NiftyPET Documentation

Release 1.1.0

Pawel J Markiewicz

Mar 05, 2021

Documentation

NiftyPET is a software platform and a Python namespace package encompassing sub-packages for high-throughput PET image reconstruction, manipulation, processing and analysis with high quantitative accuracy and precision. One of its key applications is **brain imaging in dementia** with the use of amyloid tracers. See below for the description of the above amyloid PET image reconstructed using *NiftyPET*, superimposed on the MR T1 weighted image*[0].

NiftyPET includes two packages:

- `nimpa`: https://github.com/NiftyPET/NIMPA (neuro-image manipulation, processing and analysis)
- `nipet`: https://github.com/NiftyPET/NIPET (quantitative PET neuroimaging)

The core routines are written in CUDA C and embedded in Python C extensions to enable user-friendly and high-throughput executions on NVIDIA graphics processing units (GPU). The scientific aspects of this software platform are covered in two open-access publications:

- *NiftyPET: a High-throughput Software Platform for High Quantitative Accuracy and Precision PET Imaging and Analysis* Neuroinformatics (2018) 16:95. https://doi.org/10.1007/s12021-017-9352-y
- *Rapid processing of PET list-mode data for efficient uncertainty estimation and data analysis* Physics in Medicine & Biology (2016). https://doi.org/10.1088/0031-9155/61/13/N322

An example application of *NiftyPET* in the development of novel image reconstruction:

- *Faster PET Reconstruction with Non-Smooth Priors by Randomization and Preconditioning* Physics in Medicine & Biology (2019). https://doi.org/10.1088/1361-6560/ab3d07

Although, *NiftyPET* is dedicated to high-throughput image reconstruction and analysis of brain images, it can equally well be used for **whole body imaging**. Strong emphasis is put on the data, which are acquired using positron emission tomography (PET) and magnetic resonance (MR), especially using the hybrid and simultaneous PET/MR scanners.

This software platform covers the entire processing pipeline, from the raw list-mode (LM) PET data through to the final image statistic of interest (e.g., regional SUV), including LM bootstrapping and multiple independent reconstructions to facilitate voxel-wise estimation of uncertainties.

[0] The above dynamic transaxial and coronal images show the activity of ^{18}F-florbetapir during the one-hour dynamic acquisition. Note that the signal in the brain white matter dominates over the signal in the grey matter towards the end of the acquisition, which is a typical presentation of a negative amyloid beta (Abeta) scan.

CHAPTER 1

Introduction

NiftyPET is an open source software solution for standalone and high-throughput PET image reconstruction and analysis. The key computational routines are written in CUDA C for fast and efficient processing on NVIDIA GPU devices. The routines are then embedded in Python C extensions to be readily available for high level programming in Python*[0].

1.1 Aim

The purpose of this software platform is to enable rapid processing of raw data as wells as image data for fully controlled quantitative PET image reconstruction and analysis. *NiftyPET* includes two stand-alone and independent Python packages: `nipet` and `nimpa`, which are dedicated to high-throughput image reconstruction and analysis. Strong emphasis is put on the data, which are acquired using positron emission tomography (PET) and magnetic resonance (MR), especially the hybrid and simultaneous PET/MR scanners.

This documentation intends to present all the processing chains required not only for generating a final PET image from raw data acquired on a PET/MR scanner, but also further process the image to obtain a final statistic, such as global or regional SUV (standardised uptake value) or SUVr (SUV ratio).

The `nipet` (neuro-imaging PET) Python package contains all the routines needed for robust quantitative image reconstruction from raw data with added quality control of any processing segments, e.g., photon scatter or randoms corrections. The raw data typically includes large list-mode (LM) data (usually taking up GBs), data for detector normalisation, and data for attenuation correction, i.e., the attenuation map, also called the μ-map.

The `nimpa` (neuro-image manipulation, processing and analysis) package contains all the routines for image input/output, image trimming and up-sampling for regional signal extraction (e.g. from a region of interest–ROI), image registration, and importantly, for partial volume correction (PVC).

The key aspect of `nipet` is the fast LM processing for efficient uncertainty estimation of statistic based on image or projection data. Significant emphasis in placed on reconstruction for scanners with extended axial field of view (FOV), such as the latest PET/MR systems, e.g., the Siemens Biograph mMR or the GE Signa†[0].

The processing chains include:

[0] Currently Python 2.7 is supported in version 1.0, while Python 3 support is added in version 2.0+ of NiftyPET

[0] Currently the GE Signa is not yet fully supported in *NiftyPET*.

1. list-mode data processing;
2. accurate attenuation coefficient map generation (with pseudo-CT for PET/MR);
3. detector normalisation with dead time correction;
4. exact forward and back projection between sinogram and image space;
5. estimation of reduced-variance random events;
6. high accuracy fully 3D estimation of scatter events;
7. voxel-based partial volume correction;
8. region- and voxel-level image analysis.

Due to its speed and additional functionalities, *NiftyPET* allows practical and efficient generation of multiple bootstrap realisations of raw and image datasets, being processed within arbitrarily complex reconstruction and analysis chains [5][6]. Based on these datasets, distributions of any statistic can be formed indicating the uncertainty of any given parameter of interest, for example, the regional SUVr in amyloid brain imaging.

1.2 Software infrastructure

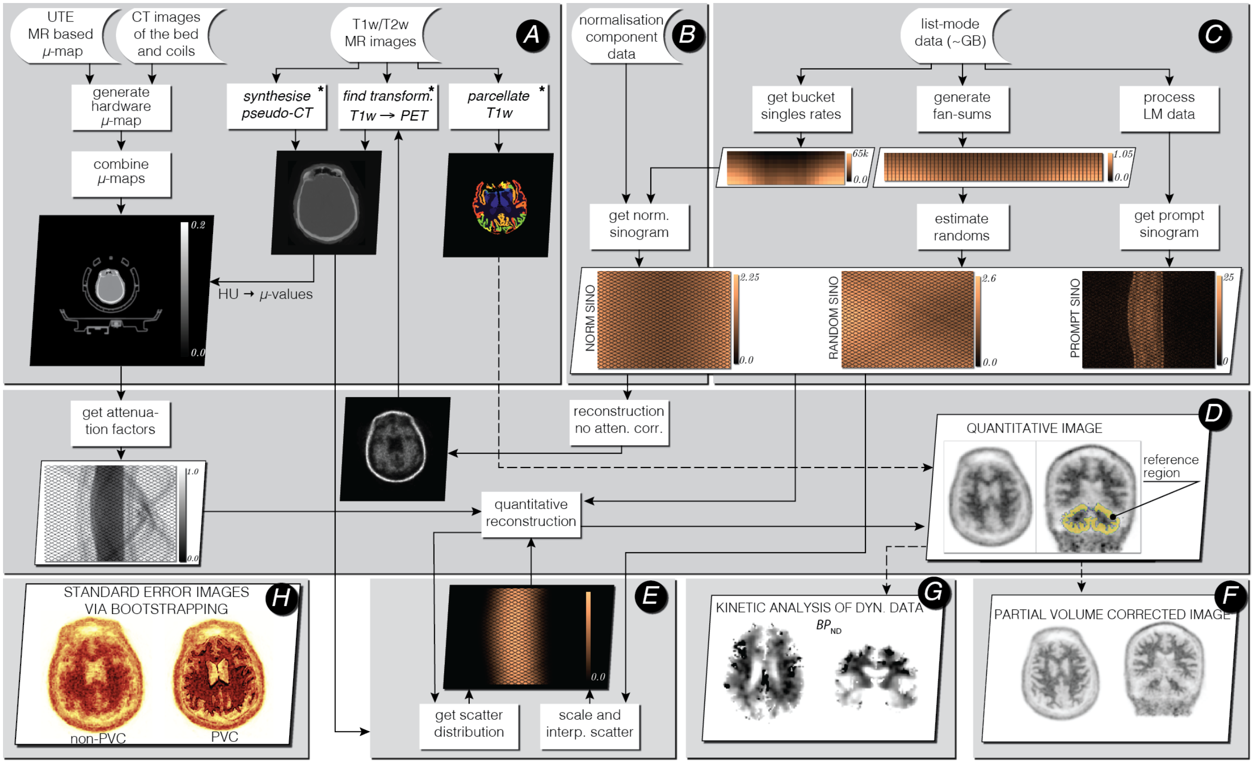

Fig. 1.1: Infrastructure for standalone PET image reconstruction and analysis of PET/MR brain data using amyloid PET tracer. The processing stages (**A-H**) are explained below. The asterisk () indicates external software packages.

For demonstration purposes, all the processing stages presented in the above figure use an amyloid brain scan acquired on the Siemens Biograph mMR. The participant was taking part in “Insight 46”–a neuroscience sub-study of the Medical Research Council National Survey of Health and Development [4].

The input data include the attenuation coefficient maps (μ-maps) of the hardware and subject (stage **A**), normalisation component data (stage **B**) and the list-mode data (stage **C**). Optionally, T1 and/or T2 weighted MR images are provided for brain parcellation used in reconstruction and analysis stage **D** and partial volume correction (PVC) stage **F**. The T1w image is also used for generating an accurate subject's μ-map.

In stage **B** the normalisation component data (relatively small file) is used to generate single factors for each sinogram bin, with the use of bucket singles obtained from list-mode (LM) processing in stage **C**. The LM processing stage **C** generates prompt and delayeds sinograms and fan sums, which are used for estimating reduced-variance randoms for each sinogram bin.

