and in parallel for fast build times. This allows the code to be modified without requiring any other changes (for example, to external Makefiles): a new command can simply be dropped into the right folder, and its corresponding binary will be compiled the next time build is invoked.

#### 3.3.3. External contributions

*MRtrix3* encourages users to contribute their own methods to the software and recognises their need to distribute these however they see fit. Therefore, the *MRtrix3* build system provides a convenient primitive for external modules that enables developers to build their own applications, linked to the core *MRtrix3* library and leveraging its image access and multi-threading primitives. These external modules can be distributed separately under any open source license or may be integrated into a future *MRtrix3* release. In either case, developers are given full credit for their work through embedding of author information, references, and copyright statement, within the code and resulting command help page and documentation.

#### 3.3.4. Python scripts

While piping of images (as described in Unix pipes (section 3.2.1)) can be used to chain several *MRtrix3* commands into composite applications, *MRtrix3* also contains a dedicated Python scripting library to aid the development of more complex applications and processing pipelines involving multiple *MRtrix3* (or other) commands. In contrast to projects such as DiPy (Garyfallidis et al., 2014), this library does not provide functionality to manipulate raw image data within Python itself, but provides functionality to invoke and manage external high-performance executables to perform the actual computations. Features of the scripting library include: standard command-line options; help page formatting and documentation generation consistent with *MRtrix3* binary commands; scratch directory management; and the ability to continue from any point in a previous execution for long or computationally demanding pipelines. The application module and library functions on which these scripts are built are additionally designed to be both accessible and functional for researchers looking to construct their own processing pipelines from existing commands and to interface with external tools, with features described in "3.3.3 External contributions" applicable to such scripts just as for C++ binaries. An example of such a script is provided in Appendix B.

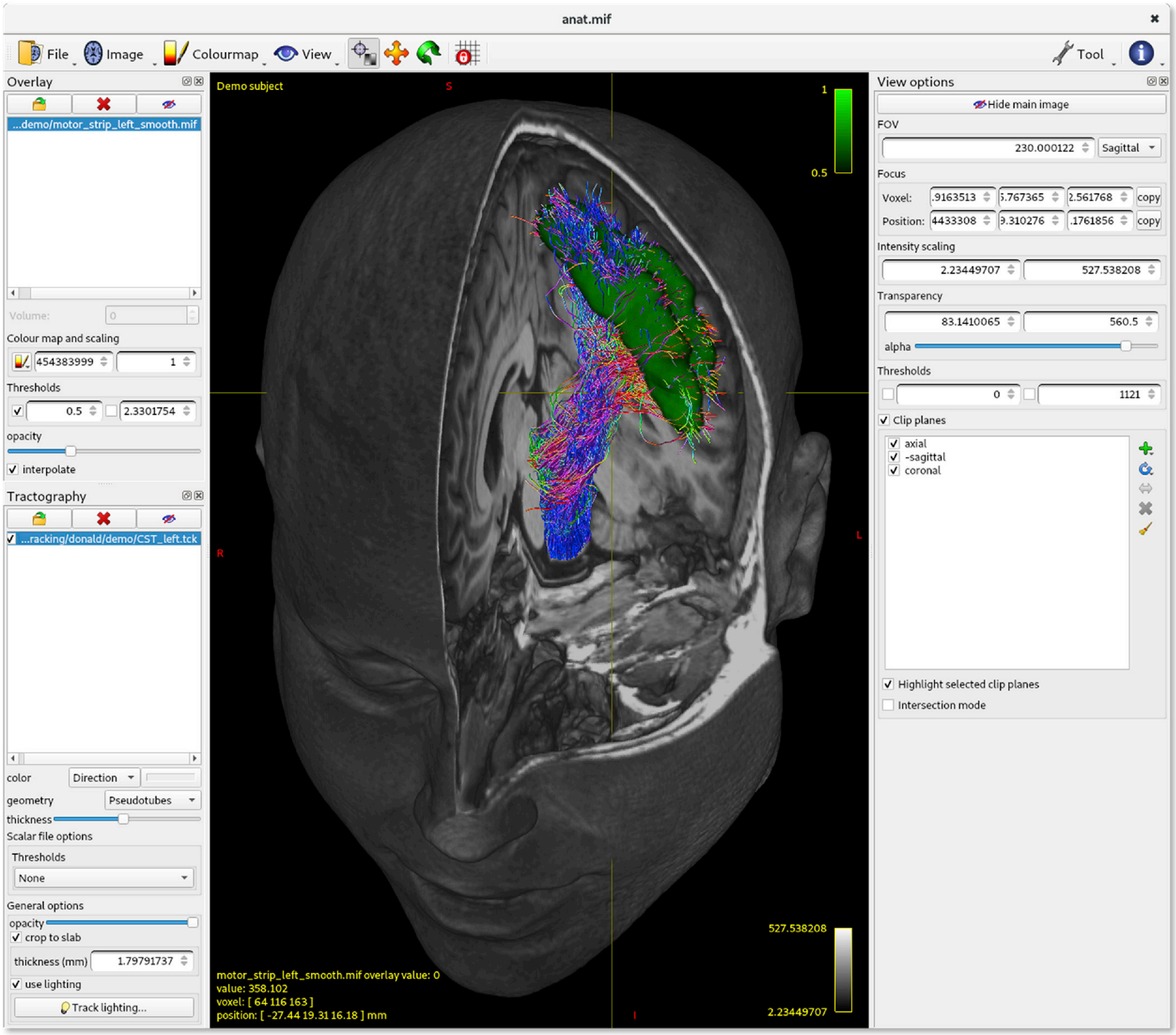
## 4. MRView

The *MRtrix3* viewer, *MRView*, was written to be cross-platform, high performance, convenient to use, and extensible. To meet these aims, *MRView* uses the high-performance industry-standard OpenGL[33] 3.3 API to leverage the graphics capabilities of modern systems, and the widely supported and open-source Qt toolkit[34] to manage the interface elements (widgets). It is also designed from the outset for interactive use from the command-line. This allows users to rapidly experiment with new ideas, develop and debug pipelines, and inspect the output of processing commands. Two aspects of its design play a role in supporting this: first, *MRView* accepts input from Unix pipes (as illustrated in the example in section 3.2.1); and second, it is quick to launch and display (through a number of design decisions, including the use of memory-mapping for "lazy loading" and delayed initialisation of interface elements until the point of use). See the *MRtrix3* website for demonstration videos.[35]

An important aspect of *MRView* is its ability to load multiple images concurrently, allowing fast switching between them via the menu or the PageUp/PageDown keys. In combination with the optimisations mentioned above, this allows interactive use in otherwise demanding workflows, such as loading a long list of images into the viewer concurrently and inspecting them with interactive framerates, even when the data are held on network file shares. This type of action is useful to quickly scan through the results of a processing step over all participants in a large study (to check the quality of the image masks, for example).

*MRView* is also capable of displaying slice-wise information at arbitrary oblique cuts through the data. This is particularly useful to

[25] https://github.com/MRtrix3/mrtrix3/.
[26] https://github.com/MRtrix3/mrtrix3/issues.
[27] https://github.com/MRtrix3/test_data
[28] https://travis-ci.org/MRtrix3/mrtrix3.
[29] https://ci.appveyor.com/project/MRtrix3/mrtrix3.
[30] http://www.mrtrix.org/developer-documentation/.
[31] http://www.doxygen.org/.
[32] http://www.mrtrix.org/developer-documentation/examples.html.
[33] https://www.opengl.org/.
[34] https://www.qt.io/.
[35] http://www.mrtrix.org/videos.



...demo/motor_strip_left_smooth.mif
Volume: 0
Colour map and scaling: 454383999 1
Thresholds: ✓ 0.5 2.3301754
opacity
✓ interpolate

✓ ...racking/donald/demo/CST_left.tck
color: Direction
geometry: Pseudotubes
thickness
Scalar file options
Thresholds: None
General options
opacity
✓ crop to slab
thickness (mm): 1.79791737
✓ use lighting

FOV: 230.000122 Sagittal
Focus
Voxel: .9163513 5.767365 2.561768 copy
Position: 4433308 9.310276 .1761856 copy
Intensity scaling: 2.23449707 527.538208
Transparency: 83.1410065 560.5
alpha
Thresholds: 0 1121
✓ Clip planes
✓ axial
✓ -sagittal
✓ coronal
✓ Highlight selected clip planes
Intersection mode

motor_strip_left_smooth.mif overlay value: 0
value: 358.102
voxel: [ 64 116 163 ]
position: [ -27.44 19.31 16.18 ] mm

527.538208
2.23449707

**Fig. 4.** *MRView* in volume render mode, showing an anatomical T1-weighted MRI with 3 clip planes, with a region of interest corresponding to the motor strip rendered using the overlay tool (in green), and probabilistic streamlines delineating the corticospinal tract shown via the tractography tool.

display features such as streamlines obtained via tractography in their most appropriate anatomical context, for instance. Off-axis rendering is available in all viewing modes (see below). *MRView* can also handle higher-dimensional images, currently up to 5D. Navigating between volumes (4th dimension) or volume groups (5th dimension) is simple and fast via the arrow keys, the menu, or the View tool (see below).

A core principle in *MRView* is that content is always displayed at the correct location in *scanner* coordinates, not relative to the *image* voxel coordinates. This applies to the main image, but also overlays and streamlines, and any directional information such as vectors or orientation density functions (ODFs). This allows the concurrent display of different types of data, independently of how these data were generated. For instance: images can be overlaid together even if their voxel sizes and/or orientations differ; streamlines produced by tractography can be displayed over other images than those used to produce them (e.g. co-registered anatomical images); ODFs or vectors will point in the expected direction irrespective of the image currently displayed; etc.

*MRView* provides a number of different viewing ***modes***, including single-slice; orthoview (a montage of 3 orthogonal slices), lightbox (a grid montage of adjacent slices or volumes); and volume render (3D rendering using ray-tracing). Fig. 4 shows a screen capture of *MRView* in volume render mode.

These modes are supplemented by a number of independent ***tools*** providing additional specific functionality. These include:

- **View**: provides fine control over generic aspects of the display, such as brightness/contrast, location of crosshairs, and access to mode-specific settings such as clip planes for the volume render.

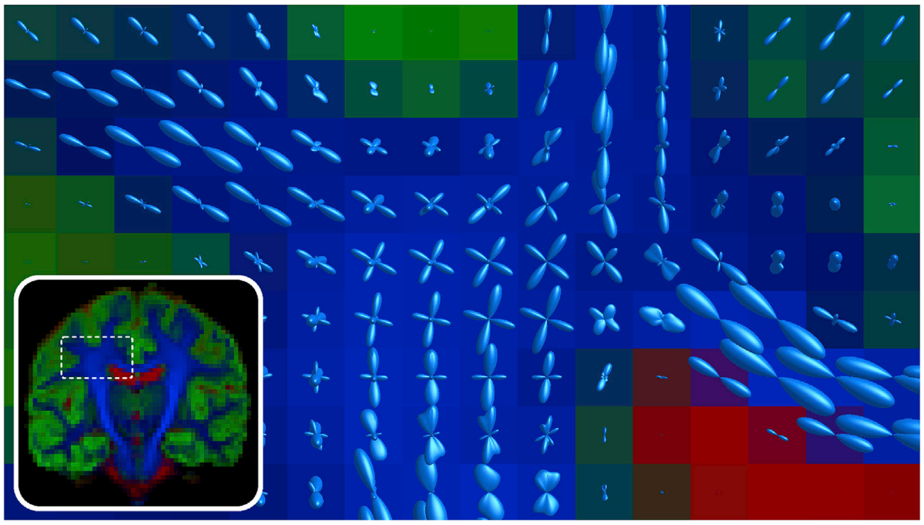


**Fig. 5.** Multi-tissue Constrained Spherical Deconvolution result using 3 tissue types: WM- (*blue*), GM- (*green*) and CSF-like (*red*) compartments, with WM fibre orientation distributions (*lighter blue*) overlaid.

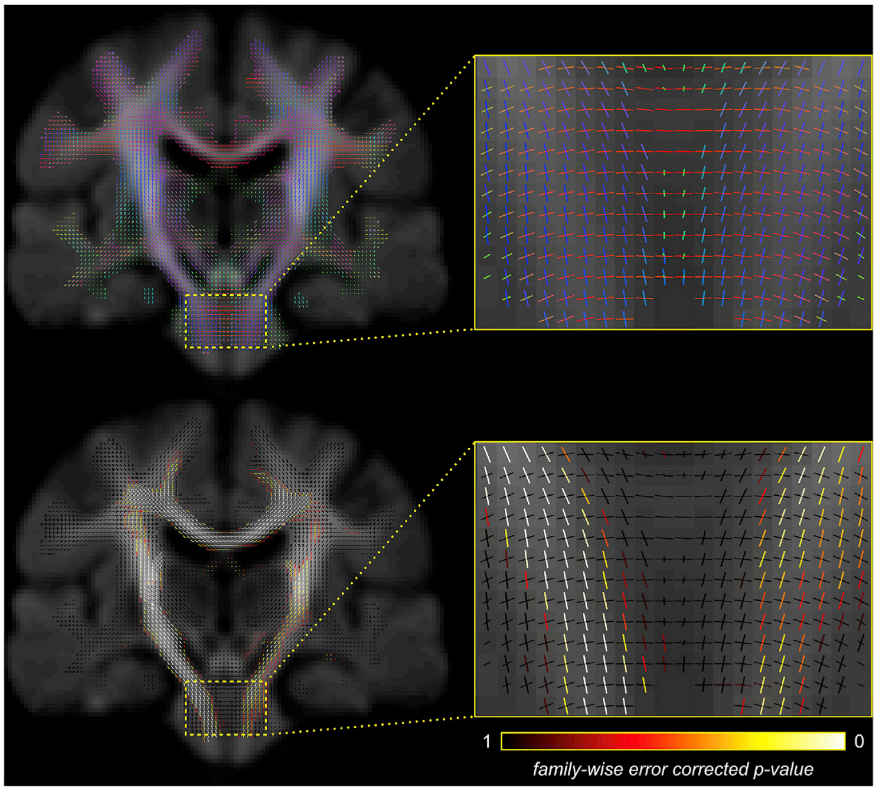
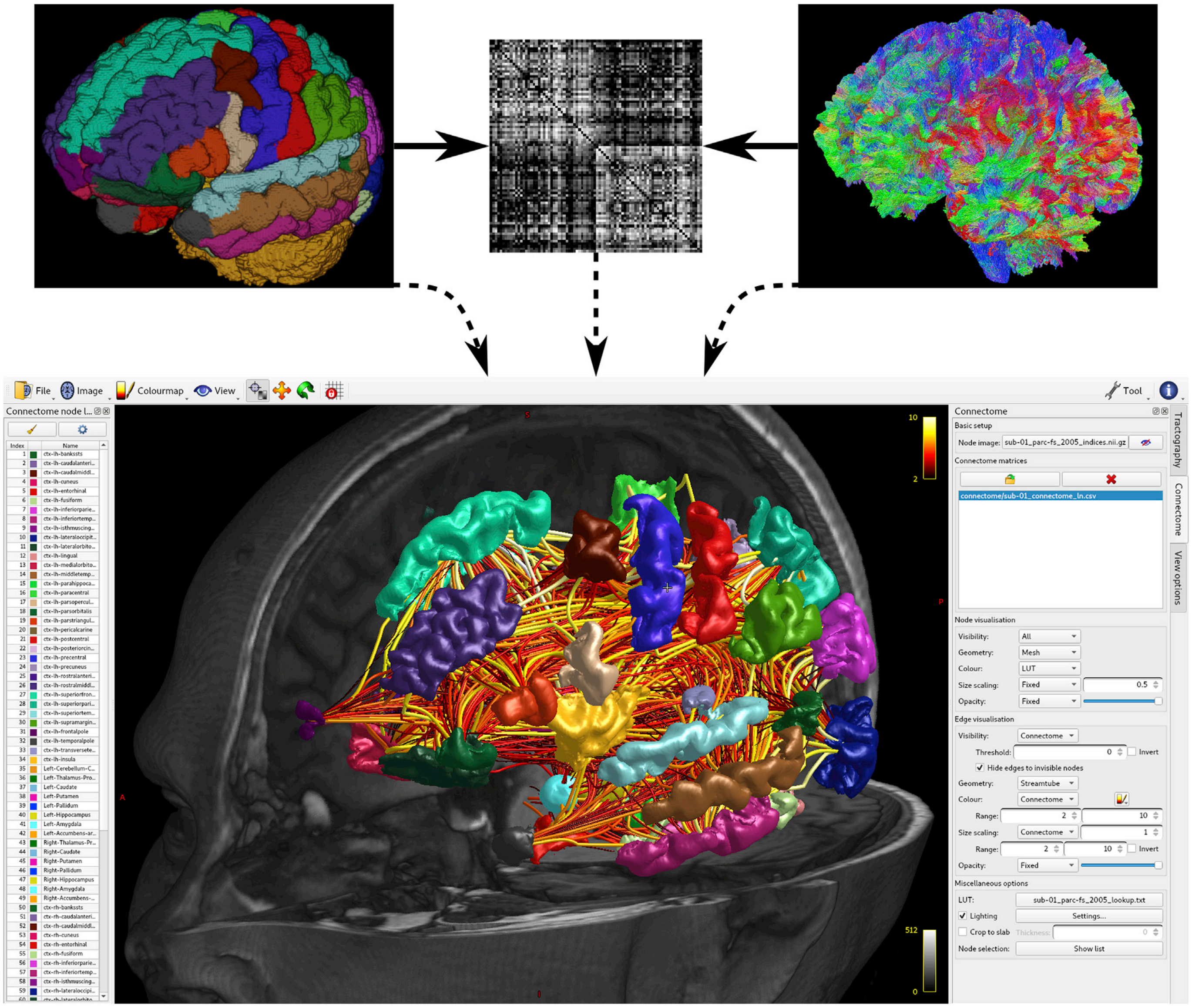