Great emphasis was put on the quantitative image reconstruction and analysis in stages **D-H** (for more details see [5]):

- forward and back projectors used for image reconstruction (stage **D**); the attenuation factors are generated with the forward projector.
- fully 3D estimation of scatter events (stage **E**), with high resolution ray tracing in image and projection space; the estimation is based on voxel-driven scatter model (VSM) and is coupled with image reconstruction, i.e., the scatter is updated every time a better image estimation of the radiotracer distribution is available.
- voxel-wise partial volume correction using MRI brain parcellations (stage **F**), based on the iterative Yang method and given point spread function (PSF) of the whole imaging system (including the hardware and the reconstruction algorithm);
- kinetic analysis using dynamic multi-frame PET data (stage **G**);
- voxel-wise uncertainty estimation based on efficient generation of bootstrap LM data replicates (stage **H**).

CHAPTER 2

Installation

Note: Quick note on ways of installation

Assumes a compatible GPU device with installed CUDA, git and cmake (details below).

- `pip` installation

  ```
  pip install --no-binary :all: --verbose nipet
  ```

  Apart from `nipet` itself, it will also install all relevant packages, including `nimpa`.

- from source using `git` and `pip`/setup.py

  ```
  git clone https://github.com/NiftyPET/NIMPA.git
  git clone https://github.com/NiftyPET/NIPET.git

  cd ./NIMPA
  pip install --no-binary :all: --verbose -e .
  cd ../NIPET
  pip install --no-binary :all: --verbose -e .
  ```

  This will install "editable" distribution at the source locations for both, `nimpa` and `nipet`. This way the packages can be modified and/or updated using `git pull`. The detailed steps for installation are given below.

NiftyPET is a Python name space package, encompassing the two packages: `nimpa` and `nipet`. Currently, the packages are available for Python 2.7 in Linux and Windows systems. So far, it has been deployed and tested on CentOS 6.8 and 7, Ubuntu 14.04, 16.04 and 18.04 as well as Windows 10 (limited), all accompanied with Python 2.7 distribution from Anaconda while using Python C extensions for the core CUDA routines. Linux systems are recommended due to their robustness and stability and the support for Windows is very limited.

2.1 Dependencies

2.1.1 Hardware

- **GPU device**: Supported are GPU devices from NVIDIA with the compute capability of at least 3.5 (e.g., NVIDIA Tesla K20/40). It is recommended to have at least 5 GB of GPU memory. *NiftyPET* supports multiple CUDA devices and so far, the following devices have been tested with *NiftyPET*:
  - NVIDIA Tesla K20/40,
  - NVIDIA Titan Xp,
  - NVIDIA GeForce GTX 1060,
  - NVIDIA GeForce GTX 1080 /Ti,
  - NVIDIA Tesla V100.
- **CPU host**: The GPU device can be accessed by a CPU host, with a reasonable computing power for some other image processing routines (e.g., image registration, etc.). It is recommended to have at least 16 GB of RAM, although we have managed to run dynamic reconstruction using old PC workstations with as little as 11 GB of RAM.

2.1.2 Software

NitfyPET installation needs pre-installed the following software:

- **C/C++ compiler**: **GCC** is used in Linux systems, while for Windows Visual Studio is used (see below for Windows installation).

  On Linux systems it can be installed as follows:
  - Ubuntu:

    ```
    apt-get install build-essential
    ```

  - CentOS:

    ```
    yum group install "Development Tools"
    ```

- **CUDA toolkit**: a parallel computing platform and programming model, which includes CUDA compiler (NVCC) and runtime API. The latest CUDA toolkit can be obtained for free from https://developer.nvidia.com/cuda-downloads. For any specific operating system follow the CUDA installation guide at https://docs.nvidia.com/cuda/index.html.

  Tip: In CentOS, it is necessary to install DKMS (Dynamic Kernel Module Support), which can be obtained from https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm. Then, it can be installed as follows:

  ```
  rpm -ivh epel-release-latest-7.noarch.rpm
  yum -y install dkms
  ```

  Make sure that CUDA is installed with appropriate paths to CUDA resources setup, that is, for CUDA 10.0 on Linux systems, it is:

```
export PATH=/usr/local/cuda-10.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64:$LD_LIBRARY_PATH
```

  This can be added to `.profile` or `.bashrc` file in your home directory (Linux). For more details see http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#post-installation-actions.

- **Git**: a version control system, used for downloading *NiftyPET* and other necessary tools, i.e.: `NiftyReg` and `dcm2niix`. For more details on installing `git` see https://git-scm.com/book/en/v2/Getting-Started-Installing-Git.

  On Linux systems it can be installed as follows:

  - Ubuntu:

    ```
    apt-get install git
    ```

  - CentOS:

    ```
    yum install git
    ```

- **cmake**: a tool for cross-platform software package building, which can be downloaded freely from https://cmake.org/install/. For the Ubuntu distribution, cmake together with the user GUI can be installed as follows:

  ```
  sudo apt-get update
  sudo apt-get install cmake cmake-curses-gui
  ```

  Tip: For CentOS, it is recommended to install `cmake` from source from the above website for the latest version (version 3). Binary and source distributions are provided.

- **Python 2.7**: a free high-level programming language, through which all the GPU routines are available for the user. The easiest way to run *NiftyPET* in Python is by using the Anaconda distribution which includes `Jupyter Notebook`. To download Anaconda visit https://www.anaconda.com/download/ and choose Python 2.7.

- **Required Python packages**: If another distribution of Python is used, make sure that the following standard Python packages are installed: `scipy`, `numpy`, `matplotlib`, `math` (these are supplied by default in the Anaconda distribution).

  - **Specialised Python packages**: the following packages for medical image I/O and manipulation are automatically installed during NiftyPET installation (not distributed by default):

    * `nibabel`: http://nipy.org/nibabel/
    * `pydicom`: http://pydicom.readthedocs.io/en/stable/getting_started.html#installing

    If for whatever reason the automatic installation fails, the two packages can be installed together as follows:

    ```
    conda install -c conda-forge nibabel
    conda install -c conda-forge pydicom
    ```

2.2 *NiftyPET* installation

2.2.1 Using `pip`

- NiftyPET:`nimpa`

To install nimpa with CUDA source compilation for the given CUDA version and operating system (Linux is preferred), simply type:

```
pip install --no-binary :all: --verbose nimpa
```

- NiftyPET:nipet

  To install nipet, the core of NiftyPET image reconstruction, type:

  ```
  pip install --no-binary :all: --verbose nipet
  ```

  This will also install nimpa if it is not yet installed and will compile the CUDA C source code for the user's Linux system and CUDA version (>=7.0).

2.2.2 From source using git and pip/setup.py

The source code of full version of nimpa and nipet packages can be downloaded to a specific folder using git as follows:

```
git clone https://github.com/NiftyPET/NIMPA.git
git clone https://github.com/NiftyPET/NIPET.git
```

Alternatively, it can also be downloaded with an *SSH* key pair setup:

```
git clone git@github.com:NiftyPET/NIMPA.git
git clone git@github.com:NiftyPET/NIPET.git
```

After a successful download, navigate to folder nimpa and run inside one of the following:

```
1) python setup.py install
2) pip install --no-binary :all: --verbose .
3) pip install --no-binary :all: --verbose -e .
```

The last option with the -e makes the installation "editable", alowing the user to modify the source code themselves or by pulling newer versions from git using git pull.

Identically for nipet, run one of the following:

```
1) python setup.py install
2) pip install --no-binary :all: --verbose .
3) pip install --no-binary :all: --verbose -e .
```

The installation will call on cmake, which will run automatically and generate make files, and then run make to build all the CUDA C routines and Python C extensions. Following this, the compiled Python modules will be installed into the specific Python package location.

2.2.3 Third party software installed with *NiftyPET*

The installation *NiftyPET* will automatically install additional third party software, used for extra capabilities, such as image registration and conversion. *NiftyReg* and *dcm2niix* will be installed in NiftyPET_tools folder, in your home directory:

- **dcm2niix**: conversion of DICOM images to NIfTI images (v1.0.20171204). If for some reason the automatic installation fails (e.g., due to a problem with dependencies), try to download the source code from https://github.com/rordenlab/dcm2niix and compile it, or use the pre-complied version with current release available at https://github.com/rordenlab/dcm2niix/releases/.

- **NiftyReg**: image registration and resampling tool. The stable version (16 Nov 2017) is fetched and installed automatically using

  ```
  git clone https://github.com/KCL-BMEIS/niftyreg/
  ```

  Some details for a manual install can be found at http://cmictig.cs.ucl.ac.uk/wiki/index.php/NiftyReg_install (can be outdated).

2.2.4 Installation in Conda environment

One of the advantages of using conda (part of Anacoda) and Python is the possibility of having separate environments for different versions of Python and/or packages installed in them. Thus, conda environments enable the user to set up *NiftyPET* differently for various applications (e.g., different image resolution, radio-pharmaceutical-optimised attenuation and/or scatter correction, etc.). Below is demonstrated an installation of NiftyPET into environment called *niftypet*.

Create environment called, for example, *niftypet*, by running this command:

```
conda create --name niftypet
```

Activate the conda environment in Linux:

```
source activate niftypet
```

in Windows:

```
activate niftypet
```

It may be necessary to install additional required packages, like the following:

```
conda install -c anaconda pycurl
conda install -c anaconda matplotlib
conda install -c anaconda ipython
conda install -c conda-forge nibabel
conda install -c conda-forge pydicom
```

NiftyPET can now be installed as described above in *NiftyPET installation*, while making sure that the conda environment is active. Please note, that for some reason it may be necessary to deactivate the conda environment and then active it again (and close the terminal) so that the *NiftyPET* package will be recognised in the specific path of the Python environment, and be thus importable (import nipet).