**Fig. 6.** An example of fixel-based analysis results displayed using *MRView*'s Fixel plot tool. This example shows the comparisons of fixels between motor neuron disease patients and healthy controls, with the use of connectivity-based fixel enhancement. Top row: Fixels are presented in directionally-encoded colours, red: left-right, green: anterior-posterior, blue: inferior-superior. Bottom row: Fixels are coloured based on family-wise error corrected $p$-value, highlighting the fixels with a significant fibre density loss in the patient group. This figure is adapted from (Raffelt et al., 2015) with copyright permission.

- **Overlay**: overlay additional images on top of the main image, with control over colour map, transparency, and thresholds.
- **ROI editor**: provides an interface to edit binary mask images.
- **Tractography**: visualisation of tractography output, with options for streamtube rendering.
- **ODF render** tool: visualisation of orientation density functions (Fig. 5), provided as spherical harmonic coefficients, samples along a set of fixed orientations ('dixels'[36]), or tensor coefficients.
- **Fixel plot** tool: display voxel-wise vectors, provided either as 4D images of X,Y,Z coefficients or using *MRtrix3*'s sparse fixel storage

[36] https://mrtrix.readthedocs.io/en/3.0_rc3/concepts/fixels_dixels.html.



Connectome node l...

| Index | | Name |
|---|---|---|
| 1 | 🟩 | ctx-lh-bankssts |
| 2 | 🟪 | ctx-lh-caudalanteri... |
| 3 | 🟫 | ctx-lh-caudalmiddl... |
| 4 | 🟥 | ctx-lh-cuneus |
| 5 | 🟥 | ctx-lh-entorhinal |
| 6 | 🟩 | ctx-lh-fusiform |
| 7 | 🟪 | ctx-lh-inferiorparie... |
| 8 | 🟥 | ctx-lh-inferiortemp... |
| 9 | 🟪 | ctx-lh-isthmuscing... |
| 10 | 🟦 | ctx-lh-lateraloccipit... |
| 11 | 🟩 | ctx-lh-lateralorbito... |
| 12 | 🟥 | ctx-lh-lingual |
| 13 | 🟥 | ctx-lh-medialorbito... |
| 14 | 🟫 | ctx-lh-middletemp... |
| 15 | 🟩 | ctx-lh-parahippoca... |
| 16 | 🟩 | ctx-lh-paracentral |
| 17 | 🟧 | ctx-lh-parsopercul... |
| 18 | 🟩 | ctx-lh-parsorbitalis |
| 19 | 🟥 | ctx-lh-parstriangul... |
| 20 | 🟫 | ctx-lh-pericalcarine |
| 21 | 🟥 | ctx-lh-postcentral |
| 22 | 🟪 | ctx-lh-posteriorcin... |
| 23 | 🟦 | ctx-lh-precentral |
| 24 | 🟪 | ctx-lh-precuneus |
| 25 | 🟪 | ctx-lh-rostralanteri... |
| 26 | 🟪 | ctx-lh-rostralmiddl... |
| 27 | 🟩 | ctx-lh-superiorfron... |
| 28 | 🟩 | ctx-lh-superiorpari... |
| 29 | 🟦 | ctx-lh-superiortem... |
| 30 | 🟩 | ctx-lh-supramargin... |
| 31 | 🟪 | ctx-lh-frontalpole |
| 32 | ⬛ | ctx-lh-temporalpole |
| 33 | 🟪 | ctx-lh-transversete... |
| 34 | 🟨 | ctx-lh-insula |
| 35 | 🟧 | Left-Cerebellum-C... |
| 36 | 🟩 | Left-Thalamus-Pro... |
| 37 | 🟦 | Left-Caudate |
| 38 | 🟪 | Left-Putamen |
| 39 | 🟦 | Left-Pallidum |
| 40 | 🟨 | Left-Hippocampus |
| 41 | 🟦 | Left-Amygdala |
| 42 | 🟧 | Left-Accumbens-ar... |
| 43 | 🟩 | Right-Thalamus-Pr... |
| 44 | 🟦 | Right-Caudate |
| 45 | 🟪 | Right-Putamen |
| 46 | 🟦 | Right-Pallidum |
| 47 | 🟨 | Right-Hippocampus |
| 48 | 🟦 | Right-Amygdala |
| 49 | 🟧 | Right-Accumbens-... |
| 50 | 🟩 | ctx-rh-bankssts |
| 51 | 🟪 | ctx-rh-caudalanteri... |
| 52 | 🟫 | ctx-rh-caudalmiddl... |
| 53 | 🟥 | ctx-rh-cuneus |
| 54 | 🟥 | ctx-rh-entorhinal |
| 55 | 🟩 | ctx-rh-fusiform |
| 56 | 🟪 | ctx-rh-inferiorparie... |
| 57 | 🟥 | ctx-rh-inferiortemp... |
| 58 | 🟪 | ctx-rh-isthmuscing... |
| 59 | 🟦 | ctx-rh-lateraloccipi... |
| 60 | 🟩 | ctx-rh-lateralorbito... |

Connectome

Basic setup

Node image: sub-01_parc-fs_2005_indices.nii.gz

Connectome matrices

connectome/sub-01_connectome_ln.csv

Node visualisation

| | | |
|---|---|---|
| Visibility: | All | |
| Geometry: | Mesh | |
| Colour: | LUT | |
| Size scaling: | Fixed | 0.5 |
| Opacity: | Fixed | |

Edge visualisation

| | | | |
|---|---|---|---|
| Visibility: | Connectome | | |
| Threshold: | 0 | | ☐ Invert |
| ✓ Hide edges to invisible nodes | | | |
| Geometry: | Streamtube | | |
| Colour: | Connectome | | |
| Range: | 2 | 10 | |
| Size scaling: | Connectome | 1 | |
| Range: | 2 | 10 | ☐ Invert |
| Opacity: | Fixed | | |

Miscellaneous options

| | |
|---|---|
| LUT: | sub-01_parc-fs_2005_lookup.txt |
| ✓ Lighting | Settings... |
| ☐ Crop to slab | Thickness: 0 |
| Node selection: | Show list |

**Fig. 7.** Generation of structural connectome matrix, and subsequent visualisation using the *MRView* Connectome tool. A brain parcellation from any of a range of data/software sources can be combined with a whole-brain tractogram generated using *MRtrix3*'s advanced tractography methods to produce a connectome matrix encapsulating a measure of connectivity between every pair of brain regions; the *MRView* connectome tool provides a range of features for displaying and navigating connectome data and/or the results of network-based statistical inference.

format[37] (Fig. 6). See (Raffelt et al., 2017b) for a description of the fixel concept.

- **Connectome** tool: visualise nodes and edges obtained through connectome analysis, using a variety of techniques (Fig. 7).
- **Screen capture** tool: write the contents of the main display to file using the widely supported open-source Portable Network Graphics (PNG) format.[38] This additionally allows the creation of movies (see for example demonstration videos on the *MRtrix3* website, and the supplementary materials provided with (Mito et al., 2018)).

## 5. Image processing tools and analyses

*MRtrix3* comes bundled with a range of end-user applications for performing general-purpose image manipulations as well as complex analyses, with a strong (though not exclusive) focus on diffusion MRI. *MRtrix3* command names follow a convention to reflect the purpose of the command and aid discoverability. Firstly, a set of shorthand "codes" are defined for the main types of data. These indicate directly within the command name itself the type of data operated on by the command; for example, "mr" for generic (magnetic resonance) images, "dwi" for diffusion-weighted images and "tck" for tracks.

Using these, two standardised formats of command names exist:

- "DataOperation", where the type of data input to the command and the operation to perform are concatenated; for example, "mrconvert" for converting images to different formats, or "mrinfo" for providing information about the image header.
- "Data2Data", where the command involves some form of conversion from one type of data to another; for example, "warp2metric"

[37] https://mrtrix.readthedocs.io/en/3.0_rc3/getting_started/image_data.html#fixel-image-directory-format.
[38] http://www.libpng.org/pub/png/.

for computing different metrics from spatial warps (deformation fields).

### 5.1. Basic image operations

*MRtrix3* provides a suite of commands for performing various common manipulations of image data. Each command is small, yet powerful and modular. This allows for performing a wide range of common processing tasks whilst providing maximal flexibility to the user. These basic commands include: querying image header information (mrinfo); converting between image formats, including data exclusion/axis permutation/header manipulation (mrconvert); concatenating images along any dimension (mrcat); a voxel-wise image calculator, allowing for simple as well as complex mathematical expressions (mrcalc); computing summary statistics (e.g. mean, std. dev., etc.) across different images or along image axes (mrmath); and generating summary statistics of image intensities within regions of interest (mrstats).

### 5.2. Diffusion analyses

*MRtrix3* includes tools to undertake different types of diffusion analyses from start to finish. In particular, it includes state-of-the-art tools for pre-processing, voxel-level modelling, fibre tracking and connectomics, and groupwise analysis. The main technologies currently included in *MRtrix3* are listed below.

**Pre-processing tools:** MP-PCA denoising (Veraart et al., 2016); removal of Gibbs ringing artefacts (Kellner et al., 2016); a convenience wrapper for FSL tools to perform motion, eddy-current, and susceptibility-induced distortion correction (Andersson et al., 2017, 2003; Andersson and Sotiropoulos, 2016); and a convenience script to perform bias field correction, based on either ANTs N4 (Tustison et al., 2010) or FSL (Smith et al., 2004; Zhang et al., 2001).

**Voxel-level modelling:** diffusion tensor imaging (DTI) (Basser et al., 1994; Veraart et al., 2013); constrained spherical deconvolution (CSD) for single-tissue fibre ODF estimation (Tournier et al., 2007, 2004) and multi-shell multi-tissue CSD for multi-shell data (Jeurissen et al., 2014), supported by several response function estimation algorithms (Dhollander et al., 2019, 2016; Tax et al., 2014; Tournier et al., 2013) (Fig. 5).

**Tractography and connectomics:** deterministic tractography using DTI (Mori et al., 1999) or fODFs (Tournier et al., 2012); probabilistic tractography using DTI via the wild bootstrap (Jones, 2008) or fODF sampling (Jeurissen et al., 2011; Tournier et al., 2012, 2010); multi-shell multi-tissue global tractography (Christiaens et al., 2015); anatomically-constrained tractography (Smith et al., 2012); spherical-deconvolution informed filtering of tractograms (SIFT) (Smith et al., 2013) and its newer SIFT2 variant (Smith et al., 2015); tools to support connectomic analyses, including network-based statistics (NBS) (Zalesky et al., 2010) and its threshold-free variant (Vinokur et al., n. d.; Baggio et al., 2018).

**Group-wise image analysis:** diffeomorphic registration of ODF images and construction of population templates (Raffelt et al., 2011); voxel-based analysis using permutation testing (Anderson and Robinson, 2001; Winkler et al., 2014) and threshold-free cluster enhancement (Smith and Nichols, 2009); and fixel based analysis (Raffelt et al., 2017b) using connectivity-based fixel enhancement (Raffelt et al., 2015).

**Visualisation and maps:** directionally-encoded colour (DEC) maps for tensors (Pajevic and Pierpaoli, 1999) and fODFs (Dhollander et al., 2015b); panchromatic sharpening (Dhollander et al., 2015a) and luminance correction (Dhollander et al., 2018); track density imaging (TDI) (Calamante et al., 2010), track-weighted imaging (TWI) (Calamante, 2017) and track orientation density imaging (TODI) (Dhollander et al., 2014).

**Table 1**
Breakdown of the runtime for each processing step. See text for details.

| Processing step | Command used | runtime |
|---|---|---|
| MP-PCA denoising | dwidenoise | 1m45s |
| Gibbs ringing removal | mrdegibbs | 12s |
| Motion, eddy-current & susceptibility induced distortion correction and outlier rejection | dwipreproc *invokes FSL commands topup, applytopup and eddy* | 32m59s |
| Bias field correction | dwibiascorrect *invokes ANTs command N4BiasFieldCorrection* | 9s |
| Response function estimation | dwi2response | 28s |
| Multi-tissue CSD | dwi2fod | 1m12s |
| T1 tissue segmentation | 5ttgen *invokes FSL commands bet, fast & run_first_all* | 7m29s |
| Probabilistic tractography with ACT (1 million streamlines) | tckgen | 7m33s |

### 5.3. Typical runtimes

The following provides a breakdown of the time required to process a typical multi-shell diffusion MRI dataset as might be acquired in routine clinical practice. This takes raw data in DICOM format, runs through the recommended pre-processing pipeline, and performs whole-brain tractography to produce 1 million streamlines.

The input data was taken from the B.A.T.M.A.N. tutorial.[39] These were acquired on a Siemens Prisma 3T equipped with a 64-channel receiver head coil, and consisted of a high-resolution structural (TE/TR = 3.67/1910 m s, 9° flip angle, 256 x 256 matrix, 1 mm isotropic voxels, 160 slices, acquisition time: 5 min) and a multi-shell dMRI acquisition (TR/TE 8500/110 m s, 2.5 x 2.5 $\times$ 2.5 mm$^3$ voxels, 96 x 96 matrix, 60 slices, 3 PE-reversed b = 0 volumes), acquired at $b = 0$, 1000, 2000, 3000 s/mm$^2$ with 15, 17, 31, 50 volumes respectively, for a total acquisition time of 17 min.

Processing was performed on a standard desktop system running 64-bit Arch Linux, with the following specifications:

- Intel(R) Core(TM) i7-4930 K CPU, 6 cores with hyper-threading, 3.40 GHz
- 32 GB RAM (4 $\times$ 8GB DDR3 DIMMs)
- NVIDIA GeForce GTX 780 (3 GB RAM, 2304 CUDA cores)

The total runtime for the full pipeline was 52 min, with the breakdown per command shown in Table 1.

## 6. Conclusion and future directions

*MRtrix3* was designed from the outset as a framework for the development and dissemination of high-performance imaging research applications, guided by the principles of open and reproducible science. Our efforts over the next few years will be focussed on three main areas:

- Continue to provide high-performance implementations of the most promising analysis methods in our areas of expertise. We believe this is the main reason our users chose to use *MRtrix3*, and one of the main motivations behind the project. While this is currently focused mainly on diffusion MRI, we hope to broaden the range of applications provided by *MRtrix3* through our own research and external collaborations.

[39] http://www.mrtrix.org/2018/09/26/mrtrix-tutorial-available-on-osf/.

- Encourage adoption of the software amongst neuroimaging users, both as a general purpose set of tools, and for advanced diffusion MRI analysis specifically. So far, the *MRtrix3* user base has grown primarily through word of mouth, facilitated particularly by our active support on the *MRtrix3* forum. We intend to expand on this with additional tutorials, video demonstrations of commonly requested tasks, hands-on software workshops, and an expanded user-editable Frequently Asked Questions section.
- Encourage use of the *MRtrix3* library amongst C++ developers in the neuroimaging community. The *MRtrix3* API has been designed from the outset to be relatively domain-agnostic, and to allow rapid development of new analysis methods with few lines of code. We are therefore keen to see it used by other researchers working in other domains, and to foster a dynamic and thriving community of developers around it. This will require increased efforts to provide tutorials, document all aspects of the API, and ongoing support for C++ developers.

As such, *MRtrix3* will continue to grow a toolset and nurture a supportive community by and for MRI researchers. While the focus of *MRtrix3* has historically been on diffusion MRI analysis, the software itself is designed to be compatible with the broadest range of applications possible. The codebase is currently stable and we have no plans for starting a new major release or to deprecate large parts of the API. We invite developers to use our software framework for new applications, and encourage researchers to consider using *MRtrix3* in their work, whether focussed on diffusion MRI or other imaging modalities.