2.3 Post-installation checks

2.3.1 Default CUDA device

The default CUDA device used for GPU calculations is chosen during the installation together with the CUDA architecture code compilation, which is specific for a given GPU device with a specific compute capability. This information is stored in resources.py in ~/.niftypet/ folder, created during the installation (additional folder may be present corresponding to the conda environment). For example, for the NVIDIA Titan Xp with compute capability of 6.1, it will look like this:

```
# DO NOT MODIFY BELOW--DONE AUTOMATICALLY
### start GPU properties ###
DEV_ID = 0
CC_ARCH = '-gencode=arch=compute_61,code=compute_61;'
### stop GPU properties ###
```

Any available (installed) CUDA devices can be chosen within Python for any image reconstruction or part of the reconstruction pipeline.

2.3.2 Paths for the third-party software

If for some reason, the paths to the tools for image registration, resampling and conversion (DICOM -> NIfTI) are found incorrect, it can be checked by viewing `resources.py` file in `~/.niftypet` folder in Linux (for `conda` environment there will be an additional folder with the name of the environment, which contains `resources.py`, specific for the environment). In Windows, it is located in the local application data folder. It is recommended that the paths and device properties are not manually edited, but are changed rather by rerunning the installation.

```
# paths to apps and tools needed by NiftyPET
### start NiftyPET tools ###
PATHTOOLS = '/path/to/NiftyPET_tools/'
RESPATH = '/path/to/NiftyPET_tools/niftyreg/bin/reg_resample'
REGPATH = '/path/to/NiftyPET_tools/niftyreg/bin/reg_aladin'
DCM2NIIX = '/path/to/NiftyPET_tools/dcm2niix/bin/dcm2niix'
HMUDIR = '/path/to/mmr_hardware_mumaps'
### end NiftyPET tools ###
```

Note that the hardware μ-maps are not distributed with this software, and have to be obtained from the Siemens Biograph mMR scanner.

2.4 Jupyter Notebook

Jupyter Notebook is a wonderful tool, useful for sharing and replicating image reconstruction methods written in Python. It allows introspection, plotting and sharing of any intermediate results (e.g., sinograms and images generated during the reconstruction pipeline) or any end result. For this reason, it is best when Python and iPython are installed through Anaconda, which by default includes Jupyter Notebook. See http://jupyter.readthedocs.io/en/latest/tryjupyter.html for more details and http://jupyter.readthedocs.io/en/latest/install.html for a manual installation.

CHAPTER 3

Accessing and querying GPU devices

IPython is an interactive Python prompt, particularly useful for PET/MR imaging. Start IPython by issuing `ipython` in the terminal to obtain the following:

```
Python 2.7.15 |Anaconda custom (64-bit)| (default, May  1 2018, 23:32:55)
Type "copyright", "credits" or "license" for more information.

IPython 5.7.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Note: At this stage Python 2.7 is supported. Support for Python 3 is coming soon.

Import `nimpa` package from Python *NiftyPET* namespace and run the :

```
from niftypet import nimpa
nimpa.gpuinfo()
```

which for two NVIDIA GPU devices used for this documentation (TITAN Xp and Quadro K4200) will produce:

```
In [1]: from niftypet import nimpa
   ...: nimpa.gpuinfo()
   ...:
Out[1]: [('TITAN X (Pascal)', 12788L, 6L, 1L), ('Quadro K4200', 4231L, 3L, 0L)]

In [2]:
```

Please note that only the first device is supported, i.e., the compute capability is `>=3.5`:

```
In [3]: nimpa.gpuinfo()[0][2:]
Out[3]: (6L, 1L)
```

The same fuction is available in `nipet`, i.e.:

```
from niftypet import nipet
nipet.gpuinfo()
```

It's also possible to get extended information about the installed GPU devices by running either

```
nipet.gpuinfo(extended=True)
```

or

```
nimpa.gpuinfo(extended=True)
```

For the above GPU devices, the following will be obtained:

```
In [4]: nimpa.gpuinfo(extended=True)
i> there are 2 GPU devices.

----------------------------------------
CUDA device: TITAN X (Pascal), ID = 0
----------------------------------------
i> total memory [MB]:12788.50
i> shared memory/block [kB]:   49.15
i> registers (32bit)/thread block: 65536
i> warp size: 32
i> compute capability: 6.1
i> clock rate [MHz]: 1531.00
i> ECC enabled? 0
i> max # threads/block: 1024

i> Memory available: 12788.50[MB]
   Used: 504.37[MB]
   Free:12284.13[MB]


----------------------------------------
CUDA device: Quadro K4200, ID = 1
----------------------------------------
i> total memory [MB]:4231.99
i> shared memory/block [kB]:   49.15
i> registers (32bit)/thread block: 65536
i> warp size: 32
i> compute capability: 3.0
i> clock rate [MHz]:  784.00
i> ECC enabled? 0
i> max # threads/block: 1024

i> Memory available: 4231.99[MB]
   Used:1117.06[MB]
   Free:3114.93[MB]

[('TITAN X (Pascal)', 12788L, 6L, 1L), ('Quadro K4200', 4231L, 3L, 0L)]
Out[4]: [('TITAN X (Pascal)', 12788L, 6L, 1L), ('Quadro K4200', 4231L, 3L, 0L)]
```

CHAPTER 4

DICOM anonymisation

DICOM data anonymisation or pseudonymisation is a process of removing sensitive personal information from DICOM attributes . This ensures compliance with the EU's General Data Protection Regulation (GDPR); so that the people who are imaged remain anonymous when their image data is distributed for wider scientific or clinical research.

The difference between anonymisation and pseudonymisation is that the former does not allow re-identification, while the latter does. In pseudonymisation the personally identifiable information is replaced with artificial identifiers, which must be kept separately and safely. Importantly, however, it has to be recognised that with current machine learning techniques, absolute protection of digital imaging data is almost impossible. For example, in an MR study of the head, effective anonymisation would also need to involve irreversible image deformation of the face [7].

4.1 Anonymisation in *NiftyPET*

The anonymisation can be achieved using function `dcmanonym` in sub-package `nimpa` through the use of Pydicom. For example, sensitive information, such as the patient's name, date of birth, etc., can be checked and displayed by:

```
from niftypet import nimpa
dcmpath = '/path/to/DICOM-file'
nimpa.dcmanonym(dcmpath, displayonly=True)
```

If it needs anonymisation or pseudonymisation, the sensitive information, such as the patient's name, date of birth (`dob`) and physician's name, has to be replaced by the user-defined strings, i.e.:

```
nimpa.dcmanonym(dcmpath,
                patient='Anonymous',
                physician='DR. Anonymous',
                dob='19000101',
                verbose=True)
```

The argument `dcmpath` in the above call, can be not only the path (a Python string) to a single DICOM file, but also it can be a Python list of DICOM file paths, the path to a folder containing DICOM files, or a dictionary `datain` containing all the data (including raw data) needed for a standalone image reconstruction. For example, assuming that the scanner (here the Biograph mMR) is initialised and the path to raw data is provided, i.e.:

```
from niftypet import nipet
from niftypet import nimpa

# get all the scanner parameters
mMRparams = nipet.get_mmrparams()
folderin = '/path/to/raw-data'

# automatically categorise the input data
datain = nipet.classify_input(folderin, mMRparams)
```

then all the DICOM data required for reconstruction can be anonymised simply by:

```
nimpa.dcmanonym(datain,
                patient='RandomID',
                physician='Dr. Hopefully Nice',
                dob='19800101',
                verbose=True)
```

Please note, that the anonymisation of the Biograph mMR data goes somewhat deeper than just modifying the DICOM attributes—it also extracts the specific CSA headers and searches them for the patient's name.

CHAPTER 5

Basic PET image reconstruction

For basic Siemens Biograph mMR image reconstruction, *NiftyPET* requires:

(1) PET list-mode data;

(2) component-based normalisation file(s);

(3) the μ-map image (the linear attenuation map as a 3D image).

An example of downloadable raw PET data of amyloid brain scan is provided in *Raw brain PET data* [1].

5.1 Initialisation

Prior to dealing with the raw input data, required packages need to be imported and the Siemens Biograph mMR scanner constants and parameters (transaxial and axial lookup tables, LUTs) have to be loaded:

```
import numpy as np
import sys, os, logging

# NiftyPET image reconstruction package (nipet)
from niftypet import nipet
# NiftyPET image manipulation and analysis (nimpa)
from niftypet import nimpa

logging.basicConfig(level=logging.INFO)
# get all the scanner parameters
mMRpars = nipet.get_mmrparams()
```

5.2 Sorting and classification of input data

All the part of the input data is aimed to be automatically recognised and sorted in Python dictionary. This can be obtained by providing a path to the folder containing the unzipped file of the freely provided raw PET data in *Raw brain PET data*. The data is then automatically explored and sorted in the output dictionary `datain`:

```
# Enter the path to the input data folder
folderin = '/path/to/input/data/folder/'

# automatically categorise the input data
datain = nipet.classify_input(folderin, mMRpars)
```

The output of datain for the above PET data should be as follows:

```
In [5]: datain
Out[5]:
{'#mumapDCM': 192,
 'corepath': '/data/amyloid_brain',
 'lm_bf': '/data/amyloid_brain/LM/17598013_1946_20150604155500.000000.bf',
 'lm_dcm': '/data/amyloid_brain/LM/17598013_1946_20150604155500.000000.dcm',
 'mumapDCM': '/data/amyloid_brain/umap',
 'nrm_bf': '/data/amyloid_brain/norm/17598013_1946_20150604082431.000000.bf',
 'nrm_dcm': '/data/amyloid_brain/norm/17598013_1946_20150604082431.000000.dcm'}
```

The parts of the recognised and categorised input data include:

Type	Path
`corepath`	the core path of the input folder
`#mumapDCM`	the number of DICOM files for the μ-map, usually 192
`mumapDCM`	path to the MR-based DICOM μ-map
`lm_bf`	path to the list-mode binary file
`lm_dcm`	the path to the DICOM header of the list-mode binary file
`nrm_bf`	path to the binary file for component based normalisation
`nrm_dcm`	path to the DICOM header for the normalisation

Note: The raw mMR PET data can also be represented by a single `*.ima` DICOM file instead of the `*.dcm` and `*.bf` pairs for list-mode and normalisation data, which will be reflected in `datain`.

5.3 Specifying output folder

The path to the output folder where the products of *NiftyPET* go, as well as the `verbose` mode can be specified as follows:

```
# output path
opth = os.path.join( datain['corepath'], 'output')

# switch on verbose mode
logging.getLogger().setLevel(logging.DEBUG)
```

With the setting as above, the output folder `output` will be created within the input data folder.

5.4 Obtaining the hardware and object μ-maps

Since MR cannot image the scanner hardware, i.e., the patient table, head and neck coils, etc., the high resolution CT-based mu-maps are provided by the scanner manufacturer. These then have to be appropriately resampled to the table and coils position as used in any given imaging setting. The hardware and object μ-maps are obtained as follow:

```
# obtain the hardware mu-map (the bed and the head&neck coil)
muhdct = nipet.hdw_mumap(datain, [1,2,4], mMRpars, outpath=opth, use_stored=True)

# obtain the MR-based human mu-map
muodct = nipet.obj_mumap(datain, mMRpars, outpath=opth, store=True)
```

The argument [1,2,4] for Obtaining the hardware μ-map correspond to the hardware bits used in imaging, i.e.:

(1) Head and neck lower coil

(2) Head and neck upper coil

(3) Spine coil

(4) Table

Currently, the different parts have to be entered manually (they are not automatically recognised which are in use).

The option `use_stored=True` allows to reuse the already created hardware μ-map, without recalculating it (the resampling can take more than a minute).

Both output dictionaries `muhdct` and `muodct` will contain images among other parameters, such as the image affine matrix and image file paths.

In order to check if both μ-maps were properly loaded, the maps can be plotted together transaxially by choosing the axial index `iz` along the z-axis, as follows:

```
# axial index
iz = 60

# plot image with a colour bar
matshow(muhdct['im'][iz,:,:] + muodct['im'][iz,:,:], cmap='bone')
colorbar()
```

This will produce the following image:

The sagittal image can be generated in a similar way, but choosing the slice along the x-axis, i.e.:

```
# axial index
ix = 170

# plot image with a colour bar
matshow(muhdct['im'][:,:,ix] + muodct['im'][:,:,ix], cmap='bone')
colorbar()
```

5.5 List-mode processing with histogramming

The large list-mode is processed to obtain histogrammed data (sinograms) as well as other statistics on the acquisition, including the head curves and motion detection:

```
hst = nipet.mmrhist(datain, mMRpars)
```

The direct prompt and delayed sinograms can be viewed by choosing the sinogram index below 127 and from 127 up to 836 for oblique sinograms, i.e.:

```
# sinogram index (<127 for direct sinograms, >=127 for oblique sinograms)
si = 60
```

(continues on next page)

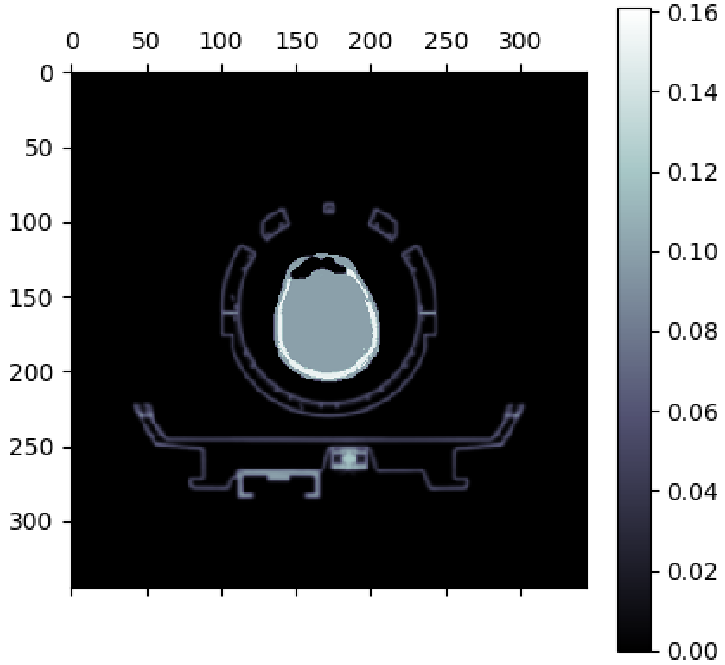

Fig. 5.1: Composite of the hardware and object μ-maps. Observed can be the human head between the upper and lower head&neck coils, and the patient table below.

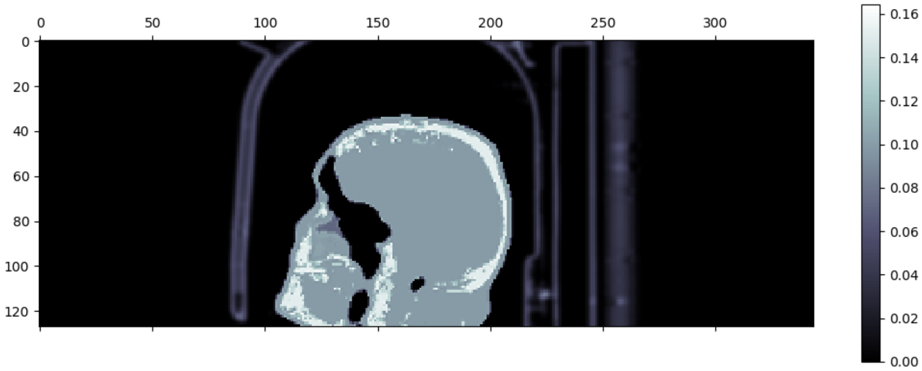

Fig. 5.2: Sagittal view of the composite of the hardware and object μ-maps. Observed can be the human head between the upper and lower head&neck coils, and the patient table on the right of the head.

(continued from previous page)

```
# prompt sinogram
matshow(hst['psino'][si,:,:], cmap='inferno')
colorbar()
xlabel('bins')
ylabel('angles')

# delayed sinogram
matshow(hst['dsino'][si,:,:], cmap='inferno')
colorbar()
xlabel('bins')
ylabel('angles')
```

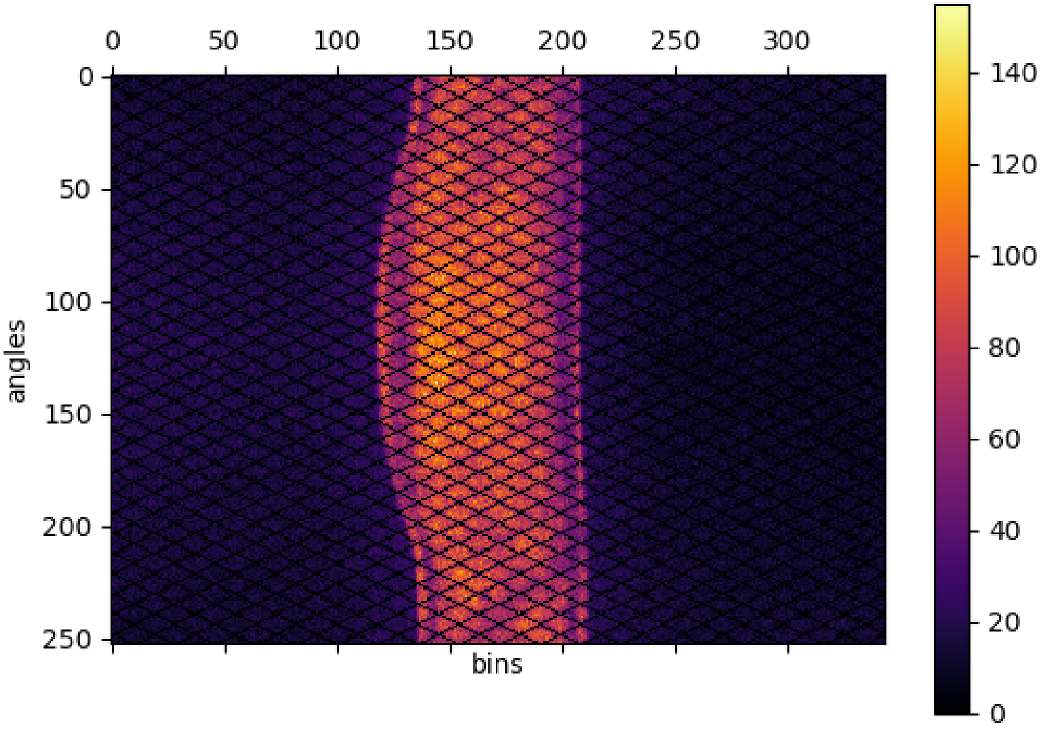

Fig. 5.3: Direct prompt sinogram for 60 minute amyloid PET acquisition.

The head-curve, which is the total number of counts detected per second across the acquisition time, for the prompt and delayed data can be plotted as follows:

```
plot(hst['phc'], label='prompt')
plot(hst['dhc'], label='delayed')
legend()
grid('on')
xlabel('time')
ylabel('counts')
```

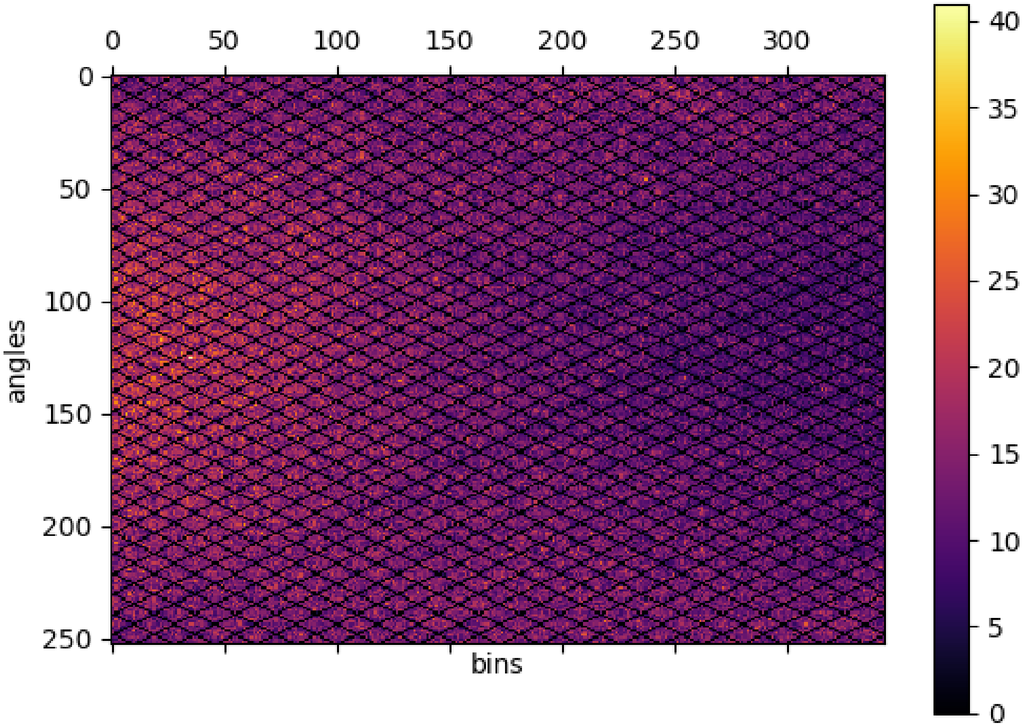

Fig. 5.4: Direct delayed sinogram for 60 minute PET acquisition.

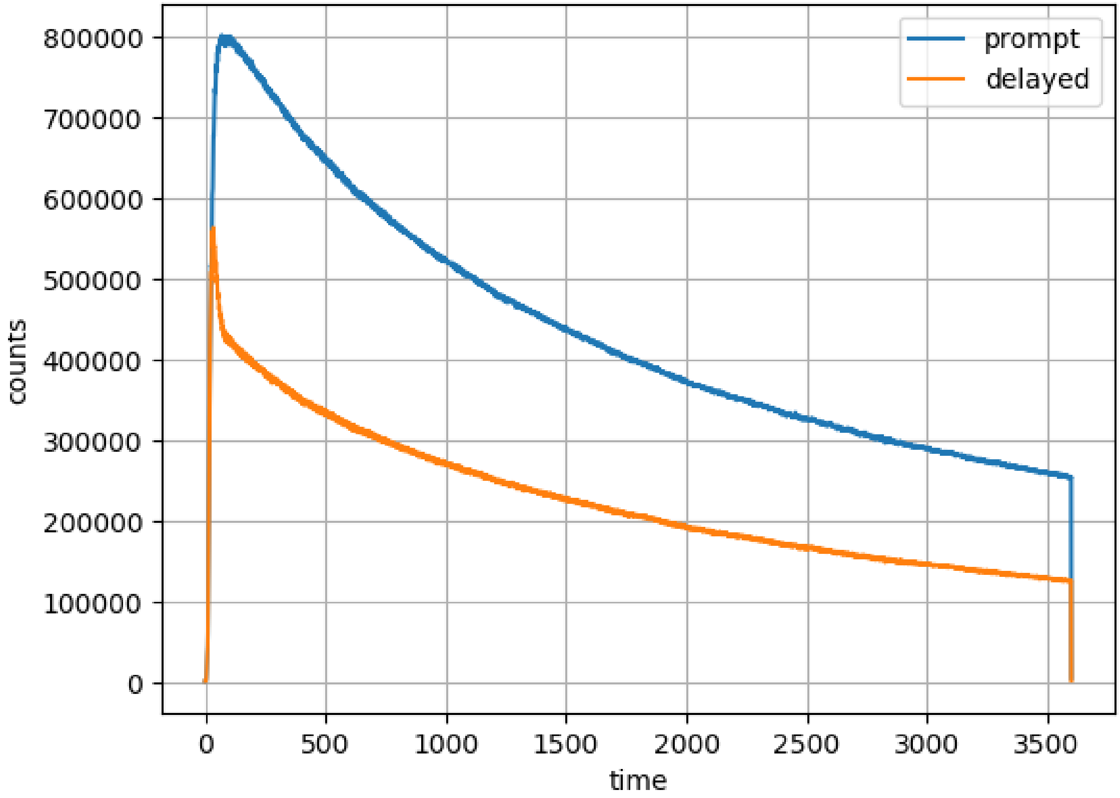

Fig. 5.5: Head curve for prompt and delayed events for the 60-minute acquisition.

In order to get general idea about the potential motion during the acquisition, the centre of mass of the radiodistribution along the axial direction can be plotted as follows:

```
plot(hst['cmass'])
grid('on')
xlabel('time')
ylabel('Centre of mas of radiodistribution')
```

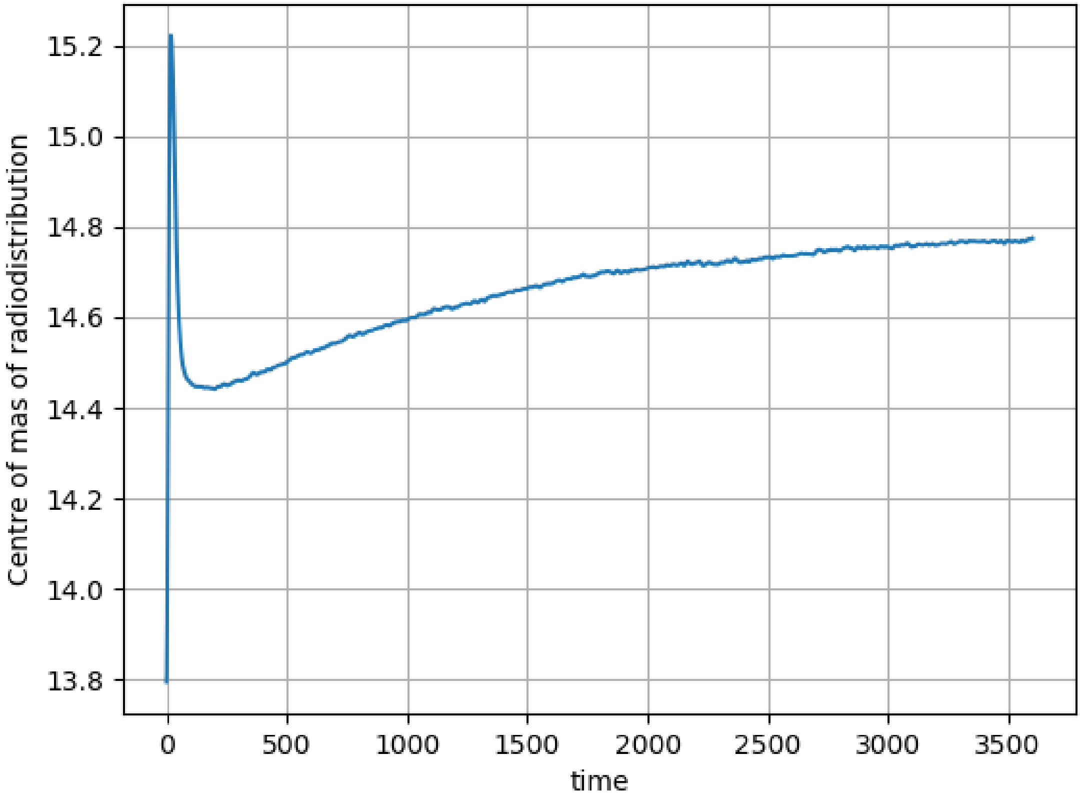

Fig. 5.6: The centre of mass of the radiodistribution for the 60-minute amyloid PET acquisition. Very little motion is observer–the smooth, exponentially varying curve is due to the tracer kinetics.

5.6 Static image reconstruction

The code below provides full image reconstruction for the last 10 minutes of the acquisition to get an estimate of the amyloid load through the ratio image (SUVr).