## Acknowledgements

This work was primarily supported by the National Health and Medical Research Council (NHMRC) of Australia (Program Grants 400121, 628952, 1091593; AC research fellowship 1026077). We are additionally grateful to the Victorian Government's Operational Infrastructure Support Program for their support.

This work also received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013/ERC grant agreement no. [319456] dHCP project), and was supported by the Wellcome/EPSRC Centre for Medical Engineering at King's College London [WT 203148/Z/16/Z]; the Medical Research Council [MR/K006355/1] and by the National Institute for Health Research (NIHR) Biomedical Research Centre based at Guy's and St Thomas' NHS Foundation Trust and King's College London. The views expressed are those of the author(s) and not necessarily those of the NHS, the NIHR or the Department of Health.

RS is recently supported by fellowship funding from the National Imaging Facility (NIF), an Australian Government National Collaborative Research Infrastructure Strategy (NCRIS) capability.

BJ is a postdoctoral fellow of the Research Foundation Flanders (FWO Vlaanderen), Grant no. 12M3116N & 12M3119N.

## Appendix A. Image strides

Image data can be stored on file or in RAM in a number of ways. At heart, multi-dimensional image data consists of one value for each position, indexed by coordinates $x$, $y$, $z$, …. The first 3 coordinates conventionally refer to the voxel indices along the spatial $x$, $y$, $z$ axes, and higher indices would refer to separate volumes, or any reasonable indexing thereof. However, when stored on file or in RAM, the data will be arranged linearly, and indexed by a single *offset*. A strategy is needed to manage access to the data by mapping voxel coordinates to file (or memory) offsets.

To illustrate the issue, consider the problem of storing the elements of a dense $m \times n$ matrix. In many linear algebra packages, matrices are stored in so-called *column-major* format: each element of the first column is located directly after the other until the end of that column, followed by the elements of the next column, and so on. Assuming this storage convention, elements that are adjacent along the column direction are stored at adjacent offsets in RAM (or file), while elements adjacent along the row direction are placed at least $m$ elements apart. Hence, the element at $(i,j)$ is at offset $= i + jm$ relative to the element at index zero. This corresponds to *strides* of [ 1 $m$ ]: the step to find the next element along the column direction requires a stride of 1 element, while the equivalent step along the row direction requires a longer stride of $m$ elements.

However, matrices can be (and often are) also stored in so-called *row-major* format, where elements along the first row are stored contiguously, followed by the next row, and so on. In this case, the element at $(i,j)$ is at offset $= in + j$; this corresponds to strides of [ $n$ 1 ]. Note that while two matrices stored in row-major and column-major format respectively may be logically equivalent, the *order* in which their elements are stored differs, and there is no reason to prefer one convention over the other. In fact, many implementations support both, since it is then trivial to take the transpose of a large matrix simply by changing the strides, with no need to re-order the matrix elements themselves.

This concept trivially extends to higher dimensions, as illustrated in Figure A1. Images can be stored with all intensities along the $x$ axis stored one after the other, followed by the intensities for the next row along the $y$ axis, followed by the intensities for the next slice along the $z$ axis. For an image of size $n_x \times n_y \times n_z$, this means the intensity at voxel $(x, y, z)$ is stored at offset $= x + yn_x + zn_xn_y$; this corresponds to strides of [ 1 $n_x$ $n_x{\times}n_y$ ]. However, there is no particular reason for data to be stored in this order, and in fact different data strides are common. In many cases, the different image formats themselves assume different storage order conventions, which can cause confusion if not properly managed. Furthermore, the strides can also reflect the direction along which each axis is mapped, e.g., left-to-right versus right-to-left traversal, by allowing strides to be negative.

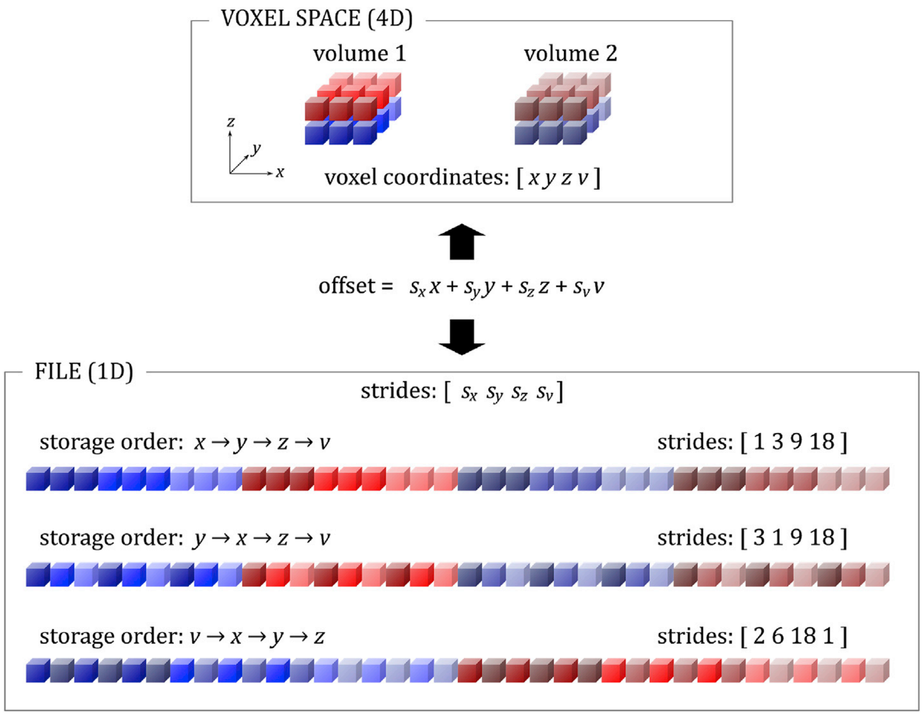

VOXEL SPACE (4D)
volume 1
volume 2
z
y
x
voxel coordinates: [ x y z v ]
offset = $s_x x + s_y y + s_z z + s_v v$
FILE (1D)
strides: [ $s_x$ $s_y$ $s_z$ $s_v$ ]
storage order: x → y → z → v
strides: [ 1 3 9 18 ]
storage order: y → x → z → v
strides: [ 3 1 9 18 ]
storage order: v → x → y → z
strides: [ 2 6 18 1 ]

**Fig. A1.** Illustration of the concept of strides when storing and accessing image data. The issue is to relate voxel coordinates [ *x y z v* ] (top box) to a single *offset* on file (bottom box). This can be done in a number of different ways. In the first case, data are stored in order of traversal along the *x* axis first, then each row along the *y* axis, then each slice along the *z* axis, and finally each volume. In the second case, the order is swapped between the *x* & *y* axes. The last case illustrates the case of volume-contiguous storage, where the data are traversed along the volume axis first, followed by the spatial axes.9

The extension to 4D can readily be appreciated in Figure A1, which also illustrates the case of volume-contiguous storage, i.e. when the intensities along the volume axis have adjacent offsets, with stride 1 for the volume index. Data stored in this way are sometimes referred to as *vector-valued* data: the data stored for each voxel is no longer a single intensity, but a *vector* of intensities. An example where this makes sense is when storing a colour image: it is then sensible to store the red, green & blue components of each pixel together. However, this is also advantageous in any situation where a vector of data is available for each voxel (e.g. all the fMRI or DWI intensities for a given voxel) and these data are processed independently per voxel (e.g. to perform a model fit). Computer hardware generally performs best when data are processed in the order they appear on file (or in memory): this avoids seek latency when reading from disk, allows burst transfers to and from the main system memory, leverages the CPU's cache prefetching engine, and maximises usage of each line of the on-board CPU cache. Volume-contiguous strides therefore make sense for high-performance per-voxel operations on 4D data, which are common in diffusion MRI, functional MRI, perfusion MRI, T2 relaxometry, and many other domains.

To allow for all these different storage conventions and to enhance performance of the various analyses to be performed, *MRtrix3* supports arbitrary data strides for all supported formats, and for data held in RAM.

### *Symbolic strides*

While the actual strides used to navigate the voxel information are unambiguous, they often involve large numbers, and depend on the exact image dimensions along each axis. To simplify the process of specifying and reporting strides, *MRtrix3* introduces the concepts of *symbolic* strides, and this is what is reported by tools such as *mrinfo*. With symbolic strides, the exact value of the stride is ignored, and all that matters is its magnitude relative to the other strides – i.e. its ranking (note that the sign of the strides is preserved). Hence, for the example shown in Figure A1, the strides would be reported as:

- $x \rightarrow y \rightarrow z \rightarrow v$, actual strides = [ 1 3 9 18 ] ⇒ symbolic strides = [ 1 2 3 4 ]
- $y \rightarrow x \rightarrow z \rightarrow v$, actual strides = [ 3 1 9 18 ] ⇒ symbolic strides = [ 2 1 3 4 ]
- $v \rightarrow x \rightarrow y \rightarrow z$, actual strides = [ 2 6 18 1 ] ⇒ symbolic strides = [ 2 3 4 1 ]

Symbolic strides provide a simple means to specify the voxel ordering in a manner that is independent of the dimensions of the image. For example, if a 4D image had dimensions 96 × 128 × 64 × 32, with voxels stored in order x → y → z → v (as per the first example above), its actual strides would be [ 1 96 12288 786432 ]; yet its symbolic strides would be [ 1 2 3 4 ], matching the example above. If the user wishes to modify the strides of an image, it is therefore much more practical for them to specify them as symbolic strides, and let the software compute the actual strides given the dimensions of the image.

## Appendix B. Example commands

The relatively simple examples below demonstrate stand-alone commands that compile/execute with *MRtrix3* without modification. They highlight a number of the powerful features available within *MRtrix3* that can be used with minimal coding effort, greatly expediting the development of image processing commands:

- Simple construction of an external *MRtrix3 module*, enabling compilation/execution of commands making use of *MRtrix3* functionalities without requiring any modification of *MRtrix3* itself;
- Consistent command-line interface, including checking validity of input parameters at the parsing stage;
- Compatibility with *any* image file format supported by the library;
- Simple construction of terminal feedback to user regarding command progress;
- Automated self-documentation that is consistent with all *MRtrix3* commands.

Note that the examples below are intended to illustrate the use of the framework, and do not necessarily constitute recommended solutions to the particular operation to be performed here.

As an example application, the code examples below all compute the voxel-wise geometric mean of voxel intensities across volumes of an input 4-dimensional image. This can already be done using Unix pipes, with the following command line invocation:

```
$ mrcalc input.mif -log - | mrmath - mean -axis 3 - | mrcalc - -exp output.mif
mrcalc: [100%] computing: log (input.mif)
mrmath: [100%] computing mean along axis 3...
mrcalc: [100%] computing: exp (/tmp/mrtrix-tmp-rZdhfS.mif)
```

The two examples provided here demonstrate the implementation of C++ and Python commands respectively to perform the operation above.

*Example binary command*

*Code (file "demo_binary.cpp")*

This listing contains the C++ code to compute the voxel-wise geometric mean of an image along the volume dimension, optionally setting any non-finite output intensities to a user-specified value. An electronic, up to date version of this file is available online from the *MRtrix3* repository[40]

```
$ # Set up the environment
$ export PYTHONPATH=~/mrtrix3/lib
$ # Execute the command
$ ~/Downloads/demo_script input.mif output.mif
demo_script:
demo_script: Note that this script makes use of commands / algorithms that have relevant
articles for citation. Please consult the help page (-help option) for more information.
demo_script:
demo_script: Generated scratch directory: /home/user/demo_script-tmp-GAS1UB/
Command:  mrconvert /home/user/input.mif /home/user/demo_script-tmp-GAS1UB/in.mif
-strides 0,0,0,1
demo_script: Changing to scratch directory (/home/user/demo_script-tmp-GAS1UB/)
Command:  mrcalc in.mif -log log.mif
Command:  mrmath -axis 3 log.mif mean meanlog.mif
Command:  mrcalc meanlog.mif -exp out.mif
Command:  mrconvert out.mif /home/user/output.mif
demo_script: Changing back to original directory (/home/user)
demo_script: Deleting scratch directory (/home/user/demo_script-tmp-GAS1UB/)
```

*Terminal usage*

This listing shows the commands to invoke to build and execute the application for the code above. This assumes that the code sample has been saved to a file called 'demo_binary.cpp', currently residing in the user's 'Downloads' folder.

[40] https://github.com/MRtrix3/MRtrix3_demo_code/blob/master/cmd/demo_binary.cpp.

```cpp
#include "command.h" // key header for building an executable command against MRtrix3
#include "image.h" // defines class for accessing image data
#include "algo/threaded_loop.h" // tools for multi-threaded looping over images
using namespace MR;
using namespace App;

// this function sets up the command-line interface for the command
void usage()
{
  // details that must be defined for all commands
  AUTHOR = "W. Heath Robinson (h.robinson@implausible.com)";
  SYNOPSIS = "Compute voxel-wise geometric mean across volumes";

  // these are compulsory command-line inputs that must always be provided by the user
  ARGUMENTS
    + Argument ("input", "an input image").type_image_in()
    + Argument ("output", "the output image").type_image_out();

  // these optional inputs can be either provided or omitted by the user
  OPTIONS
    + Option ("invalid", "value if any intensity is non-positive (default = NaN)")
      // this option requires that a value be specified alongside it when used
      + Argument ("value").type_float();

  // example citation for the command docs
  REFERENCES
    + "Robinson, W.H. A treatise on geometric averaging. Contrived Science Weekly,
2019.";

  // manually set the copyright notice if the default is inappropriate
  COPYRIGHT = "Copyright (c) 2019 Heath Robinson Labs.";
}

// this is the functor class defining the operation to be applied
class GeometricMean
{
  public:
    GeometricMean (const float value) : invalid (value) { }
    // the operation to be performed independently for each voxel
    void operator() (Image<float>& in, Image<float>& out) {
      // grab voxel intensities across volumes as a vector:
      values = in.row(3);
      if (values.minCoeff() > 0.0)
        out.value() = std::exp (values.array().log().mean());
      else
        out.value() = invalid;
    }
    Eigen::VectorXd values;
    const float invalid;
};

// this function contains the primary operation of the command
void run ()
{
  // access the input image and check validity
  auto in = Image<float>::open (argument[0]);
  if (in.ndim() != 4)
    throw Exception ("expected 4D input image");

  // copy the input image's header, and modify to suit
  Header header = in;
  header.ndim() = 3; // <= remove 4th dimension
  header.datatype() = DataType::Float32; // <= use floating-point output

  // create the output image, using the modified header as a template
  auto out = Image<float>::create (argument[1], header);

  // create an instance of the functor with the value optionally provided by the user
  GeometricMean functor (get_option_value ("invalid", NaN));
  // performs the operation using multiple threads while displaying a progress bar
  ThreadedLoop ("computing geometric mean", in, 0, 3).run (functor, in, out);
  // output image write to disk is finalised automatically when it goes out of scope
}
```

*Help page*

This listing shows the help page print-out produced by invoking the ‘demo_binary’ command, compiled as above, with no arguments or with the ‘-help’ option. Note that the documentation matches the information provided by the developer in the ‘usage ()’ function (see code listing in 9.1.1).

```
~ $ mkdir -p ~/module/cmd                          # create necessary folder structure
~ $ cd ~/module                                    # navigate to your new module folder
~/module $ mv ~/Downloads/demo_binary.cpp cmd/     # place the cpp file in the cmd folder
~/module $ ~/MRtrix3/build                         # build file against MRtrix3 library
(1/3) [CC] tmp/cmd/demo_binary.o
(2/3) [CC] tmp/src/project_version.o
(3/3) [LN] bin/demo_binary

# you can now run your new command:
~ $ ~/module/bin/demo_binary ~/input.mif ~/output.nii.gz -invalid 0.0
demo_binary: [100%] computing geometric mean
~ $
```

*Example Python script*

*Code (file “demo_script”)*

This listing contains Python code to perform the same function as the C++ code above. An electronic, up to date version of this file is available online from the *MRtrix3* repository[41]

```
#!/usr/bin/env python

# this function sets up the command-line interface for the command
def usage(cmdline):
  cmdline.set_author('W. Heath Robinson (h.robinson@implausible.com)')
  cmdline.set_synopsis('Compute voxel-wise geometric mean across volumes')
  # command arguments and options
  cmdline.add_argument('input', help='an input image')
  cmdline.add_argument('output', help='the output image')
  cmdline.add_argument('-invalid', help='value if any intensity is non-positive (default
= NaN)')
  # optional citation and non-default copyright statement
  cmdline.add_citation('Robinson, W.H. A treatise on geometric averaging. Contrived
Science Weekly, 2019.')
  cmdline.set_copyright('Copyright (c) 2019 Heath Robinson Labs.')


# this function contains the primary operation of the command
def execute():
  from mrtrix3 import path, run
  # import data in temporary directory
  app.make_scratch_dir()
  run.command('mrconvert ' + path.from_user(app.ARGS.input) + ' ' +
path.to_scratch('in.mif') + ' -strides 0,0,0,1')
  app.goto_scratch_dir()
  # primary command processing
  run.command('mrcalc in.mif -log log.mif')
  run.command('mrmath -axis 3 log.mif mean meanlog.mif')
  if app.ARGS.invalid:
    run.command('mrcalc meanlog.mif -exp NaN {} -replace
out.mif'.format(app.ARGS.invalid))
  else:
    run.command('mrcalc meanlog.mif -exp out.mif')
  # copy to output
  run.command('mrconvert out.mif ' + path.from_user(app.ARGS.output))

# Execute the script
import mrtrix3
mrtrix3.execute()
```

[41] https://github.com/MRtrix3/MRtrix3_demo_code/blob/master/bin/demo_script.