```
recon = nipet.mmrchain(
    datain, mMRpars,
    frames = ['timings', [3000, 3600]],
    mu_h = muhdct,
    mu_o = muodct,
    itr=4,
```

(continues on next page)

(continued from previous page)

```
    fwhm=0.0,
    outpath = opth,
    fcomment = 'niftypet-recon',
    store_img = True)
```

The input arguments are as follows:

argument	description
`datain`	input data (list-mode, normalisation and the μ-map)
`mMRpars`	scanner parameters (scanner constants and LUTs)
`frames`	definitions of time frame(s);
`mu_h`	hardware μ-map
`mu_o`	object μ-map
`itr`	number of iterations of OSEM (14 subsets).
`fwhm`	full width at half-maximum for the image post-smoothing
`outpath`	path to the output folder
`fcomment`	prefix for all the generated output files
`store_img`	store images (yes/no)

- the argument `timings` indicates that the start/stop times in the following sublist is user-specified and can be done for multiple time frames (see section *Dynamic time frame definitions*).

The reconstructed image can be viewed as follow:

```
matshow(recon['im'][60,:,:], cmap='magma')
colorbar()
```

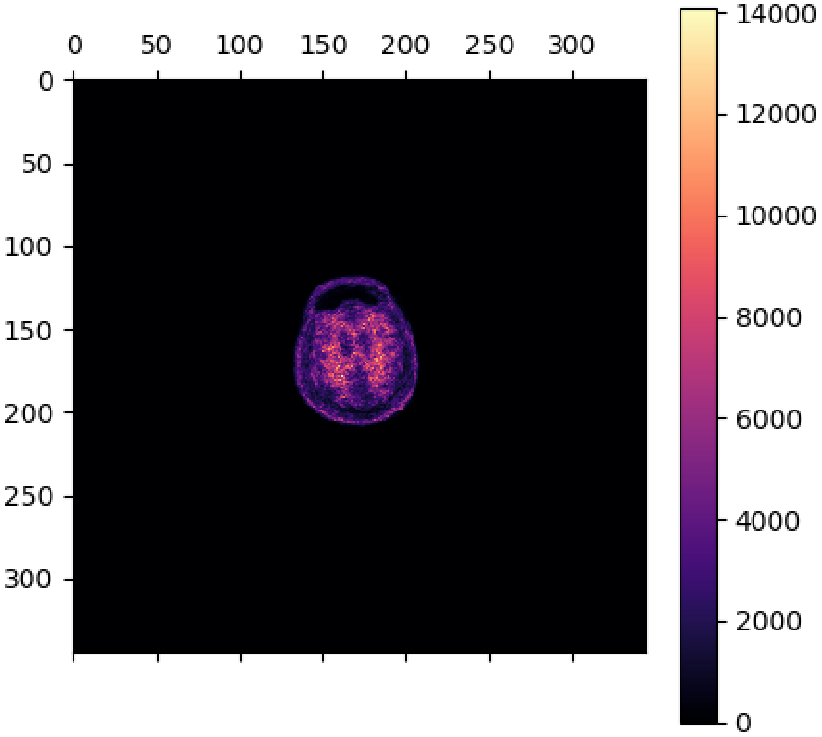

Fig. 5.7: The transaxial slice of the amyloid PET reconstructed image. Voxel intensities are in Bq.

CHAPTER 6

Dynamic image reconstruction

Dynamic imaging involves reconstruction of a series of static images across the whole duration of the acquisition time (usually not less than one hour) by dividing the time into usually non-overlapping dynamic time frames. It is assumed that the average tracer concentration during any frame is representative of the whole frame and any concentration changes within the frame are considered minor relative to the changes across the whole acquisition.

Dynamic reconstruction of the Biograph mMR data uses the same function, `nipet.mmrchain` as shown in section *Static image reconstruction* for the mMR chain processing. In addition to the basic parameters used, additional parameters need to be defined for dynamic reconstruction.

6.1 Dynamic time frame definitions

The extra parameters include the definition of the dynamic time frames. For example, for identical acquisitions, as the provided amyloid PET data (see *Raw brain PET data*), the one-hour dynamic PET data was divided into 31 dynamic frames in [8], i.e.: 4 × 15s, 8 × 30s, 9 × 60s, 2 × 180s, 8 × 300s, which makes the total of 3600s (one hour).

These definitions can then be represented using Python lists in three ways:

1. the simplest way—a 1-D list, with elements representing the consecutive frame durations in seconds, i.e., for the above definitions it will be:

```
frmdef_1D = [
   15,  15,  15,  15,
   30,  30,  30,  30,
   30,  30,  30,  30,
   60,  60,  60,
   60,  60,  60,
   60,  60,  60,
   180, 180,
   300, 300, 300, 300,
   300, 300, 300, 300
   ]
```

2. a more elegant representation is by using a 2-D Python list, starting with a string 'def', followed by two-element sub-lists with the repetition number and the duration in seconds [s], respectively, i.e.:

```
frmdef_2D = ['def', [4, 15], [8, 30], [9, 60], [2, 180], [8, 300]]
```

3. and the last most flexible representation is by using again a 2-D list, starting with a string 'timings' or 'fluid', followed by two-element sub-lists with the start time, t_0 and the end time t_1 for each frame, i.e.:

```
frmdef_t = [
         'timings',
         [0, 15],
         [15, 30],
         [30, 45],
         [45, 60],
         [60, 90],
         [90, 120],
         [120, 150],
         [150, 180],
         [180, 210],
         [210, 240],
         [240, 270],
         [270, 300],
         [300, 360],
         [360, 420],
         [420, 480],
         [480, 540],
         [540, 600],
         [600, 660],
         [660, 720],
         [720, 780],
         [780, 840],
         [840, 1020],
         [1020, 1200],
         [1200, 1500],
         [1500, 1800],
         [1800, 2100],
         [2100, 2400],
         [2400, 2700],
         [2700, 3000],
         [3000, 3300],
         [3300, 3600]
         ]
```

The last representation is the most flexible as it allows the time frames to be defined independent of each other. This is especially useful when defining overlapping time frames or frames which are not necessarily consecutive time-wise.

All the definitions can be summarised in one dictionary using the above `frmdef_1D` or `frmdef_2D` definitions, i.e.:

```
# import the NiftyPET sub-package if it is not loaded yet
from niftypet import nipet

# frame dictionary
frmdct = nipet.dynamic_timings(frmdef_1D)
```

or

```
# frame dictionary
frmdct = nipet.dynamic_timings(frmdef_2D)
```

resulting in:

```
In [1]: frmdct
Out[1]:
{'frames': array([ 15,  15,  15,  15,  30,  30,  30,  30,  30,  30,  30,  30,
↪   60,
         60,  60,  60,  60,  60,  60,  60,  60, 180, 180, 300, 300, 300,
        300, 300, 300, 300, 300], dtype=uint16),
 'timings': ['timings',
  [0, 15],
  [15, 30],
  [30, 45],
  [45, 60],
  [60, 90],
  [90, 120],
  [120, 150],
  [150, 180],
  [180, 210],
  [210, 240],
  [240, 270],
  [270, 300],
  [300, 360],
  [360, 420],
  [420, 480],
  [480, 540],
  [540, 600],
  [600, 660],
  [660, 720],
  [720, 780],
  [780, 840],
  [840, 1020],
  [1020, 1200],
  [1200, 1500],
  [1500, 1800],
  [1800, 2100],
  [2100, 2400],
  [2400, 2700],
  [2700, 3000],
  [3000, 3300],
  [3300, 3600]],
 'total': 3600}
```

Please note, that internally, consecutive dynamic frames are represented as an array of unsigned 16-bit integers.

6.2 Dynamic reconstruction

The dynamic reconstruction can be invoked after the following setting-up and pre-processing:

```
import os
import logging
from niftypet import nipet
from niftypet import nimpa
```

(continues on next page)

(continued from previous page)

```
logging.basicConfig(level=logging.INFO)
# dynamic frames for kinetic analysis
frmdef = ['def', [4, 15], [8, 30], [9, 60], [2, 180], [8, 300]]

# get all the constants and LUTs
mMRpars = nipet.get_mmrparams()

#------------------------------------------------------
# GET THE INPUT
folderin = '/path/to/amyloid_brain'

# recognise the input data as much as possible
datain = nipet.classify_input(folderin, mMRpars)
#------------------------------------------------------

# switch on verbose mode
logging.getLogger().setLevel(logging.DEBUG)

# output path
opth = os.path.join( datain['corepath'], 'output')

#------------------------------------------------------
# GET THE MU-MAPS
muhdct = nipet.hdw_mumap(datain, [1,2,4], mMRpars, outpath=opth, use_
↪stored=True)

# UTE-based object mu-map
muodct = nipet.obj_mumap(datain, mMRpars, outpath=opth, store=True)
#-----------------------------------------------------
```

Since multiple image frames are reconstructed, the `mmrchain` function apart from 4-D NIfTI image storing, also enables to store intermediate 3-D NIfTI images for each dynamic frame using the option `store_img_intrmd = True`:

```
recon = nipet.mmrchain(
         datain,
         mMRpars,
         frames = frmdef,
         mu_h = muhdct,
         mu_o = muodct,
         itr = 4,
         fwhm = 0.,
         outpath = opth,
         fcomment = '_dyn',
         store_img = True,
         store_img_intrmd = True)
```

The path to the reconstructed 4-D image can be accessed through the output dictionary, `recon`:

```
In [2]: recon['im'].shape
Out[2]: (31, 127, 344, 344)
```

or the stored 4-D NIfTI image in:

```
In [19]: recon['fpet']
Out[19]: '/path/to/NIfTI-output'
```

6.3 Time offset due to injection delay

In most cases the injection is performed with some delay relative to the time of starting the scan. Also, the first recorded counts will be random events, as the activity is detected from outside the field of view (FOV). The offset caused by the injection delay and random events, can be separated and omitted by running first list mode processing with histogramming, followed by estimating the time offset and accounting for it in the timings of the frames:

```
# histogram the list mode data (in <datain> dictionary) using scanner parameters (<mMRpars>)
hst = nipet.mmrhist(datain, mMRpars)

# offset for the time from which meaningful events are detected
toff = nipet.lm.get_time_offset(hst)

# dynamic frame timings
frm_timings = nipet.lm.dynamic_timings(frmdef, offset=toff)
```

The reconstruction is then performed with the augmented timings in the following way:

```
recon = nipet.mmrchain(
         datain,
         mMRpars,
         frames = frm_timings,
         mu_h = muhdct,
         mu_o = muodct,
         itr = 4,
         fwhm = 0.,
         outpath = opth,
         fcomment = '_dyn',
         store_img = True,
         store_img_intrmd = True)
```

Note, that the reconstruction pipeline accepts different definitions of the dynamic frames as shown above.

6.4 Visualisation of dynamic frame timings

The time frames can be visualised with one line of code:

```
# draw the frame timings over the head-curve
nipet.lm.draw_frames(hst, frm_timings)
```

which plots the timings of dynamic frames over the prompts, delayeds and the difference between the two as shown below in Fig. 6.1.

In order to zoom in to a particular time interval, e.g., from 0s to 150s, and visualise clearly the time offset, the following line of code can be used:

```
# draw the frame timings over the head-curve, with time limits of tlim
nipet.lm.draw_frames(hst, frm_timings, tlim = [0, 150])
```

resulting in the following plot (Fig. 6.2):

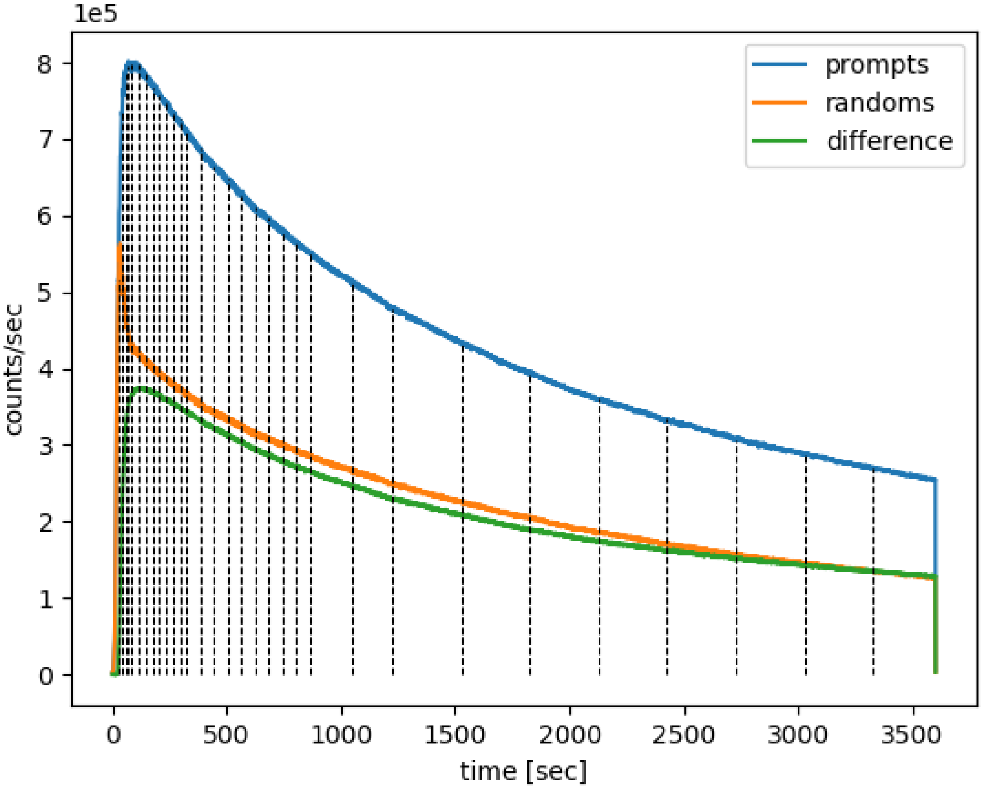

Fig. 6.1: The dynamic frame intervals are marked with dashed horizontal black curves on top of the head-curve (prompts and delayeds per second).

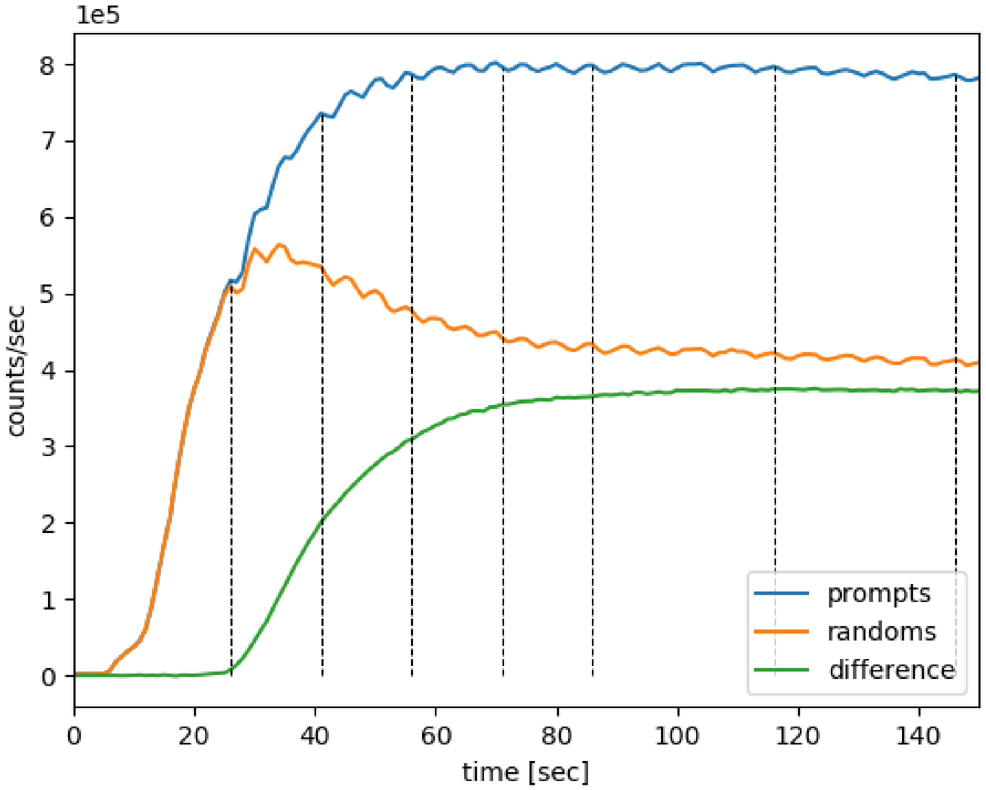

Fig. 6.2: The dynamic frame intervals are marked with dashed horizontal black curves on top of the part of head-curve (prompts and delayeds per second). Note that almost 30s of list mode data from the beginning is discarded.

CHAPTER 7

Corrections for quantitative PET

7.1 Decay correction

During any PET acquisition, the radio-tracer activity, A_t, decays according to:

$$A_t = A_0 e^{-\lambda t},$$

where A_0 is the reference activity at time t_0. The decay constant λ is defined as:

$$\lambda = \frac{\ln(2)}{T_{1/2}},$$

where $T_{1/2}$ is the half-life of the radionuclide.

Likewise, for activity A_t at time t, the original activity is simply:

$$A_0 = A_t e^{\lambda t}.$$

7.1.1 Real PET acquisition example

Consider the one-hour dynamic amyloid PET data provided in *Raw brain PET data*. The injected radioactivity at $t_0 = 0$ was $A_0 = 409$ MBq, and since the used amyloid tracer is ^{18}F-based, the half life for the radioisotope is $T_{1/2} = 6586.272$ s (≈ 110 min), and hence $\lambda = 1.052 \times 10^{-4}$ s^{-1}. The radioactive decay from the injected activity is shown in black in Fig. 7.1.

Consider also a time frame of the last 10 minutes of acquisition, from $t_1 = 3000$ to $t_2 = 3500$ seconds, as shown in Fig. 7.1. In order to correct for the decay not only within the time frame, but also relative to the beginning of the scan at injection, the measurable activity of the time frame needs to be compared to the ideal case of no radioactive decay.

In this example, the ideal radioactivity would remain constant at $A_0 = 409$ MBq, as is shown by the horizontal dashed line. Therefore, for the considered duration of the time frame, the measurable ideal activity would be $A_0 \Delta t$.

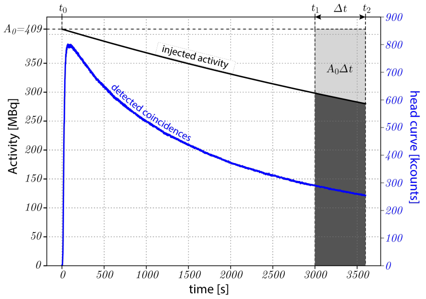

Fig. 7.1: Radioactive decay, shown in black, from the injected activity of $A_0 = 409$ MBq. The recorded prompt events are shown in blue. The decay correction calculations are shown for a frame of the last 10 minutes of acquisition. **Please note** that the injected activity is significantly greater than the recorded events (coincidences) as the field of view of the scanner allows only part of the body to be scanned. **Note** also the significantly different distribution of the detected coincidences, which is decaying considerably faster than the injected activity, and which is due to tracer clearance from the participant's head, dead-time and other factors.

In the real case scenario, however, the activity decays, and the measurable activity with the time frame is:

$$\begin{aligned}\int_{t_1}^{t_2} A_0 e^{-\lambda t}\mathrm{d}t &= -\tfrac{1}{\lambda}e^{-\lambda t}\Big|_{t_1}^{t_2}\\ &= \frac{A_0}{\lambda}\left(e^{-\lambda t_1} - e^{-\lambda t_2}\right)\end{aligned}$$

The decay correction, $C_{\text{decay}}^{t_0}$, for the time frame relative to the beginning of scan (injection), $t_0 = 0$, is simply the ratio of the two activities, i.e., the ideal one to the decaying, Δt, of the frame:

$$\begin{aligned}C_{\text{decay}}^{t_0} &= \frac{A_0\Delta t}{A_0(e^{-\lambda t_1} - e^{-\lambda t_2})/\lambda}\\ &= \frac{\lambda\Delta t}{e^{-\lambda t_1}(1 - e^{-\lambda\Delta t})},\end{aligned}$$

and finally obtaining:

$$C_{\text{decay}}^{t_0} = \frac{\lambda e^{\lambda t_1}\Delta t}{1 - e^{-\lambda\Delta t}}.$$

See also http://www.turkupetcentre.net/petanalysis/decay.html.

7.1.2 Controlling decay correction in *NiftyPET*

The decay correction is by default applied automatically with the reference to the beginning of scan as recorded in the list-mode data. It does not need to be the injection time, i.e., in case of static scans, when the patient waits a time post-injection before is scanned. In *NiftyPET* decay correction is controlled by the dictionary entry `Cnt['DCYCRR']`. For example, if the scanner is initialised as follows:

```
# NiftyPET image reconstruction package (nipet)
from niftypet import nipet
# NiftyPET image manipulation and analysis (nimpa)
from niftypet import nimpa

# get all the Biograph mMR parameters
mMRpars = nipet.get_mmrparams()
```

Then the default decay correction can be switched off, if the following line:

```
mMRpars['Cnt']['DCYCRR'] = False,
```

is placed before image reconstruction. By default `mMRpars['Cnt']['DCYCRR'] = True`.

CHAPTER 8

Raw brain PET data

Downloadable raw PET data of a single brain amyloid scan (^{18}F-florbetapir), with all the necessary input components for independent image reconstruction, can be obtained from the research data repository **Zenodo** (click the DOI link to download the data) [1]:

The downloaded zip file contains a full dynamic list-mode PET data acquired on a Siemens Biograph mMR for 60 minutes, using amyloid tracer ^{18}F-florbetapir, provided by Avid Radiopharmaceuticals, Inc., a wholly owned subsidiary of Lilly. The file also includes normalisation files and the μ-map based on MR Ultrashort TE (UTE) sequence [2][3], all three parts needed for an independent image reconstruction using NiftyPET.

8.1 Citing the data

When using this data set for research or publications, please cite the following two publications: [1][4]:

- Markiewicz, P. J., Cash, D., & Schott, J. M. (2018, November 5). Single amyloid PET scan on the Siemens Biograph mMR. London: Zenodo. https://doi.org/10.5281/ZENODO.1472951
- Lane, C. A., Parker, T. D., Cash, D. M., Macpherson, K., Donnachie, E., Murray-Smith, H., . . . Schott, J. M. (2017). Study protocol: Insight 46 - a neuroscience sub-study of the MRC National Survey of Health and Development. BMC Neurology, 17(1), 75. https://doi.org/10.1186/s12883-017-0846-x

If the data is reconstructed and/or analysed using *NiftyPET*, additionally please cite [5]:

- Markiewicz, P. J., Ehrhardt, M. J., Erlandsson, K., Noonan, P. J., Barnes, A., Schott, J. M., . . . Ourselin, S. (2018). NiftyPET: a High-throughput Software Platform for High Quantitative Accuracy and Precision PET Imaging and Analysis. Neuroinformatics, 16(1), 95–115. https://doi.org/10.1007/s12021-017-9352-y

CHAPTER 9

References

CHAPTER 10

Acknowledgements

This project is being developed at University College London (UCL) at the department of Medical Physics and Biomedical Engineering and the Centre for Medical Image Computing (CMIC). Initially, it was supported and funded by the Engineering and Physical Sciences Research Council (EPSRC) of the United Kingdom (UK). Currently, the project is being further developed under the following funding streams:

1. The Innovative Medicines Initiative 2 Joint Undertaking under grant agreement No 115952. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and EFPIA.
2. The Dementias Platform UK MR-PET Partnership, supported by the Medical Research Council (MRC) in the UK.

I gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K20 and Titan X Pascal GPUs used for this research and work.

AMYPAD Amyloid imaging to prevent Alzheimer´s Disease	Dementias Platform UK Medical Research Council	
efpia	imi innovative medicines initiative	
MRC Medical Research Council	EPSRC Engineering and Physical Sciences Research Council	logos/Nvidia_logo.png

Bibliography

[1] Pawel J Markiewicz, David Cash, and Jonathan M Schott. Single amyloid PET scan on the Siemens Biograph mMR. nov 2018. URL: https://zenodo.org/record/1472951\protect\T1\textbraceleft\T1\textbackslash{}# \protect\T1\textbraceright.XBbW49\protect\T1\textbraceleft\T1\textbackslash{}_\protect\T1\textbracerightnjx0, doi:10.5281/ZENODO.1472951.

[2] Matthew D Robson, Peter D Gatehouse, Mark Bydder, and Graeme M Bydder. Magnetic Resonance: An Introduction to Ultrashort TE (UTE) Imaging. 2003. URL: https://insights.ovid.com/pubmed?pmid=14600447, arXiv:NIHMS150003, doi:10.1097/00004728-200311000-00001.

[3] Vincent Keereman, Yves Fierens, Tom Broux, Yves De Deene, Max Lonneux, and Stefaan Vandenberghe. MRI-Based Attenuation Correction for PET/MRI Using Ultrashort Echo Time Sequences. *Journal of Nuclear Medicine*, 51(5):812–818, may 2010. URL: http://www.ncbi.nlm.nih.gov/pubmed/20439508http://jnm.snmjournals.org/cgi/doi/10.2967/jnumed.109.065425, doi:10.2967/jnumed.109.065425.

[4] Christopher A. Lane, Thomas D. Parker, Dave M. Cash, Kirsty Macpherson, Elizabeth Donnachie, Heidi Murray-Smith, Anna Barnes, Suzie Barker, Daniel G. Beasley, Jose Bras, David Brown, Ninon Burgos, Michelle Byford, M. Jorge Cardoso, Ana Carvalho, Jessica Collins, Enrico De Vita, John C. Dickson, Norah Epie, Miklos Espak, Susie M.D. Henley, Chandrashekar Hoskote, Michael Hutel, Jana Klimova, Ian B. Malone, Pawel Markiewicz, Andrew Melbourne, Marc Modat, Anette Schrag, Sachit Shah, Nikhil Sharma, Carole H. Sudre, David L. Thomas, Andrew Wong, Hui Zhang, John Hardy, Henrik Zetterberg, Sebastien Ourselin, Sebastian J. Crutch, Diana Kuh, Marcus Richards, Nick C. Fox, and Jonathan M. Schott. Study protocol: Insight 46 - a neuroscience sub-study of the MRC National Survey of Health and Development. *BMC Neurology*, 17(1):75, dec 2017. URL: http://bmcneurol.biomedcentral.com/articles/10.1186/s12883-017-0846-x, doi:10.1186/s12883-017-0846-x.

[5] Pawel J. Markiewicz, Matthias J. Ehrhardt, Kjell Erlandsson, Philip J. Noonan, Anna Barnes, Jonathan M. Schott, David Atkinson, Simon R. Arridge, Brian F. Hutton, and Sebastien Ourselin. NiftyPET: a High-throughput Software Platform for High Quantitative Accuracy and Precision PET Imaging and Analysis. *Neuroinformatics*, 16(1):95–115, jan 2018. URL: http://link.springer.com/10.1007/s12021-017-9352-y, doi:10.1007/s12021-017-9352-y.

[6] P J Markiewicz, K Thielemans, J M Schott, D Atkinson, S R Arridge, B F Hutton, and S Ourselin. Rapid processing of PET list-mode data for efficient uncertainty estimation and data analysis. *Physics in Medicine and Biology*, 61(13):N322–N336, jul 2016. URL: http://stacks.iop.org/0031-9155/61/i=13/a=N322?key=crossref.817e85ee0602e878b69ace94c186e0cb, doi:10.1088/0031-9155/61/13/N322.

[7] European Society of Radiology European Society of Radiology (ESR). The new EU General Data Protection Regulation: what the radiologist should know. *Insights into imaging*, 8(3):295–299, jun 2017. URL: http://www.

ncbi.nlm.nih.gov/pubmed/28439718http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5438318, doi:10.1007/s13244-017-0552-7.

[8] Catherine J Scott, Jieqing Jiao, Andrew Melbourne, Ninon Burgos, David M Cash, Enrico De Vita, Pawel J Markiewicz, Antoinette O'Connor, David L Thomas, Philip Sj Weston, Jonathan M Schott, Brian F Hutton, and Sébastien Ourselin. Reduced acquisition time PET pharmacokinetic modelling using simultaneous ASL–MRI: proof of concept. *Journal of Cerebral Blood Flow & Metabolism*, pages 0271678X1879734, 2018. URL: https://journals.sagepub.com/doi/pdf/10.1177/0271678X18797343http://journals.sagepub.com/doi/10.1177/0271678X18797343, doi:10.1177/0271678X18797343.