*Terminal usage*

This listing shows the commands to invoke to set up and execute the application for the code above. This assumes that the code sample has been saved to a file called ‘demo_script’, that currently resides in the user’s ‘Downloads’ folder.

```
MRtrix 3.0_RC3-634-g730b9511       demo_binary                     Apr 23 2019

     demo_binary: part of the MRtrix3 package

SYNOPSIS

     Compute voxel-wise geometric mean across volumes

USAGE

     demo_binary [ options ] input output

        input        an input image

        output       the output image


OPTIONS

  -invalid value
     value if any intensity is non-positive (default = NaN)


Standard options

  -info
     display information messages.

  -quiet
     do not display information messages or progress status. Alternatively,
     this can be achieved by setting the MRTRIX_QUIET environment variable to a
     non-empty string.

  -debug
     display debugging messages.

  -force
     force overwrite of output files. Caution: Using the same file as input and
     output might cause unexpected behaviour.

  -nthreads number
     use this number of threads in multi-threaded applications (set to 0 to
     disable multi-threading).

  -help
     display this information page and exit.

  -version
     display version information and exit.

AUTHOR
     W. Heath Robinson (h.robinson@implausible.com)

COPYRIGHT
     Copyright (c) 2019 Heath Robinson Labs.

REFERENCES
     Robinson, W.H. A treatise on geometric averaging. Contrived Science
     Weekly, 2019.
```

*Help page*

The help page of the demo_script listed in 9.2.1 is formatted identically to the demo_binary command help page listed in 9.1.3.

## References

DSI Studio WWW Document, 2019. URL. http://dsi-studio.labsolver.org. accessed 4.29.19.

MIRTK WWW Document, 2019. Medical image registration ToolKit (MIRTK). URL. https://mirtk.github.io/. accessed 4.29.19.

Anderson, M.J., Robinson, J., 2001. Permutation tests for linear models. Aust. N. Z. J. Stat. 43, 75–88.

Andersson, J.L.R., Sotiropoulos, S.N., 2016. An integrated approach to correction for off-resonance effects and subject movement in diffusion MR imaging. Neuroimage 125, 1063–1078.

Andersson, J.L.R., Skare, S., Ashburner, J., 2003. How to correct susceptibility distortions in spin-echo echo-planar images: application to diffusion tensor imaging. Neuroimage 20, 870–888.

Andersson, J.L.R., Graham, M.S., Drobnjak, I., Zhang, H., Filippini, N., Bastiani, M., 2017. Towards a comprehensive framework for movement and distortion correction of diffusion MR images: within volume movement. Neuroimage 152, 450–466.
Ashburner, J., 2012. SPM: a history. Neuroimage 62, 791–800.
Avants, B.B., Tustison, N., Song, G., 2009. Advanced normalization tools (ANTS). Insight J 2, 1–35.
Baggio, Hugo C., Abos, Alexandra, Segura, Barbara, Campabadal, Anna, Garcia-Diaz, Anna, Uribe, Carme, Compta, Yaroslau, Jose Marti, Maria, Valldeoriola, Francesc, Junque, Carme, June 2018. Statistical inference in brain graphs using threshold-free network-based statistics. Hum. Brain Mapp. 39 (6), 2289–2302.
Basser, P.J., Mattiello, J., LeBihan, D., 1994. MR diffusion tensor spectroscopy and imaging. Biophys. J. 66, 259–267.
Calamante, F., 2017. Track-weighted imaging methods: extracting information from a streamlines tractogram. Magma 30, 317–335.
Calamante, F., Tournier, J.-D., Jackson, G.D., Connelly, A., 2010. Track-density imaging (TDI): super-resolution white matter imaging using whole-brain track-density mapping. Neuroimage 53, 1233–1243.
Christiaens, D., Reisert, M., Dhollander, T., Sunaert, S., Suetens, P., Maes, F., 2015. Global tractography of multi-shell diffusion-weighted imaging data using a multi-tissue model. Neuroimage 123, 89–101.
Cook, P.A.A., Bai, Y., Seunarine, K.K., Hall, M.G., Parker, G.J., Alexander, D.C., 2006. Camino: open-source diffusion-MRI reconstruction and processing. In: 14th Scientific Meeting of the International Society for Magnetic Resonance in Medicine, vol. 2759.
Cox, R.W., 1996. AFNI: software for analysis and visualization of functional magnetic resonance neuroimages. Comput. Biomed. Res. 29, 162–173.
Dhawan, A.P., 2011. Medical Image Analysis. John Wiley & Sons.
Dhollander, T., Emsell, L., Van Hecke, W., Maes, F., Sunaert, S., Suetens, P., 2014. Track orientation density imaging (TODI) and track orientation distribution (TOD) based tractography. Neuroimage 94, 312–336.
Dhollander, T., Raffelt, D., Smith, R., Connelly, A., 2015a. Panchromatic sharpening of FOD-based DEC maps by structural T1 information. In: Proceedings of the International Society for Magnetic Resonance in Medicine, p. 566.
Dhollander, T., Smith, R., Tournier, J.-D., Jeurissen, B., Connelly, A., 2015b. Time to move on: an FOD-based DEC map to replace DTI's trademark DEC FA. In: Proceedings of the International Society for Magnetic Resonance in Medicine., p. 1027.
Dhollander, T., Raffelt, D., Connelly, A., 2016. Unsupervised 3-tissue response function estimation from single-shell or multi-shell diffusion MR data without a co-registered T1 image. In: ISMRM Workshop on Breaking the Barriers of Diffusion MRI, p. 5.
Dhollander, T., Mito, R., Connelly, A., 2018. Tissue-Encoded Colour Fluid-Attenuated Inversion Recovery (TEC-FLAIR) map: contrast fusion designed for improved characterisation of white matter lesion heterogeneity. In: Proceedings of the International Society for Magnetic Resonance in Medicine, p. 1570.
Dhollander, T., Mito, R., Raffelt, D., Connelly, A., 2019. Improved white matter response function estimation for 3-tissue constrained spherical deconvolution. In: Proceedings of the International Society for Magnetic Resonance in Medicine, 555.
Fischl, B., 2012. FreeSurfer. Neuroimage 62, 774–781.
Garyfallidis, E., Brett, M., Amirbekian, B., Rokem, A., van der Walt, S., Descoteaux, M., Nimmo-Smith, I., Contributors, Dipy, 2014. Dipy, a library for the analysis of diffusion MRI data. Front. Neuroinf. 8, 8.
Gorgolewski, K.J., Auer, T., Calhoun, V.D., Craddock, R.C., Das, S., Duff, E.P., Flandin, G., Ghosh, S.S., Glatard, T., Halchenko, Y.O., Handwerker, D.A., Hanke, M., Keator, D., Li, X., Michael, Z., Maumet, C., Nichols, B.N., Nichols, T.E., Pellman, J., Poline, J.-B., Rokem, A., Schaefer, G., Sochat, V., Triplett, W., Turner, J.A., Varoquaux, G., Poldrack, R.A., 2016. The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. Sci. Data 3, 160044.
Gorgolewski, K.J., Alfaro-Almagro, F., Auer, T., Bellec, P., Capotă, M., Chakravarty, M.M., Churchill, N.W., Cohen, A.L., Craddock, R.C., Devenyi, G.A., Eklund, A., Esteban, O., Flandin, G., Ghosh, S.S., Guntupalli, J.S., Jenkinson, M., Keshavan, A., Kiar, G., Liem, F., Raamana, P.R., Raffelt, D., Steele, C.J., Quirion, P.-O., Smith, R.E., Strother, S.C., Varoquaux, G., Wang, Y., Yarkoni, T., Poldrack, R.A., 2017. BIDS apps: improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods. PLoS Comput. Biol. 13, e1005209.
Jenkinson, M., Beckmann, C.F., Behrens, T.E.J., Woolrich, M.W., Smith, S.M., 2012. FSL. Neuroimage 62, 782–790.
Jeurissen, B., Leemans, A., Jones, D.K., Tournier, J.-D., Sijbers, J., 2011. Probabilistic fiber tracking using the residual bootstrap with constrained spherical deconvolution. Hum. Brain Mapp. 32, 461–479.
Jeurissen, B., Tournier, J.-D., Dhollander, T., Connelly, A., Sijbers, J., 2014. Multi-tissue constrained spherical deconvolution for improved analysis of multi-shell diffusion MRI data. Neuroimage 103, 411–426.
Jiang, H., van Zijl, P.C.M., Kim, J., Pearlson, G.D., Mori, S., 2006. DtiStudio: resource program for diffusion tensor computation and fiber bundle tracking. Comput. Methods Progr. Biomed. 81, 106–116.
Johansen-Berg, H., Behrens, T.E.J., 2013. Diffusion MRI: from Quantitative Measurement to In-Vivo Neuroanatomy. Academic Press.
Jones, D.K., 2008. Tractography gone wild: probabilistic fibre tracking using the wild bootstrap with diffusion tensor MRI. IEEE Trans. Med. Imaging 27, 1268–1274.
Jones, D.K., 2010. Diffusion MRI Theory, Methods, and Applications. Oxford University Press.
Kellner, E., Dhital, B., Kiselev, V.G., Reisert, M., 2016. Gibbs-ringing artifact removal based on local subvoxel-shifts. Magn. Reson. Med. 76, 1574–1581.
Leemans, A., Jeurissen, B., Sijbers, J., Jones, D.K., 2009. ExploreDTI: a graphical toolbox for processing, analyzing, and visualizing diffusion MR data. In: Proc Intl Soc Mag Reson Med, 3537.
Mito, R., Raffelt, D., Dhollander, T., Vaughan, D.N., Tournier, J.-D., Salvado, O., Brodtmann, A., Rowe, C.C., Villemagne, V.L., Connelly, A., 2018. Fibre-specific white matter reductions in Alzheimer's disease and mild cognitive impairment. Brain 141, 888–902.
Mori, S., Crain, B.J., Chacko, V.P., 1999. Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging. Ann. Neurol. 45, 265–269.
Nolden, M., Zelzer, S., Seitel, A., Wald, D., Müller, M., Franz, A.M., Maleike, D., Fangerau, M., Baumhauer, M., Maier-Hein, L., Maier-Hein, K.H., Meinzer, H.-P., Wolf, I., 2013. The medical imaging interaction toolkit: challenges and advances. Int. J. Comput. Assist. Radiol. Surg. 8, 607–620.
Pajevic, S., Pierpaoli, C., 1999. Color schemes to represent the orientation of anisotropic tissues from diffusion tensor data: application to white matter fiber tract mapping in the human brain. Magn. Reson. Med. 42, 526–540.
Raffelt, D., Tournier, J.-D., Fripp, J., Crozier, S., Connelly, A., Salvado, O., 2011. Symmetric diffeomorphic registration of fibre orientation distributions. Neuroimage 56, 1171–1180.
Raffelt, D., Smith, R.E., Ridgway, G.R., Tournier, J.-D., Vaughan, D.N., Rose, S., Henderson, R., Connelly, A., 2015. Connectivity-based fixel enhancement: whole-brain statistical analysis of diffusion MRI measures in the presence of crossing fibres. Neuroimage 117, 40–55.
Raffelt, D., Dhollander, T., Tournier, J.-D., Tabbara, R., Smith, R.E., Pierre, E., Connelly, A., 2017a. Bias field correction and intensity normalisation for quantitative analysis of apparent fibre density. Proc. Intl. Soc. Mag. Reson. Med. 3541.
Raffelt, D., Tournier, J.-D., Smith, R.E., Vaughan, D.N., Jackson, G., Ridgway, G.R., Connelly, A., 2017b. Investigating white matter fibre density and morphology using fixel-based analysis. Neuroimage 144, 58–73.
Smith, S.M., Nichols, T.E., 2009. Threshold-free cluster enhancement: addressing problems of smoothing, threshold dependence and localisation in cluster inference. Neuroimage 44, 83–98.
Smith, S.M., Jenkinson, M., Woolrich, M.W., Beckmann, C.F., Behrens, T.E.J., Johansen-Berg, H., Bannister, P.R., De Luca, M., Drobnjak, I., Flitney, D.E., Niazy, R.K., Saunders, J., Vickers, J., Zhang, Y., De Stefano, N., Brady, J.M., Matthews, P.M., 2004. Advances in functional and structural MR image analysis and implementation as FSL. Neuroimage 23, 208–219.
Smith, R.E., Tournier, J.-D., Calamante, F., Connelly, A., 2012. Anatomically-constrained tractography: improved diffusion MRI streamlines tractography through effective use of anatomical information. Neuroimage 62, 1924–1938.
Smith, R.E., Tournier, J.-D., Calamante, F., Connelly, A., 2013. SIFT: spherical-deconvolution informed filtering of tractograms. Neuroimage 67, 298–312.
Smith, R.E., Tournier, J.-D., Calamante, F., Connelly, A., 2015. SIFT2: enabling dense quantitative assessment of brain white matter connectivity using streamlines tractography. Neuroimage 119, 338–351.
Tax, C.M.W., Jeurissen, B., Vos, S.B., Viergever, M.A., Leemans, A., 2014. Recursive calibration of the fiber response function for spherical deconvolution of diffusion MRI data. Neuroimage 86, 67–80.
Tournier, J.-D., Calamante, F., Gadian, D.G., Connelly, A., 2004. Direct estimation of the fiber orientation density function from diffusion-weighted MRI data using spherical deconvolution. Neuroimage 23, 1176–1185.
Tournier, J.-D., Calamante, F., Connelly, A., 2007. Robust determination of the fibre orientation distribution in diffusion MRI: non-negativity constrained super-resolved spherical deconvolution. Neuroimage 35, 1459–1472.
Tournier, J.D., Calamante, F., Connelly, A., 2010. Improved probabilistic streamlines tractography by 2nd order integration over fibre orientation distributions. In: Proceedings of the International Society for Magnetic Resonance in Medicine, p. 1670.
Tournier, J.-D., Calamante, F., Connelly, A., 2012. MRtrix: diffusion tractography in crossing fiber regions. Int. J. Imaging Syst. Technol. 22, 53–66.
Tournier, J.-D., Calamante, F., Connelly, A., 2013. Determination of the appropriate b value and number of gradient directions for high-angular-resolution diffusion-weighted imaging. NMR Biomed. 26, 1775–1786.
Tustison, N.J., Avants, B.B., Cook, P.A., Zheng, Y., Egan, A., Yushkevich, P.A., Gee, J.C., 2010. N4ITK: improved N3 bias correction. IEEE Trans. Med. Imaging 29, 1310–1320.
Veraart, J., Sijbers, J., Sunaert, S., Leemans, A., Jeurissen, B., 2013. Weighted linear least squares estimation of diffusion MRI parameters: strengths, limitations, and pitfalls. Neuroimage 81, 335–346.
Veraart, J., Novikov, D.S., Christiaens, D., Ades-Aron, B., Sijbers, J., Fieremans, E., 2016. Denoising of diffusion MRI using random matrix theory. Neuroimage 142, 394–406.
Vinokur, L., Zalesky, A., Raffelt, D., Smith, R.E., Connelly, A., n.d. A novel threshold -free network-based statistics method: demonstration using simulated pathology, in: OHBM. p. 4144.
Winkler, A.M., Ridgway, G.R., Webster, M.A., Smith, S.M., Nichols, T.E., 2014. Permutation inference for the general linear model. Neuroimage 92, 381–397.
Yoo, T.S., Ackerman, M.J., Lorensen, W.E., Schroeder, W., Chalana, V., Aylward, S., Metaxas, D., Whitaker, R., 2002. Engineering and algorithm design for an image processing Api: a technical report on ITK–the Insight Toolkit. Stud. Health Technol. Inform. 85, 586–592.
Zalesky, A., Fornito, A., Bullmore, E.T., 2010. Network-based statistic: identifying differences in brain networks. Neuroimage 53, 1197–1207.
Zhang, Y., Brady, M., Smith, S., 2001. Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm. IEEE Trans. Med. Imaging 20, 45–57.